Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Descrição da Implementação dos Filtros para Recuperação da Taxa de Erro dos Algoritmos de Aprendizado de Máquina Usados no Ambiente Discover

André Henrique Kemp

Gustavo Enrique de Almeida Prado Alves Batista

Maria Carolina Monard

N. 175

RELATÓRIOS TÉCNICOS DO ICMC

USP – São Carlos

Setembro de 2002

Descrição da Implementação dos Filtros para Recuperação da Taxa de Erro dos Algoritmos de Aprendizado de Máquina Usados no Ambiente Discover[1]

André Henrique Kemp
Gustavo Enrique de Almeida Prado Alves Batista
Maria Carolina Monard

Universidade de São Paulo/ILTC
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística
Caixa Postal 668, 13560-970 - São Carlos, SP
e-mail:{akemp, gbatista, mcmonard}@icmc.usp.br

Sumário

O sistema *Discover*, que vem sendo implementado no Laboratório de Inteligência Computacional – LABIC – do ICMC - USP, tem a finalidade de auxiliar no processo de descoberta de conhecimento em Base de Dados. Para permitir uma maior flexibilidade do sistema, foi desenvolvida uma sintaxe padrão para a representação tanto do arquivo de atributos quanto do arquivo de dados utilizados por algoritmos de aprendizado de máquina simbólico [Batista and Monard, 2002]. Essa sintaxe permite uma maior integração entre os diversos algoritmos de aprendizado de máquina simbólico que integram o sistema, tais como C4.5, C4.5rules, CN2, entre outros.

O *Discover* possui também a opção de trabalhar com os arquivos de dados na forma de tabelas do Sistema de Base de Dados MySQL [Axmark et al., 2000]. Para isso foi criado um método que carrega os arquivos de dados, na sintaxe padrão, para tabelas do MySQL.

Este trabalho tem por objetivo apresentar a implementação de mais alguns *scripts* que compõem o sistema *Discover*, os quais incluem a implementação do método que faz a carga dos arquivos de dados para o MySQL e a implementação de filtros que recuperam a taxa de erro gerada por cada um dos algoritmos de Aprendizado de Máquina Simbólico.

Setembro de 2002

[1]Trabalho realizado com auxílio da FAPESP.

Sumário

1 Introdução **1**

2 MySQL **1**

3 DBI **2**

4 Sintaxe da Linguagem do Ambiente *Discover* **4**

4.1 Visão Geral 5

4.2 Tipos de Dados 5

4.3 Gramática da Sintaxe Padrão 6

5 Biblioteca do Sistema *Discover* **7**

6 *Script* para a Carga do *Dataset* no MySQL (*MySQLLoad*) **8**

7 *Scripts* Implementados para Recuperação das Taxas de Erro **10**

7.1 Filtro para o C4.5 (*fc45*) 10

7.2 Filtro para o C4.5rules (*fc45r*) 13

7.3 Filtro para o CN2 (*fcn2*) 13

8 Conclusão **19**

Lista de Figuras

1 Arquitetura de uma Aplicação DBI [DBI, 2001] 3

2 Algoritmo `MySQLLoad` 9

3 Exemplo de Saída do Algoritmo C4.5 11

4 Algoritmo `fc45` 12

5 Exemplo de Saída do Algoritmo C4.5rules 15

6 Algoritmo `fc45r` 16

7 Exemplo de Saída do Algoritmo CN2 17

8 Algoritmo `fcn2` 18

1 Introdução

O sistema *Discover* [Baranauskas and Batista, 2000], que vem sendo implementado no Laboratório de Inteligência Computacional – LABIC – do ICMC - USP, tem a finalidade de auxiliar no processo de descoberta de conhecimento em Base de Dados. Para permitir uma maior flexibilidade do sistema, foi desenvolvida uma sintaxe padrão para a representação tanto do arquivo de atributos quanto do arquivo de dados utilizados por algoritmos de Aprendizado de Máquina Simbólico (AM) [Batista and Monard, 2002]. Essa sintaxe permite uma maior integração entre os diversos algoritmos de AM que integram o sistema (C4.5, C4.5rules, CN2, entre outros). Além disso, também foi desenvolvida uma biblioteca orientada a objetos implementada na linguagem de programação PERL [Till, 1996], para realizar, entre outras coisas, a conversão dos arquivos da sintaxe desses diversos algoritmos para a sintaxe padrão utilizada pelo *Discover*.

O *Discover* possui também a opção de trabalhar com os arquivos de dados na forma de tabelas do Sistema de Base de Dados MySQL [Axmark et al., 2000]. Para isso foi criado um método que carrega os arquivos de dados, na sintaxe padrão, para tabelas do MySQL.

Este trabalho tem por objetivo apresentar a implementação de alguns *scripts* que compõem o sistema *Discover*. Estes *scripts* incluem a implementação do método que faz a carga dos arquivos de dados para o MySQL e a implementação de filtros que recuperam a taxa de erro gerada por cada um dos algoritmos de Aprendizado de Máquina Simbólico.

O trabalho começa com uma apresentação do Sistema de Base de Dados MySQL (Seção 2). A seguir, é descrito o módulo DBI que faz a comunicação do MySQL com a linguagem de programação PERL (Seção 3). Na Seção 4 é feita uma breve descrição da sintaxe padrão definida para o sistema *Discover*, seguida pela apresentação da biblioteca implementada para o sistema *Discover* (Seção 5). Finalmente, nas Seções 6 e 7, são mostradas as implementações dos scripts desenvolvidos, seguidas por uma seção que apresenta as conclusões do trabalho.

