Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Projeto e Implementação de um Filtro para Transformar Logs de Servidores Web em Arquivos no Formato Padrão do Sistema Discover

Ramon Chiara
Maria Carolina Monard

Nº 183

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Janeiro de 2003

Projeto e Implementação de um Filtro para Transformar Logs de Servidores Web em Arquivos no Formato Padrão do Sistema Discover*

Ramon Chiara
Maria Carolina Monard

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística
Laboratório de Inteligência Computacional
Caixa Postal 668, 13560-970 - São Carlos, SP, Brasil
e-mail: {ramon, mcmonard}@icmc.usp.br

Resumo

Existe um crescente interesse em se aplicar técnicas de Mineração de Dados em arquivos de logs de servidores Web. Estes arquivos armazenam os registros dos acessos feitos ao servidor, sendo uma potencial fonte de dados para ser minerada.

Um dos projetos em andamento no LABIC — Laboratório de Inteligência Computacional — é o desenvolvimento do Sistema Discover. Esse sistema tem como objetivo auxiliar no processo de KDD (*Knowledge Discovery in Databases*). Uma de suas propostas é a padronização da sintaxe dos arquivos de entrada utilizados por algoritmos de Aprendizado de Máquina.

Este relatório descreve os detalhes de um filtro que foi desenvolvido para transformar logs de servidores Web em arquivos de dados na sintaxe padrão do Sistema Discover. São explicados em detalhes tanto a arquitetura quanto o formato dos arquivos gerados a fim de facilitar uma futura manutenção.

Palavras-Chave: Data Mining, Web Mining.

Janeiro de 2003

*Projeto realizado com o auxílio da CAPES e FINEP

Sumário

1 Introdução **1**

2 O Projeto Discover **2**
2.1 O Sistema Discover . . . 3
2.2 A Sintaxe Padrão para Arquivos de Dados . . . 4
2.2.1 Arquivos . . . 4
2.2.2 Tipos de Dados . . . 5
2.2.3 A Gramática da Sintaxe Padrão . . . 6

3 O Funcionamento da Web **7**
3.1 O arquivo de log . . . 9
3.2 Interações Problemáticas . . . 11

4 O Filtro **12**
4.1 Definição do arquivo `.names` . . . 13
4.2 Implementação . . . 16
4.2.1 Obtenção dos parâmetros de entrada . . . 17
4.2.2 Criação do arquivo `.names` . . . 17
4.2.3 Criação do arquivo `.data` . . . 17
4.3 Exemplos de Entradas e Saídas . . . 19
4.4 Alterando o Filtro . . . 20

5 Conclusão **22**

Lista de Figuras

1 Exemplo de utilização de filtros no Sistema Discover . . . 1
2 Interação entre filtros, sintaxes e bibliotecas . . . 3
3 Gramática da sintaxe do arquivo `.names`. . . . 6
4 Transferência de um arquivo HTML sem figuras através do protocolo HTTP 8
5 Transferência de um arquivo HTML com figuras através do protocolo HTTP 8
6 O protocolo HTTP também carrega outras informações, como o `referer` . . 11
7 Funcionamento de um servidor *proxy* . . . 12
8 Entrada e saídas do filtro `log2discovery.pl`. . . . 16
9 Conteúdo do arquivo `.names` gerado pelo filtro. . . . 18

1 Introdução

Existe um crescente interesse em se aplicar técnicas de Mineração de Dados em arquivos de logs de servidores Web [Chiara, 2001, Kosala and Blockeel, 2000, Kimball and Merz, 2000, Spiliopoulou et al., 1999, Zaïane et al., 1998]. Estes arquivos armazenam os registros dos acessos feitos ao servidor, sendo uma potencial fonte de dados para ser minerada.

No entanto, ao se aplicar as técnicas de Mineração de Dados utilizando-se algoritmos de Aprendizado de Máquina enfrenta-se um problema: a maioria dos algoritmos de Aprendizado de Máquina tem um formato diferente para os arquivos de entrada. De forma a contornar esta dificuldade, foi desenvolvida uma sintaxe padrão para a representação desses arquivos de entrada [Batista, 2001], os quais consistem num arquivo de dados (exemplos) propriamente dito, e num arquivo que descreve os atributos desses dados.