2 MySQL

Na definição do sistema *Discover* optou-se pelo uso da ferramenta MySQL [MySQL, 1995] na implementação do sistema. Mas, o que é MySQL? MySQL pode ser visto sob três pontos de vista diferentes:

1. **Sistema Gerenciador de Base de Dados:** Uma base de dados é uma coleção estruturada de dados. Para adicionar, acessar e processar dados armazenados em uma base de dados é necessário uma Sistema Gerenciados de Base de Dados – SGBD – como o MySQL. Tal sistema pode ter um papel central tanto quanto um utilitário *stand alone*, ou simplesmente fazer parte de outras aplicações.

2. **Base de Dados Relacional:** Uma base de dados relacional armazena os dados em tabelas separadas ao invés de colocar todos os dados em uma única tabela. Isso permite velocidade e flexibilidade. As tabelas são ligadas por **relações**, que possibilitam combinar dados de várias tabelas.

3. **Software de Código Aberto:** Código aberto significa que qualquer usuário pode usá-lo e modificá-lo. Qualquer pessoa pode fazer o *download* do MySQL da Internet e usá-lo sem pagar nada. O código pode ser alterado por qualquer usuário para que o *software* se adeque às sua necessidades.

O desenvolvimento do MySQL começou com a necessidade, vista por um grupo, de usar o mSQL [Finnin, 2001] para conectar suas bases de dados usando rotinas rápidas, de baixo nível, por eles implementadas. Contudo, após alguns testes concluíram que o mSQL não era rápido nem flexível o suficiente para suas necessidades. Daí surgiu uma nova interface para SQL conhecida hoje como MySQL.

O MySQL apresenta uma arquitetura cliente/servidor multiplataforma com suporte a uma série de linguagens, inclusive à linguagem PERL [Till, 1996], usada no desenvolvimento do ambiente computacional, dentro do qual se insere este trabalho. Essa comunicação se dá através de API's (*Application Programming Interface*). Em particular, uma das API's responsáveis por fazer a comunicação com a linguagem PERL chama-se DBI. Uma discussão mais detalhada a respeito desta API será apresentada na Seção 3. Maiores detalhes sobre o MySQL podem ser encontradas em [Axmark et al., 2000].

São esses motivos discutidos anteriormente, sobretudo por ser um software de livre distribuição, que levaram à escolha deste pacote de Base de Dados para o desenvolvimento do projeto *Discover*.

3 DBI

DBI [DBI, 2001] é um módulo de acesso a base de dados para a linguagem de programação PERL. Ele define um conjunto de métodos, variáveis e convenções que fornecem uma interface consistente com a base de dados, independente da base de dados sendo usada (que no caso deste trabalho é o MySQL).

O DBI é só uma interface, uma camada de *cola* entre uma aplicação e um ou mais *drivers* de bases de dados. São os *drivers* que fazem a maior parte do trabalho real. Cabe ao DBI somente prover uma interface padrão para os *drivers* operarem. A Figura 1 mostra a arquitetura de uma aplicação DBI. O DBI envia as chamadas de métodos para o *driver* de execução apropriado e é também responsável pela carga dinâmica de *drivers*, checagem e manuseio de erros, fornecimento de implementações *default* para os métodos além de muitos outros serviços não específicos das bases de dados.

Para mostrar como isso acontece, discutem-se a seguir os métodos mais importantes dessa interface. Maiores detalhes podem ser encontrados em [DBI, 2001].

Antes de usar o DBI é necessário primeiro carregá-lo. Isso é feito com o comando:

```
use DBI;
```

A seguir é preciso fazer a conexão com uma base de dados, através do seguinte comando:

```
$dbh = DBI->connect($dsn, $user, $password);
```

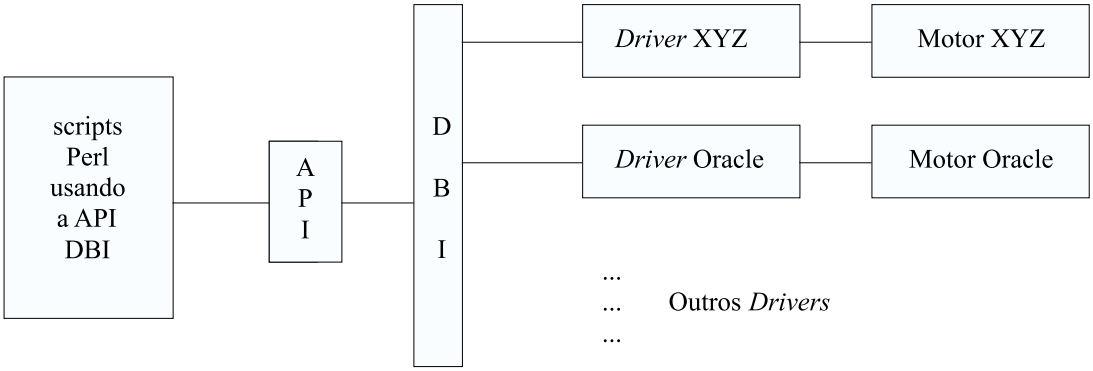

Figura 1: Arquitetura de uma Aplicação DBI [DBI, 2001]

onde:

> `$dbh` é um objeto, ou um *handle* para a conexão;
> `$dsn` é o nome da base de dados a que se deseja conectar;
> `$user` é o nome do usuário que realiza a conexão e;
> `$password` é a senha desse usuário.

Vale destacar que o usuário `$user` deve estar cadastrado no Gerenciador de Base de Dados MySQL, sendo que o cadastro é feito pelo gerente do sistema (root). A senha é um campo opcional.