Esses arquivos, na sintaxe padrão, são utilizados pelo Sistema Discover, que está sendo implementado no LABIC[1] — Laboratório de Inteligência Computacional. Esse sistema tem como objetivo auxiliar no processo de KDD (*Knowledge Discovery in Databases*). Ele consiste em um conjunto de *scripts* PERL [Wall and Schwartz, 1991] e em uma biblioteca de rotinas que são utilizadas pelos *scripts*. Esses *scripts* estão sendo integrados através de uma interface gráfica [Geromini, 2002]. Alguns desses *scripts* são filtros que transformam arquivos de dados na sintaxe padrão para a sintaxe dos arquivos de entrada dos diversos algoritmos de Aprendizado de Máquina suportados pelo Sistema Discover— Figura 1.

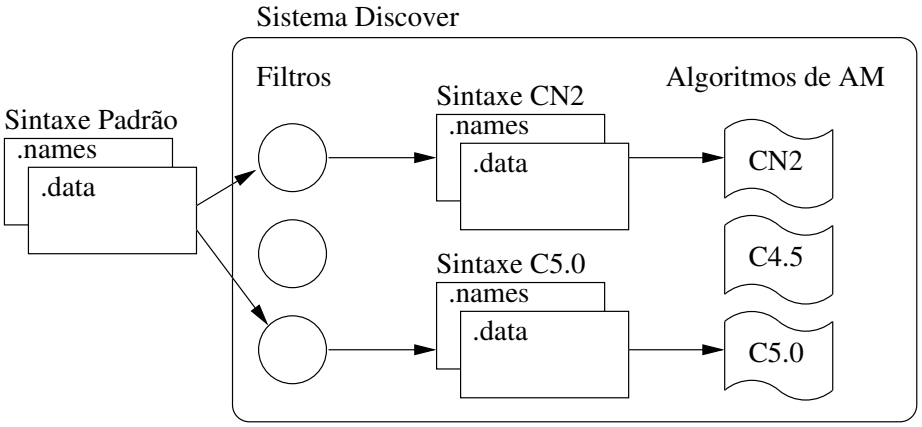

Figura 1: Exemplo de utilização de filtros no Sistema Discover

Dessa forma, para que os arquivos de logs de servidores Web possam ser utilizados no Sistema Discover, um filtro que transforma esses logs em arquivos de dados na sintaxe padrão foi por nós desenvolvido. O objetivo deste relatório é descrever os detalhes envolvidos no desenvolvimento desse filtro de forma a fornecer uma base para uma futura manutenção.

O trabalho está estruturado da seguinte forma: na Seção 2 é feita uma breve descrição

[1] http://labic.icmc.sc.usp.br

do Sistema DISCOVER bem como da sintaxe padrão utilizada pelos arquivos de entrada do sistema. Na Seção 3 é descrito o funcionamento da Web. Isso é importante para que haja um bom entendimento do conteúdo dos arquivos de log. Na Seção 4 são descritos os detalhes técnicos do filtro e na Seção 5 é apresentada a conclusão sobre o trabalho.

2 O Projeto Discover

A descrição aqui apresentada está baseada nos trabalhos realizados por integrantes da equipe de desenvolvimento do Sistema DISCOVER. Assim, maiores informações podem ser obtidas em [Batista, 2001, Kemp et al., 2002, Prati, 2002].

O processo de KDD requer experimentação, análise e comparação de diversos modelos de extração de conhecimento, na busca por um resultado satisfatório. Na condução desses experimentos, atividades como transformações de formato, adaptações, execução de diferentes algoritmos de Aprendizado de Máquina, medições, entre outras, devem ser executadas um grande número de vezes.

Muitas dessas tarefas podem ser automatizadas utilizando-se sistemas integrados comerciais. Geralmente, essas ferramentas tem um caráter mais exploratório e fazem uso de algoritmos e ferramentas proprietários, o que dificulta o seu uso por pesquisadores que pretendem analisar e desenvolver novos algoritmos e ferramentas. Uma alternativa é o uso de ferramentas de domínio público. Entretanto, essas ferramentas também têm algumas características não apropriadas para realizar pesquisas mais abrangentes.

Esses fatores conduzem, diversas vezes, à necessidade de se utilizar os algoritmos de aprendizado tal como foram implementados pelos seus idealizadores e, por conseguinte, todas as atividades necessárias para a execução de experimentos devem ser feitas para cada algoritmo particular. Essa necessidade implica, muitas vezes, no desenvolvimento de programas para automatizar essas tarefas.

Nos últimos anos, diversos pesquisadores de nosso laboratório de pesquisa (LABIC) têm utilizado algoritmos de Aprendizado de Máquina em suas pesquisas, desenvolvendo, na forma de *scripts* PERL, uma série de ferramentas para facilitar a configuração e execução de experimentos [Prati et al., 2001b, Prati et al., 2001a, Batista, 2001, Kemp et al., 2001].

Surgiu, então, a proposta de desenvolver um projeto conjunto no qual todos os membros do laboratório que trabalham na área de KDD estariam envolvidos: o projeto DISCOVER [Baranauskas and Batista, 2000]. A princípio, o projeto DISCOVER consistiria apenas de um repositório de *scripts* . Por meio da combinação desses *scripts* independentes seria possível a realização de tarefas mais complexas. Posteriormente surgiu a proposta de se criar um ambiente integrado, no qual os *scripts* seriam substituídos por bibliotecas de classes e essas classes empacotadas como componentes, com a composição dos componentes sendo feita através de uma interface gráfica [Geromini, 2002].

2.1 O Sistema Discover

A idéia central do Sistema Discover consiste em utilizar os algoritmos de aprendizado implementados pela comunidade e as ferramentas com finalidades específicas desenvolvidas pelos pesquisadores relacionados ao projeto, tais como ferramentas de pré-processamento de dados e pré-processamento de texto, tanto para aprendizado supervisionado quanto para aprendizado não-supervisionado, amostragem e avaliação de erro, mesclagem de regras, cobertura de regras, qualidade de regras, entre outros.

De uma maneira geral, o Sistema Discover pode ser entendido como um conjunto de métodos que são aplicados sobre os dados ou sobre o conhecimento extraído a partir dos dados. Dessa forma, é muito importante que o sistema ofereça um base sólida para manipular dados e conhecimento. Essa base é composta por sintaxes padrões para a representação de dados e de conhecimento, e por bibliotecas que oferecem um conjunto de funcionalidades básicas de manipulação de dados e de conhecimento. Atualmente, existem definidas sintaxes padrões para a representação de dados no formato atributo-valor e para a representação do conhecimento induzido por diversos indutores simbólicos, bem como bibliotecas que oferecem funcionalidades sobre essas sintaxes padrões. Futuramente, novas sintaxes padrão devem ser especificadas, principalmente para a representação de regras de regressão [Dosualdo, 2002], regras de associação [Melanda, 2002] e *clusters* [Martins, 2001]. A Figura 2 mostra, de uma forma simplificada, como os filtros, sintaxes e bibliotecas interagem uns com os outros, para o caso de algoritmos simbólicos de aprendizado supervisionado.