Como o custo da conexão pode ser muitas vezes computacionalmente caro, é usual estabelecer a conexão no começo da execução do programa e a desconexão somente no fim da execução do mesmo. Para a desconexão da base é usado o seguinte método:

```
$dbh->disconnect;
```

O DBI ainda permite que uma aplicação prepare declarações SQL para execução posterior. Tais declarações são identificadas como um *handle de declaração* e ficam armazenadas em uma variável PERL, a qual, por convenção, é chamada de `$sth`. Por exemplo, o código para uma operação de inserção em série poderia ser:

```
1 - $sth = $dbh->prepare("INSERT INTO table(foo,bar,baz) VALUES (?,?,?)");

2 - $sth->execute($foo, $bar, $baz );
```

onde a primeira linha de código é responsável pelo preparo da declaração e a segunda linha pela execução da linha preparada. Os símbolos de interrogação ("?") presentes na linha de preparo são substituídos pelos valores das variáveis passadas como parâmetros na linha de execução, respeitando a ordem de passagem dos parâmetros (primeiro símbolo "recebe" a primeira variável, segundo símbolo "recebe" a segunda variável e assim por diante).

É importante salientar que as operações não estão confirmadas na base de dados após serem realizadas. Em outras palavra, é necessário confirmar todas as operações realizadas para que sejam finalmente sedimentadas. Isso é feito através do método:

```
$dbh->commit;
```

Para desfazer uma mudança que ainda não foi confirmada usa-se o método:

```
$dbh->rollback;
```

para desfazer as mudanças.

Para se ter acesso aos resultados de uma operação, como por exemplo o resultado de uma instrução SQL `SELECT`, o módulo oferece alguns métodos. Esses métodos consistem de métodos que recuperam uma coluna da tabela até métodos que recuperam todos os dados gerados por uma declaração. Discute-se a seguir o método que recupera todo os dados de uma única vez. Sua declaração é feita da seguinte forma:

```
$tbl_ary_ref = $sth->fetchall_arrayref;
```

O `fetchall_arrayref` é usado para "carregar" todos os dados retornados após uma declaração preparada e executada. Ele retorna uma referência para um vetor que contém uma referência para cada entrada (linha) da tabela obtida como resultado da execução de uma declaração. Se não há entradas, o método retorna um vetor vazio (ou uma referência para um vetor vazio).

Esse método ainda apresenta uma certa flexibilidade quanto aos dados a serem recuperados. Por exemplo, para recuperar só a primeira coluna de todas as entradas obtidas tem-se um comando da forma:

```
$tbl_ary_ref = $sth->fetchall_arrayref([0]);
```

Para recuperar a penúltima e a última coluna de todas as entradas têm-se o seguinte comando:

```
$tbl_ary_ref = $sth->fetchall_arrayref([-2,-1]);
```

Já para recuperar os campos chamados `foo` e `bar` de todas as entradas, tratando-os como uma referência para um hash, o comando é o seguinte:

```
$tbl_ary_ref = $sth->fetchall_arrayref({ foo=>1, bar=>1 });
```

Nesta seção foram descritos os principais métodos do DBI que serão usados na comunicação dos *scripts* PERL, discutidos neste trabalho, juntamente com o gerenciador de base de dados MySQL. A seguir, é apresentada a sintaxe desenvolvida para os arquivos do ambiente *Discover*.

4 Sintaxe da Linguagem do Ambiente *Discover*

O sistema *Discover* visa, entre outras coisas, integrar alguns dos algoritmos mais conhecidos de Aprendizado de Máquina Simbólico em uma única ferramenta, além de realizar estudos comparativos entre eles. Contudo, os diversos algoritmos de classificação usados (C4.5, C5.0, CN2, entre outros) possuem sintaxes diferentes para seus arquivos de entrada, tanto o arquivo de atributos quanto o de dados. Isso faz necessária a definição de uma sintaxe padrão para representar os arquivos que serão usados nesse ambiente. Uma vez gerados tais arquivos nessa sintaxe padrão, o ambiente possui filtros para converter os arquivos dessa sintaxe para a sintaxe de um algoritmo em particular (sintaxe padrão para a sintaxe do C4.5, por exemplo). O ambiente também possui filtros para realizar o caminho inverso (sintaxe do C4.5 para a sintaxe padrão). A seguir é

descrito sucintamente como foi definida essa sintaxe pelo grupo de desenvolvimento do *Discover*. Maiores detalhes, como extensões da sintaxe e outras características podem ser encontrados em [Batista, 2001].

4.1 Visão Geral

A sintaxe padrão utiliza arquivos do tipo texto para declarar os atributos (e seus respectivos tipos) e os valores que esses atributos assumem em um dado conjunto de exemplos. Os atributos são declarados em um arquivo com a extensão `.names`. Os valores que esses atributos assumem em um conjunto de exemplos são declarados em um outro arquivo com a extensão `.data`. Os dois arquivos devem possuir o mesmo nome, se diferenciando apenas pela extensão.

A primeira declaração em um arquivo `.names` define qual deve ser o atributo classe. O atributo classe pode ser qualquer atributo presente no conjunto de exemplos. Após a declaração do atributo classe, são declarados os demais atributos. Cada atributo possui um identificador e um tipo de dado associado a ele. São válidos os identificadores que são combinações de números, letras e "_" (underscore), em qualquer seqüência. Para identificadores mais complexos que envolvem outros caracteres que não sejam os especificados anteriormente (como espaços, letras acentuadas, etc) é necessário colocar o identificador entre aspas. Desta forma, são identificadores válidos: abc, 1, 1a, _1a, " _12a ", "válido".

No arquivo `.data` são declarados os valores que os atributos presentes no arquivos `.names` assumem para um conjunto de exemplos. Cada linha de um arquivo `.data` representa um exemplo. Desta forma, o "separador de registros" (*record separator*) é o caracter de nova linha (representado em muitas linguagem de programação por `\n`. Cada linha possui uma seqüência de valores separados por vírgula, ou seja, a vírgula é o separador de campos (*field separator*). Cada valor presente em uma linha está associado a um atributo do arquivo `.names`. Sendo assim, a ordem em que os valores são declarados em uma determinada linha deve ser a mesma ordem na qual os atributos foram declarados no arquivo `.names`.