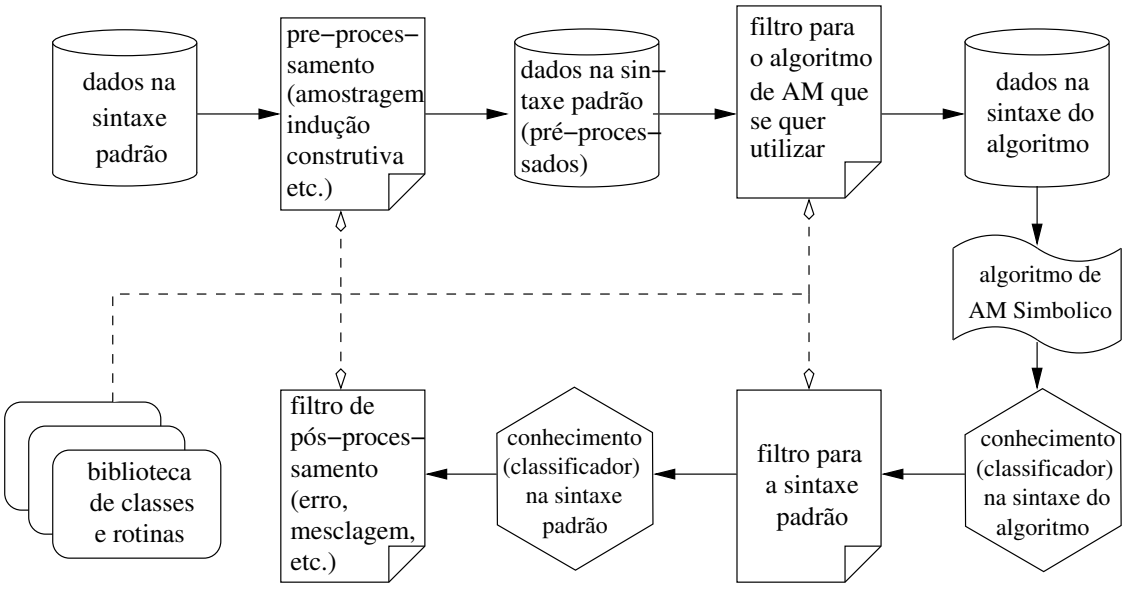

Figura 2: Interação entre filtros, sintaxes e bibliotecas

A vantagem do Sistema Discover como ferramenta de apoio à pesquisa em KDD, em relação a outros sistemas, é a visão unificada que os formatos padrões oferecem para quem está

desenvolvendo novos componentes, além de um conjunto de ferramentas para a manipulação dos mesmos.

Uma outra vantagem que se espera é que, a partir da conclusão dos diversos trabalhos que estão sendo realizados, o Sistema DISCOVER ofereça uma série de vantagens em relação à composição das ferramentas desenvolvidas em nosso laboratório, evitando que essas implementações se percam e, além disso, reunindo-as em um ambiente integrado.

2.2 A Sintaxe Padrão para Arquivos de Dados

Como já mencionado, 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 com que seja necessário definir uma sintaxe padrão para representar os arquivos que serão utilizados 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 (C4.5, por exemplo). Assim, novos algoritmos de Aprendizado de Máquina podem ser facilmente adicionados ao DISCOVER através da implementação desses filtros, como ilustrado na Figura 1, página 1.

2.2.1 Arquivos

A sintaxe padrão, definida em [Batista, 2001], utiliza arquivos do tipo texto para declarar os atributos (e seus respectivos tipos) e os valores que esses atributos assumem em um determinado conjunto de exemplos. Os atributos são declarados em um arquivo com a extensão `.names` e 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.

No caso de Aprendizado de Máquina supervisionado, 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 arquivo `.names` assumem para um conjunto de exemplos. Cada linha de um arquivo `.data` representa um

exemplo. Desta forma, o "separador de registros" é o caractere 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. 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`.

2.2.2 Tipos de Dados

Foram definidos os seguintes tipos de dados[2], que podem ser associados aos identificadores de atributos:

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 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 uma string de tamanho indefinido e 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 da 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`.

[2] Os termos utilizados nas implementações de cada componente do Sistema DISCOVER estão em inglês pois pretende-se compartilhar os resultados do projeto com a comunidade científica.

2.2.3 A Gramática da Sintaxe Padrão

A gramática que define a sintaxe do arquivo `.names` é mostrada na Figura 3.

```
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>
```

Figura 3: Gramática da sintaxe do arquivo `.names`.

Para implementar *scripts* usando a sintaxe definida para o Sistema DISCOVER, foi desenvolvida uma biblioteca orientada a objetos, em Perl [Wall and Schwartz, 1991], que implementa uma série de métodos para a manipulação de arquivos na sintaxe padrão. A descrição da organização dessa biblioteca, bem como sua principais funções, podem ser encontradas em [Batista, 2003].

3 O Funcionamento da Web

Nesta seção é descrito o funcionamento da Web com o objetivo de fornecer um embasamento mínimo para o entendimento do conteúdo dos arquivos de log, tratados neste trabalho.

Quando um browser é utilizado para navegar na Web, muitas coisas estão acontecendo para que uma página requisitado seja transferida do servidor para a máquina do usuário e, finalmente, seja mostrada para ele. Quando digita-se, no browser, um endereço de uma página que contenha apenas texto, por exemplo `http://www.nossosite.com/texto.html`[3], os seguintes passos, ilustrados na Figura 4, são executados:

1. O browser procura pelo servidor `www.nossosite.com` e conecta-se com ele utilizando a porta *default* 80, caso nenhuma outra tenha sido indicada.

2. Uma vez conectado, o browser envia uma requisição para o servidor pedindo a página `texto.html`. A forma pela qual é feita essa requisição segue o protocolo HTTP (HyperText Transfer Protocol)[Group, 1999a].

3. (a) O servidor, de posse do nome do arquivo requisitado, procura-o em seu disco.
   (b) Uma vez achado, ele envia o conteúdo deste arquivo para o browser. Deve-se observar que o protocolo HTTP ainda está em efeito. Ele define, por exemplo, o que o servidor deve enviar para o browser caso o arquivo requisitado não exista em seu disco.