4.2 Tipos de Dados

Foram definidos os seguintes tipos de dados, que podem ser associados aos identificadores de atributos [Batista and Monard, 2002] :

- Nominal: O tipo `nominal` é utilizado para declarar um atributo que pode assumir um grupo restrito de valores.
- Enumerated: O tipo de dado `enumerated` é muito semelhante ao tipo de dado `nominal`. A principal diferença é que com o tipo `enumerated` é possível identificar uma ordem entre os valores que o atributo pode assumir. Entretanto, não existe uma definição clara de distância entre esses valores. Um exemplo de tipo `enumerated` é um atributo que pode assumir os valores `pequeno, médio` e `grande`.
- Integer: O tipo de dado `integer` é utilizado para declarar um atributo que pode assumir um valor inteiro.

- Real: O tipo `real` é semelhante ao tipo de dado `integer`, com a diferença que um atributo `real` pode armazenar números com ou sem parte fracionária.
- String: Um atributo `string` pode assumir como valor um string de tamanho indefinido que pode conter quaisquer caracteres incluindo quebra de linha (`\n`). Para identificar os limites de um string é necessário inserir o símbolo de aspas (“) no início e no fim do string.
- Date: O tipo de dado `date` permite declarar um atributo que pode conter uma data (dia, mês e ano). A princípio, os valores das datas devem estar no formato `mm/dd/aaaa` (formato americano).
- Time: O tipo de dado `time` permite declarar um atributo que pode conter um horário (hora, minuto e segundo). A princípio, os valores dos horários devem estar no formato `hh:mm:ss`.

4.3 Gramática da Sintaxe Padrão

A gramática que define a sintaxe do arquivo `.names` é a seguinte:

```
S ::= <class-defs> | <feature-defs>

<class-defs> ::= <feature-name> . | null .

<feature-name> ::= <identifier>

<feature-defs> ::= <feature-name> : <feature-type> .
  | <feature-name> : <feature-type> : <extended-defs> .
  | <feature-name> : <feature-type> := <expression> : <extended-defs> .

<feature-type> ::= real
  | integer
  | boolean
  | nominal
  | nominal (<list>)
  | enumerated (<list>)
  | date
  | time
  | string

<extended-defs> ::= <extended-def> | <extended-def> :
<extended-defs>

<extended-def> ::= <identifier> | <identifier> (<list>)

<list> ::= <identifier> | <identifier> , <list>
```

Para implementar *scripts* usando a sintaxe definida para o ambiente *Discover*, foi desenvolvida uma biblioteca orientada a objetos, em PERL, que implementa os filtros sintáticos necessários, além de uma série de outros métodos para a manipulação de arquivos. A organização dessa biblioteca, bem como sua principais funções são discutidas na Seção 5. Detalhes dessa biblioteca, bem como suas principais funções encontram-se em [Batista and Monard, 2002]

5 Biblioteca do Sistema *Discover*

Como já mencionado, o ambiente descrito neste trabalho será integrado a um ambiente computacional maior, chamado *Discover*, que vem sendo desenvolvido no Laboratório de Inteligência Computacional - LABIC da USP de São Carlos por professores, alunos de pós-graduação e iniciação científica. Esse sistema tem como principal objetivo integrar os diversos trabalhos em Descoberta de Conhecimento em Bancos de Dados desenvolvidos e em desenvolvimento no LABIC.

Por ser este um trabalho de equipe, para permitir que um programa feito por um integrante da equipe possa ser utilizado por outro, foi decidido implementar o sistema em forma de *scripts*, ou seja, pequenos programas que realizam uma única tarefa. Quando se deseja realizar tarefas mais complexas é necessário unir os *scripts* através de um *super-script*. Um *super-script* faz chamadas a mais de um *script*, de forma a quebrar um problema complexo em vários problemas mais simples.

A maior parte dos *scripts* do sistema *Discover* trabalha manipulando dados (entrada para um sistema de aprendizado de máquina) ou regras (saída de um sistema de aprendizado de máquina simbólico). Uma grande parte dos *scripts* lêem dados de um arquivo, fazem alguma operação sobre esses dados e voltam a gravá-los em um arquivo. Ainda, outra grande parte dos *scripts* trabalha manipulando regras. Para facilitar a integração dos trabalhos foi decidido criar duas sintaxes padrão para o sistema, uma para regras [Prati et al., 2001a] e outra para os dados [Batista, 2001].

A sintaxe padrão para regras permite que as regras induzidas por vários indutores simbólicos sejam armazenadas utilizando uma única sintaxe, facilitando assim a integração do conhecimento induzido por algoritmos diferentes de AM Simbólico. Juntamente com a definição da sintaxe padrão, foi criado um conjunto de *scripts* que faz a conversão das regras geradas por diversos indutores para a sintaxe padrão [Prati et al., 2001b].

Da mesma forma, foi criada uma biblioteca orientada a objetos para dar suporte aos *scripts* que utilizam a sintaxe padrão para conjuntos de exemplos [Batista and Monard, 2002]. É importante notar que a sintaxe padrão para conjuntos de exemplos deve ser completa o suficiente para não limitar o uso dos diversos indutores associados ao projeto *Discover*. Por exemplo, alguns poucos sistemas de aprendizado possuem suporte a um tipo nominal ordenado. Para que essa característica não seja perdida, é importante que a sintaxe padrão dê suporte a esse tipo de dado. Também, a sintaxe padrão para conjuntos de exemplos deve suportar extensões e atributos virtuais, os quais são fundamentais para realizar indução construtiva [Lee, 2000]. Todas essas características tornam a sintaxe bastante complexa e justificam a criação de uma biblioteca que fornecerá operações básicas de manipulação da sintaxe padrão aos *scripts* do sistema *Discover*.

A biblioteca oferece um conjunto de operações básicas sobre a sintaxe padrão, entre elas encontram-se:

1. Um conjunto de métodos capazes de carregar na memória o arquivo de definição de atributos e o arquivo com os exemplos. Esses métodos são responsáveis por verificar possíveis erros sintáticos e de incompatibilidade de tipos de dados. Foram definidos métodos que permitem ao usuário da biblioteca acessar e modificar as informações referentes aos dados carregados em memória;

2. Métodos para armazenar em arquivos os dados carregados em memória. Existem diversas operações para armazenar os dados, entre elas a possibilidade de armazenar em diferentes sintaxes, como as sintaxes do CN2 e do C4.5, por exemplo;

3. Métodos para extrair estatísticas descritivas básicas dos atributos, como média, desvio padrão, mínimo, máximo, entre outras;

4. Métodos para realizar pré-processamentos básicos nos dados, como por exemplo normalizações;

5. Método para exportar um conjunto de exemplos para uma tabela em um banco de dados MySQL;

6. Métodos para particionar os dados em conjuntos de treinamento e teste com a finalidade de estimar o erro dos classificadores induzidos. Veja Seção 6

6 *Script* para a Carga do *Dataset* no MySQL (*MySQLLoad*)

Essa função, implementada como parte da biblioteca discutida na Seção 5, tem o objetivo básico de carregar um arquivo de dados em uma tabela do MySQL usando DBI, descrito na Seção 3. O algoritmo por nós implementado, denominado *MySQLLoad* é descrito na Figura 2 na página oposta.

Esse algoritmo utiliza outros métodos presentes na biblioteca. Contudo, não serão discutidos a fundo detalhes de implementação dessas funções e sim a idéia principal do algoritmo. São passados como parâmetros a esse método o nome da base de dados e nome da tabela que se deseja criar. A partir daí começa a ser construída a linha do comando, que fica armazenada na variável `comando_sql`, para criar uma tabela no MySQL. Como pode ser observado, essa construção é gradativa, feita através de sucessivas concatenações de *strings*. Os nomes dos atributos e seus tipos são obtidos através de métodos da biblioteca que recuperam esses valores do arquivo de atributos, fornecidos pelo usuário. Vale destacar que esse arquivo foi previamente convertido e se encontra na sintaxe padrão descrita na Seção 4. Para cada nome de atributo recuperado, entra-se em um *loop* para, dado o seu tipo, encontrar o tipo correspondente definido no MySQL. Essa correspondência se dá da seguinte forma:

1. `nominal` é convertido para `VARCHAR(255)` no MySQL;

2. `enumerated` é convertido para `ENUM` no MySQL;

3. `integer` é convertido para `INTEGER` no MySQL;

4. `real` é convertido para `REAL` no MySQL;

```
início
        ler(nome_base);
        ler(nome_tabela);
        conectar(nome_base);

        comando_sql := ¨CREATE TABLE tabela (¨;
        para (i := 0, i < Número_de_Atributos, i++)
         início
                comando_sql := concatena(comando_sql, Nome_Atributo(i));
                tipo          := Tipo_Atributo(i);
                se (tipo == ¨nominal¨)
                        comando_sql := concatena(comando_sql, ¨VARCHAR(255);¨)
                senao se (tipo == ¨enumerated¨)
                 início
                        concatena(comando_sql, ¨ ENUM (¨);
                        lista := Tipo_Lista(i);
                        para (j := 0, j < num_elementos(lista), i++)
                                concatena(comando_sql, ¨ ¨lista[j]¨ ¨);
                        concatena(comando_sql, ¨ lista[num˙elementos(lista)]¨);
                 fim
                senão se (tipo == ¨integer¨)
                        comando_sql := concatena(comando_sql, " INTEGER");
                senão se (tipo == ¨real¨)
                        comando_sql := concatena(comando_sql, " REAL");
                senão se (tipo == "string")
                        comando_sql := concatena(comando_sql, " TEXT");
                senão se (tipo == "date")
                        comando_sql := concatena(comando_sql, " DATE");
                senão se (tipo == "time")
                        comando_sql := concatena(comando_sql, " TIME");
                se (i < Número_de_Atributos)
                        comando_sql := concatena(comando_sql, ", ");
         fim

        executar_comando_SQL(comando_sql);
        preparar_para_tabela(arquivo);
        carregar_tabela(arquivo);
        desconectar(nome_base)
fim.
```

Figura 2: Algoritmo `MySQLLoad`

5. `string` é convertido para `STRING` no MySQL;
6. `date` é convertido para `DATE` no MySQL;
7. `time` é convertido para `TIME` no MySQL;

Por exemplo, todo atributo declarado como *string* no arquivo de atributos na sintaxe padrão será declarado como uma coluna de tipo *TEXT* no MySQL, um atributo *nominal* no arquivo de atributos como uma coluna do tipo *ENUM* (seguido de uma lista de valores) no MySQL, e assim sucessivamente. Construída essa linha de comando, ela é executada e uma tabela é criada. Agora é preciso carregar o arquivo de dados para a tabela. Para isso é usado o método `preparar_para_tabela`. Esse método gera um arquivo intermediário que separa os atributos contidos em uma mesma linha por caracteres TAB. Isso é feito porque é esse o formato que o MySQL aceita para carregar arquivos texto diretamente para suas tabelas. Após isso, a tabela é carregada com o arquivo e finalmente desconecta-se da base.