4. Neste momento, o arquivo requisitado já foi transferido e o seu conteúdo encontra-se na memória do computador do usuário. O browser lê este conteúdo interpretando-o, fecha a conexão com o servidor e mostra o arquivo para o usuário. Observa-se que o browser primeiro interpreta o arquivo para depois fechar a conexão. Isso acontece para que ele aproveite a conexão aberta para mandar outras requisições caso isso se faça necessário — caso de arquivos HTML (HyperText Markup Language) que contenham figuras.

Esse cenário traduz o funcionamento básico da Web. No caso de um arquivo HTML que contenha texto e figuras, o funcionamento é basicamente o mesmo, com a diferença que, após o browser ter lido e interpretado o arquivo HTML e detectado figuras, ele faz novas requisições independentes para o servidor, como mostra a Figura 5. Uma vez que as figuras foram transferidas, elas podem ser mostradas nos seus devidos lugares, ou seja, no lugar do texto onde o arquivo HTML indicou que haviam figuras.

Resumindo, o protocolo HTTP é um protocolo do tipo *request/response*, ou seja, o browser faz requisições e o servidor responde a elas. Além disso, o HTTP é, também, um protocolo *stateless*, isto é, ele não guarda nenhum tipo de informação sobre o estado da conexão entre o browser e o servidor, de forma que, depois de uma requisição ser atendida, se o mesmo usuário fizer outra requisição o servidor não se lembrará que trata-se do mesmo usuário,