7 *Scripts* Implementados para Recuperação das Taxas de Erro

Estimar a taxa de erro de algoritmos de Aprendizado de Máquina é uma tarefa altamente repetitiva. Usualmente o método 10-fold cross validation é utilizado. Isso significa que o algoritmo de aprendizado é executado 10 vezes, utilizando as diversas partições do conjunto de dados. Mas, como normalmente os experimentos realizados por pesquisadores da área envolvem medir o erro em diversos conjuntos de exemplos e com diferentes algoritmos de aprendizado, é bastante provável que os algoritmos de aprendizado sejam executados centenas de vezes. Essa é a principal motivação para criar um ambiente que estime automaticamente o erro. Para cada execução, os sistemas de aprendizado geram um arquivo de *log* no qual podem ser encontradas diversas informações, como por exemplo o conhecimento extraído e os erros cometidos pelo classificador no conjunto de teste. Neste trabalho, o que se deseja é identificar automaticamente onde se encontram as informações sobre os erros de classificação, normalmente disponibilizadas através de uma matriz de confusão, e recuperar essas informações. Posteriormente, os erros recuperados de cada iteração deverão ser combinados para que seja calculada a estimativa da taxa de erro do sistema de aprendizado. Esta seção descreve alguns *scripts* implementados para recuperar automaticamente o erro no conjunto de teste dos arquivos de *log* gerados pelos sistemas de aprendizado C4.5 [Quinlan, 1988], C5.0 [Rulequest-Research, 1999] e CN2 [Clark and Niblett, 1989].

7.1 Filtro para o C4.5 (*fc45*)

Este programa, executado na forma de linha de comando, tem a seguinte forma:

`fc45` [*nome do arquivo*]

O arquivo fornecido como parâmetro ao programa é o arquivo obtido pelo redirecionamento da saída do algoritmo C4.5 [Quinlan, 1988] para um arquivo. Um trecho desse arquivo, com as informações relevantes para o entendimento deste programa, pode ser observado na Figura 3 na próxima página.

Uma vez fornecido o arquivo, o programa procura pelo padrão `''classified as''`. Se esse padrão não é encontrado no arquivo, a variável `resultado` assumirá um valor diferente de zero. Com isso, uma mensagem de erro será impressa informando que não foi usado um *conjunto de teste* para o cálculo da taxa de erro (mais informações em [Kemp et al., 2001]) e o programa será

```
C4.5 [release 8] decision tree generator        Thu Feb  8 18:20:20 2001
-------------------------------------------

                              ...
                              ...

Tree saved


Evaluation on training data (490 items):

         Before Pruning           After Pruning
        ----------------   ----------------------------
        Size      Errors   Size      Errors   Estimate

          88   19( 3.9%)     57   24( 4.9%)    (11.9%)   <<

Evaluation on test data (200 items):

         Before Pruning           After Pruning
        ----------------   ----------------------------
        Size      Errors   Size      Errors   Estimate

          88   35(17.5%)     57   35(17.5%)    (11.9%)   <<


          (a)       (b) <-classified as
         ----  ----
            74     16   (a): class +
            19     91   (b): class -
```

Figura 3: Exemplo de Saída do Algoritmo C4.5

abortado. Encontrado esse padrão, o programa lê mais duas linhas do arquivo para se posicionar no começo da matriz de confusão. Como essa matriz é a última saída gerada pelo C4.5, a leitura de dados se dará até o final do arquivo, sendo isso controlado por um *loop*. Nesse *loop*, uma linha da matriz de confusão é particionada através da função `particiona`. Essa função elimina o caracter TAB presente no começo de todas as linha e depois retorna um vetor no qual cada caracter da linha da matriz de confusão é um elemento desse vetor. A essa altura entra-se em um segundo *loop* que tem como condição de parada o primeiro caracter TAB encontrado no vetor. Foi escolhido esse critério de parada porque é justamente um caracter TAB que separa o fim de uma linha da matriz de confusão de outras informações inseridas pelo C4.5.

A única restrição, nesse estágio, acontece na quantidade de dígitos dos valores da matriz de confusão. Como será discutido mais adiante, o tamanho máximo foi fixado em 4 dígitos (ou seja, no máximo 9999). O processo de obtenção dos valores é simples. Dado o vetor que a função `particiona` retorna, o que se faz é pegar as quatro primeiras posições desse vetor, concatenar, armazenar esse valor numa posição específica de uma matriz, ignorar a próxima, pegar as quatro

```
início
        ler(nome_arq);
        arquivo    := abrir(nome_arq);

        linha      := próxima_linha(arquivo);
        resultado := (linha <> ¨classified as¨);

        enquanto (não_vazio(linha) e (resultado <> 0))
         início
                linha      := próxima_linha(arquivo);
                resultado := (linha == ¨classified as¨);
         fim

        se (resultado <> 0) então
            sair_do_programa(¨Não foi usado um arquivo de teste!¨);

        linha := próxima_linha(arquivo);
        linha := próxima_linha(arquivo);

        i := 0;
        j := 0;

        enquanto (não_vazio(linha))
         início
                k := 0;
                quebra := particiona(linha);

                enquanto (quebra[k] <> "\t")
                 início
                        concatena := concatenação(quebra[k+1],quebra[k+2],quebra[k+3],quebra[k+4]);
                        se concatena == "     ";
                         início
                                vetor[i][j] := 0;
                         fim
                        senão
                         início
                                vetor[i][j] := concatena;
                         fim
                        j := j + 1;
                        k := k + 5;
                 fim
                i := i + 1;
                j := 0;
                linha := próxima_linha(arquivo);
         fim
fim.
```

Figura 4: Algoritmo `fc45`

na seqüência, concatenar, e assim sucessivamente até que se encontre um caracter TAB. Se o valor obtido da concatenação for *"vazio"*, então é colocado o valor 0 (zero) na posição correspondente da matriz resultante. Isso é feito dessa forma porque foi identificado esse padrão na organização do arquivo.

7.2 Filtro para o C4.5rules (*fc45r*)

Este programa, executado na forma de linha de comando, tem a seguinte forma:

`fc45r` [*nome do arquivo*]

O arquivo fornecido como parâmetro ao programa é o arquivo obtido pelo redirecionamento da saída do algoritmo C4.5rules [Quinlan, 1988] para um arquivo. Um trecho desse arquivo, com as informações que são de relevância para o entendimento deste programa, pode ser observado na Figura 5 na página 15.