[3] Endereço fictício

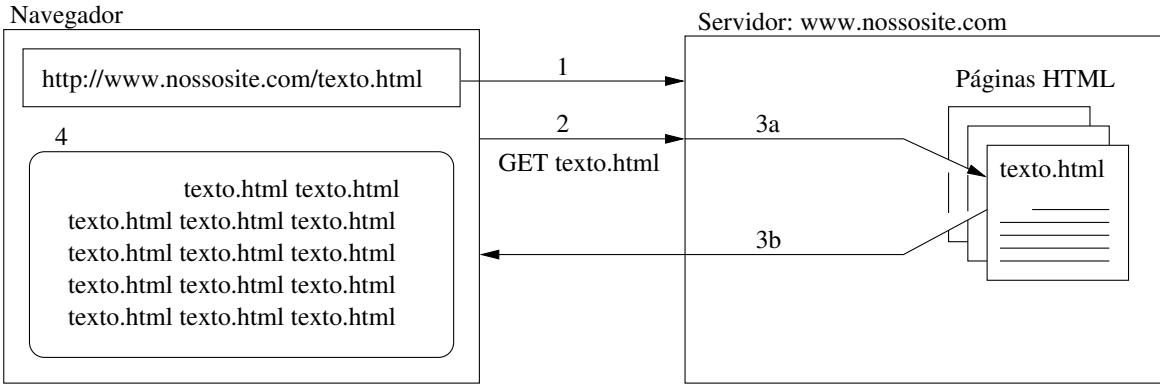

Figura 4: Transferência de um arquivo HTML sem figuras através do protocolo HTTP

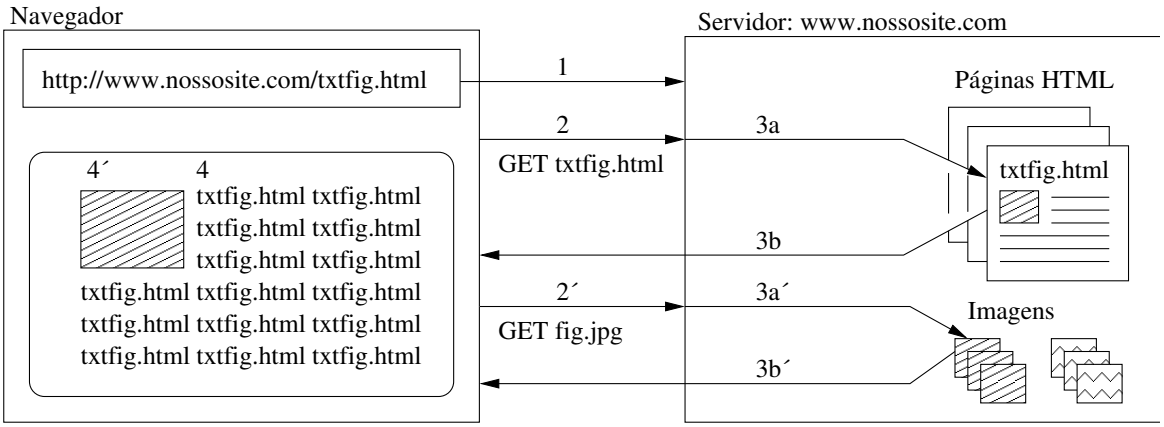

Figura 5: Transferência de um arquivo HTML com figuras através do protocolo HTTP

tratando-o como um novo cliente. Em outras palavras, isso significa que o usuário não se identifica para o servidor (por meio de um login, por exemplo) e recupera vários documentos : ele faz uma conexão separada para cada arquivo que ele quer recuperar, seja um arquivo de texto, HTML, imagens ou qualquer outro tipo (no caso de *downloads*, applets Java, etc.).

Isso não quer dizer que o HTTP seja um protocolo ruim. Pelo contrário, isso faz com que o protocolo seja simples de se entender e de se implementar. O único problema que isso implica está no caso de querermos aplicar técnicas de Mineração de Dados [Weiss and Indurkhya, 1998] no log do servidor, pois informações muito úteis estarão faltando nos dados como, por exemplo, quem é o usuário que está acessando o servidor.

3.1 O arquivo de log

Todos os servidores Web têm a capacidade de registrar, em um arquivo de log, a sua interação com os clientes. Toda vez que um servidor responde a uma solicitação HTTP, ela é registrada no arquivo de log. É importante notar que, apesar de um registro ser feito para cada requisição, o servidor estará atendendo várias solicitações de vários usuários simultaneamente. Por isso, as entradas para uma sessão particular (todas as requisições feitas por um usuário), não são contíguas. Os registros individuais de uma sessão estarão espalhadas por todo o arquivo de log do servidor.

Tendo isso em mente e observando os passos necessários na transferência de um arquivo, ilustrados na Figura 4 página 8, pode-se concluir quais são as informações que podem ser armazenadas no log[4]:

host endereço do computador do usuário. Essa informação vem do protocolo TCP/IP, que é o protocolo utilizado na Internet e que possibilita a conexão entre dois computadores. O servidor precisa conhecer o endereço do cliente para poder enviar a resposta através da Internet.

ident alguns clientes possuem, em sua máquina, um programa chamado `identd`, que possibilita que o servidor Web saiba informações sobre o usuário que o está acessando. Caso isso aconteça, essa informação será armazenada.

authuser o protocolo HTTP permite um tipo de segurança básica [Group, 1999b], na qual o usuário deve se identificar com um `login` e uma `senha`. Caso a requisição tenha sido de um arquivo protegido por esse tipo de autenticação, o nome do usuário será registrado.

date também são registradas a data e a hora em que foi feita a solicitação de um arquivo.

request o nome do arquivo que foi requisitado é outra informação que é armazenada no log. Na verdade, o `request` é mais que isso: ele contém a string de requisição, o que inclui,

[4]Os termos utilizados para nomear as informações que são armazenadas no log estão em inglês porque é esta é a forma usual de referenciá-los.

além do nome do arquivo requisitado, o método que deve ser utilizado na recuperação do mesmo. Um exemplo de uma string de requisição seria `GET texto.html`, conforme visto na Figura 4 página 8.

status o protocolo HTTP retorna para o cliente, além do arquivo requisitado, um código de status, indicando o sucesso da requisição ou, em outras situações, alguma anomalia como, por exemplo, "arquivo não encontrado", "autorização negada", e outros.

bytes o número de bytes retornado para o cliente, excluindo os cabeçalhos do protocolo HTTP, ou seja, o tamanho do arquivo requisitado.

Essas são as informações básicas que são observadas numa interação entre um browser e um servidor Web. As informações como `ident`, `authuser`, `status` e `bytes`, apesar de serem básicas, necessitaram uma visão um pouco mais aprofundada do protocolo HTTP para serem determinadas. Mas todas essas informações não são as únicas que podem ser obtidas. Algumas outras podem ser registradas no log pois também circulam, através do protocolo HTTP, na comunicação entre o computador cliente e servidor Web, tais como:

user-agent nome e versão do browser sendo utilizado pelo usuário.

referer quando clica-se num link de uma página que está sendo mostrada no browser, este pode enviar para o servidor Web o endereço da página atualmente sendo mostrada. Desta forma, pode-se saber em que página havia um link para um arquivo armazenado no servidor que está realizando a requisição. A Figura 6 descreve um navegador mostrando a página `hub.html` que encontra-se no servidor `www.outrosite.com`. Essa página possui um link para o arquivo `texto.html` que encontra-se no servidor `www.nossosite.com`. Se o usuário clicar no link, uma requisição será feita para o servidor `www.nossosite.com` e ele poderá saber que esse link estava no endereço `http://www.outrosite.com/hub.html`.

Normalmente, os servidores Web estão configurados para armazenar as requisições num formato chamado CLF (Common Log Format) [Hallam-Baker and Behlendorf, 1996]. Um arquivo no formato CLF contém uma linha separada para cada requisição. Uma linha é composta por vários *tokens* separados por espaços. Não por coincidência, esses *tokens* são exatamente as informações básicas apresentadas anteriormente:

```
host ident authuser date request status bytes
```

Se um *token* não possui valor definido, então esse valor é representado por um hífen (`-`). Normalmente, os valores de `ident` e `authuser` estarão faltando por serem pouco utilizados.

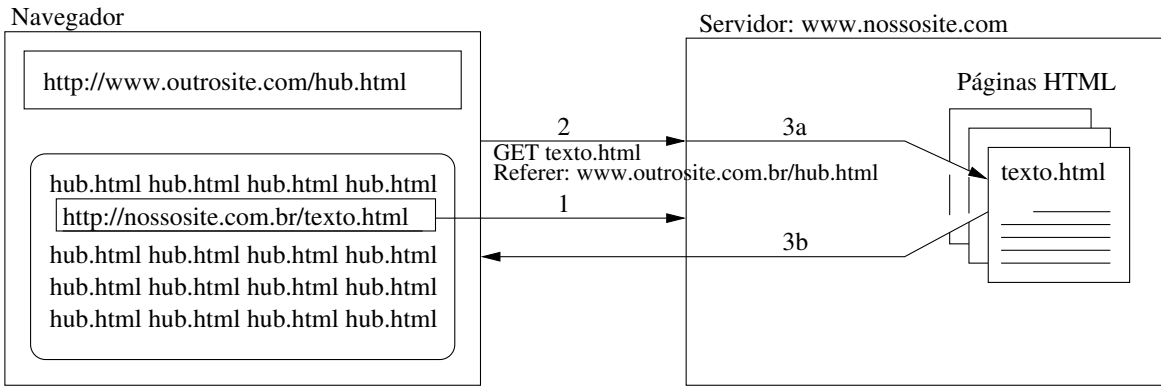

Figura 6: O protocolo HTTP também carrega outras informações, como o `referer`

Muitos webmasters[5] se utilizam de ferramentas como o Webalizer[6] para obter estatísticas de acessos aos sites pelos quais são responsáveis. Por isso, a maioria dos servidores Web são configurados para armazenar, também, os *tokens* `user-agent` e o `referer`, de forma que análises mais detalhadas possam ser feitas por esses tipos de ferramentas.

3.2 Interações Problemáticas

Quando um navegador faz uma solicitação HTTP, nem sempre ela é satisfeita pelo servidor Web. Isso se deve ao fato de que muitos provedores de Internet utilizam servidores *proxy* para reduzir o tráfego na rede. Esses servidores são utilizados para armazenar, em cache, os conteúdos freqüentemente solicitados. A Figura 7 ilustra esse caso.

A utilização de um servidor *proxy* introduz um problema no registro dos acessos no arquivo de log pois, muitas vezes, os servidores *proxy* não notificam adequadamente o servidor Web de que a requisição foi satisfeita por ele. Dessa forma, o servidor Web fica sem saber que um acesso foi feito e essa informação não é registrada no arquivo de log.

Os caches do navegador também introduzem incertezas na tentativa de monitorar todos os eventos que ocorrem durante uma sessão[7] de usuário. A maioria dos navegadores armazena uma cópia dos objetos (páginas, figuras, etc.) recuperados recentemente em um cache local. Assim, em vez de obter o arquivo do servidor, o navegador o recupera do cache e o servidor Web não fica sabendo disso, não podendo registrar o acesso.