Como pode ser observado na Figura 6 na página 16, o algoritmo que implementa esse filtro é idêntico ao algoritmo do filtro usado para o C4.5 (Seção 7.1), com a única diferença de que no começo do programa procura-se pela segunda ocorrência do padrão `''classified as''`. Isso se dá nesse filtro porque o C4.5rules, quando calcula a taxa de erro sobre o conjunto de teste, calcula também a taxa de erro aparente. E essa taxa é mostrada em uma tabela com um formato idêntico ao da tabela que desejamos. Assim, o que é feito é simplesmente ignorar a sua existência.

7.3 Filtro para o CN2 (*fcn2*)

Este programa, executado na forma de linha de comando, tem a seguinte forma:

`fcn2` [*nome do arquivo*]

O CN2 [Clark and Niblett, 1987] é um sistema que permite interação com o usuário, diferente do C4.5 e do C4.5rules que são programa de linha de comando, o que torna a forma de obtenção deste arquivo um pouco diferente. Primeiro cria-se um arquivo com toda a seqüência de interações que deseja-se fazer no CN2. A seguir executa-se o CN2 tomando o cuidado de redirecionar a entrada do programa (mandar que ele receba os comando do arquivo e não do teclado) e a saída (para um outro arquivo). É esse outro arquivo que é passado para o filtro.

Assim como os dois filtros discutidos anteriormente (Seções 7.1 e 7.2), tudo começa com a procura de um padrão dentro do arquivo fornecido. Nesse caso, o padrão procurado é o padrão `''PREDICTED''`. Vale destacar que esse padrão só ocorre no início da matriz de confusão que apresenta a taxa de erro verdadeira. Se esse padrão não é encontrado a variável `resultado` assume um valor diferente de zero, uma mensagem de erro é impressa e o programa é abortado.

Como pode ser observado na Figura 7 na página 17, a matriz de confusão aparece no meio do arquivo, sendo o seu fim marcado pela ocorrência do padrão `''Overall''`. É por isso que o loop para a obtenção dos dados tem como condição de parada uma variável que armazena a ocorrência

ou não desse padrão. O processo de obtenção dos valores é simples. A função `particiona` definida nesse filtro retorna um vetor em que cada posição desse vetor contém um elemento da matriz (todos os dados irrelevantes que aparecem no começo da linha são descartados). Assim, o processo de extração de dados do vetor se resume a copiar os dados do vetor para sua devida posição na matriz resultante até que seja encontrado o padrão `‘‘.’’`, o qual indica que chegou-se num valor de porcentagem, que não é de interesse. Então o processo é repetido até que o padrão `‘‘Overall’’` seja encontrado. Esse algoritmo encontra-se descrito na Figura 8 na página 18.

Vale destacar que o processo de particionamento deste filtro é diferente dos dois anteriores porque o formato da saída para as matrizes é diferente. A diferença fundamental é o fato de que o CN2, quando uma entrada da matriz assume um valor 0 (zero), informa que o valor é zero (imprime o valor zero, como pode ser observado na Figura 7 na página 17). Já o C4.5 e o C4.5rules não. Eles simplesmente deixam aquela entrada na matriz vazia. Desta forma, para este dois últimos algoritmos foi preciso encontrar uma forma de estabelecer quais são os limites para a presença de dados e quais são os limites para a separação desses dados. A forma como isso foi realizado é discutida na Seção 7.1.

```
C4.5 [release 8] rule generator Thu Feb  8 18:22:08 2001
-------------------------------------

                                  ...
                                  ...

          (a)   (b)     <-classified as
          ----  ----
           191    26    (a): class +
             8   265    (b): class -


Evaluation on test data (200 items):

Rule  Size  Error  Used  Wrong          Advantage
----  ----  -----  ----  -----          ---------
  28     2   3.6%    51    5 (9.8%)      3 (4|1)   +
  15     3   4.6%    12    6 (50.0%)    -1 (3|4)   +
  24     2   5.1%    11    1 (9.1%)      3 (4|1)   +
   1     3   5.1%    12    3 (25.0%)     6 (7|1)   +
  14     3   5.7%     3    0 (0.0%)      3 (3|0)   +
   6     3  12.9%     1    0 (0.0%)      1 (1|0)   +
   8     6  14.3%     1    1 (100.0%)   -1 (0|1)   +
  21     5   6.9%    16    2 (12.5%)     0 (0|0)   -
  35     1   8.1%    78    7 (9.0%)      0 (0|0)   -
  17     4  17.8%     1    0 (0.0%)      0 (0|0)   -

Tested 200, errors 31 (15.5%)    <<


          (a)   (b)     <-classified as
          ----  ----
            75    15    (a): class +
            16    94    (b): class -
```

Figura 5: Exemplo de Saída do Algoritmo C4.5rules

```
início
        ler(nome_arq);

        arquivo    := abrir(nome_arq);
        cont       := 0;
        linha      := próxima_linha(arquivo);
        resultado := (linha <> ¨classified as¨);

        enquanto (não_vazio(linha) e (cont < 2))
         início
                linha      := próxima_linha(arquivo);
                resultado := (linha <> ¨classified as¨);
                se (resultado == 0) então
                 início
                        cont := cont + 1;
                 fim
         fim

        se (resultado <> 0) então
            sair_do_programa(¨Não foi usado um arquivo de teste!¨);

        linha := próxima_linha(arquivo);
        linha := próxima_linha(arquivo);

        i := 0;
        j := 0;

        enquanto (não_vazio(linha))
         início
                k := 0;
                quebra := particiona(linha);

                enquanto (quebra[k] <> "\t")
                 início
                        concatena := concatenação(quebra[k+1],quebra[k+2],quebra[k+3],quebra[k+4]);
                        se concatena == "     ";
                         início
                                vetor[i][j] := 0;
                         fim
                        senão
                         início
                                vetor[i][j] := concatena;
                         fim
                        j := j + 1;
                        k := k + 5;
                 fim
                i := i + 1;
                j := 0;
                linha := próxima_linha(arquivo);
         fim
fim.
```

Figura 6: Algoritmo `fc45r`

```
                    Welcome to CN2!

                        ...
                        ...

IF    sepal−length < 6.95
THEN  class = Iris−versicolor  [0 2 0]
ELSE
(DEFAULT) class = Iris−viginica  [0 0 1]
CN> Execute
EVAL> all
Executing rules...
        PREDICTED
ACTUAL      Iris−se Iris−ve Iris−vi Accuracy
_Iris−set        50       0       0    100.0 %
_Iris−ver         0      50       0    100.0 %
_Iris−vig         0       0      50    100.0 %
_Overall accuracy: 100.0 %
EVAL>
CN> Write
WRITE> Filename? saida.iris
Writing current rules to saida.iris
CN> Quit
Have a nice day!
```

Figura 7: Exemplo de Saída do Algoritmo CN2

```
início
        ler(nome_arq);
        arquivo    := abrir(nome_arq);

        linha      := próxima_linha(arquivo);
        resultado := (linha <> ¨PREDICTED¨);

        enquanto (não_vazio(linha) e (resultado <> 0))
         início
                linha      := próxima_linha(arquivo);
                resultado := (linha <> ¨PREDICTED¨);
         fim

        se (resultado <> 0) então
            sair_do_programa(¨Não foi usado um arquivo de teste!¨);

        linha := próxima_linha(arquivo);
        linha := próxima_linha(arquivo);

        i := 0;
        j := 0;

        resultado := (linha <> ¨Overall¨);

        enquanto (resultado <> 0)
         início
                k := 1;
                quebra := particiona(linha);
                resultado2 := (quebra[k] <> ".");
                enquanto (resultado2 <> 0)
                 início
                        vetor[i][j] := quebra[k];
                        j := j + 1;
                        k := k + 1;
                        resultado2 := (quebra[k] <> ".");
                 fim
                i := i + 1;
                j := 0;
                linha := próxima_linha(arquivo);
         fim
fim.
```

Figura 8: Algoritmo `fcn2`

8 Conclusão

Este trabalho teve por objetivo apresentar a implementação de alguns *scripts* que compõem o sistema *Discover*. Estes *scripts* incluem a implementação do método que faz a carga dos arquivos de dados para o MySQL e a implementação de filtros que recuperam a taxa de erro gerada por cada um dos algoritmos de Aprendizado de Máquina simbólico.

Além de discutir os *scripts* de filtro e de carga de arquivos para a base de dados, este trabalho apresenta considerações de grande importância para o sistema *Discover*, tais como a sintaxe padrão de arquivos adotada para o sistema *Discover* e a biblioteca PERL desenvolvida para o *Discover*. Ainda, vale destacar a apresentação de algumas das tecnologias utilizadas pelo projeto, tais como o sistema de base de dados MySQL e a API de comunicação desse sistema de banco de dados com a linguagem PERL chamada DBI.

Referências

[DBI, 2001] (2001). DBI CPAN Documentation. http://theoryx5.uwinnipeg.ca/CPAN/data/DBI/DBI.html.

[Axmark et al., 2000] Axmark, D., Widenius, M., and DuBois, P. (2000). *MySQL 3.23 Reference Manual*. http://www.mysql.com/doc.html.

[Baranauskas and Batista, 2000] Baranauskas, J. A. and Batista, G. E. A. P. A. (2000). O Projeto *Discover*: Idéias Iniciais (Comunicação Pessoal).

[Batista, 2001] Batista, G. E. A. P. A. (2001). Definição da Sintaxe Padrão do Sistema Discover. http://www.icmc.sc.usp.br/∼gbatista/Discover/SintaxePadraoFinal.html.

[Batista and Monard, 2002] Batista, G. E. A. P. A. and Monard, M. C. (2002). A Biblioteca da Sintaxe Padrão para Arquivos de Exemplos do Sistema Discover. Technical report. Trabalho em Andamento.

[Clark and Niblett, 1987] Clark, P. and Niblett, T. (1987). Induction in Noise Domains. In Bratko, I. and Lavrač, N., editors, *Proceedings of 2nd European Working Session on Learning*, pages 11–30, Wilmslow, UK. Sigma.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283.

[Finnin, 2001] Finnin, T. (2001). A brief mSQL Tutorial. http://www.csee.umbc.edu/∼finin/461/msql/.

[Kemp et al., 2001] Kemp, A. H., Batista, G. E. A. P. A., and Monard, M. C. (2001). Descrição da Implementação dos Métodos Estatísticos de Resampling do Ambiente *Discovery*. Technical Report 143, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_143.ps.zip.

[Lee, 2000] Lee, H. D. (2000). Seleção e construção de features relavantes para o aprendizado de máquina. Dissertação de Mestrado, ICMC-USP.

[MySQL, 1995] MySQL (1995). Mysql web site. http://www.mysql.com.

[Prati et al., 2001a] Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2001a). Extração de informações padronizadas para a avaliação de regras produzidas por algoritmos de aprendizado de máquina simbólicos. Technical Report 145, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip.

[Prati et al., 2001b] Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2001b). Uma proposta de unificação da linguagem de representação de conceitos de algoritmos de aprendizado de máquina simbólicos. Technical Report 137, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip.

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

[Rulequest-Research, 1999] Rulequest-Research (1999). Data Mining Tools See5 and C5.0. http://www.rulequest.com/see5-info.html em 03/1999.

[Till, 1996] Till, D. (1996). *Teach Yourself Perl 5 in 21 Days*. Sams Publishing.