Um outro tipo de incerteza surge quando um usuário executa vários navegadores para um mesmo site. Nesse caso, a informação da seqüencia de acessos perde-se. Para entender o porquê isso acontece, basta imaginar dois navegadores abertos, mostrando uma mesma página do site. O usuário pode clicar, então, em dois links diferentes, um em cada página.

[5] Pessoa responsável pela manutenção de um site.

[6] `http://www.webalizer.org`

[7] Uma sessão de usuário compreende o conjunto de páginas, figuras e outros arquivos armazenados no servidor Web, durante o período de tempo em que ele ficou acessando o site.

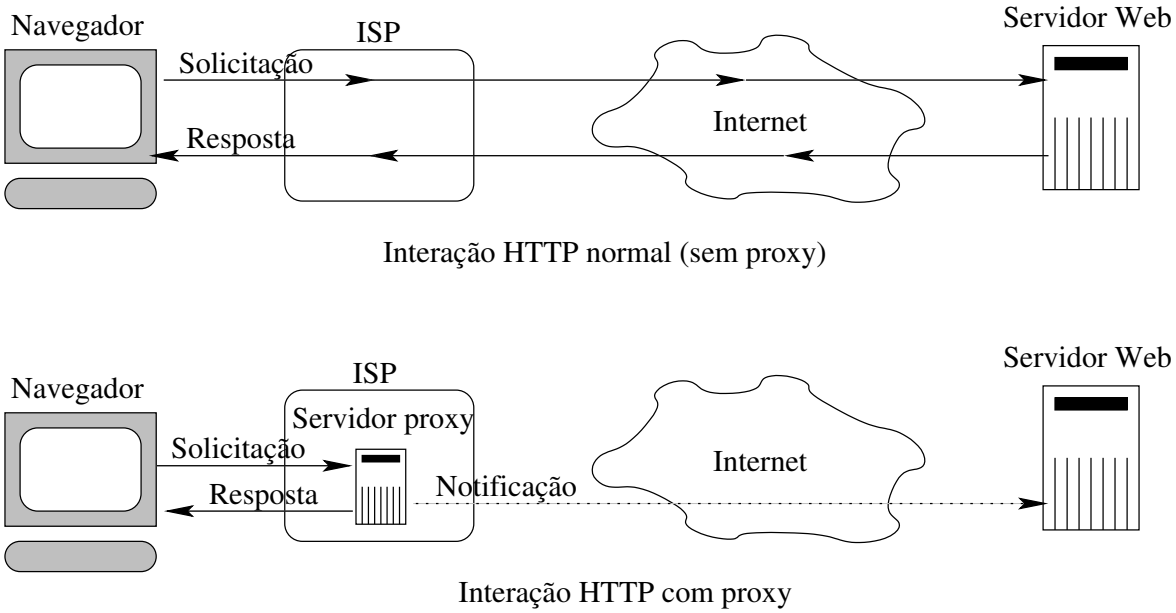

Figura 7: Funcionamento de um servidor *proxy*

Ao se olhar para o log, os dois acessos são vistos, mas não se sabe se o usuário clicou em um link, voltou para a página original (que seria recuperada do cache do navegador) e, em seguida, clicou no outro link. A situação tende a piorar porque, provavelmente, o usuário continuará a fazer acessos através dos dois navegadores independentes ou então pode abrir novas janelas.

A conclusão disso tudo é a de que não se pode ter certeza de se ter um registro completo de todas as ações do usuário. Entretanto, apesar dessas incertezas, é possível extrair bastantes informações dos arquivos de log de servidores Web [Kimball and Merz, 2000].

4 O Filtro

O filtro `log2discovery.pl` foi por nós desenvolvido com a finalidade de transformar arquivos de log de servidores Web em arquivos na sintaxe padrão do Sistema Discover, com o objetivo de extrair informações e adquirir conhecimento utilizando os dados desses arquivos de log e as ferramentas que constituem o Discover.

No desenvolvimento do filtro `log2discovery.pl`, foi utilizada a linguagem *freeware* Perl [Wall and Schwartz, 199 que é a mesma que foi utilizada no desenvolvimento do Sistema Discover. Além disso, a própria linguagem Perl foi desenvolvida como uma ferramenta para processamento de *strings*, o que facilita muito o desenvolvimento de um filtro como o proposto neste trabalho.

4.1 Definição do arquivo `.names`

Antes de iniciar o desenvolvimento do filtro, a estrutura do arquivo `.names` teve que ser definida. Procurou-se, para isso, incluir os campos que um arquivo de log possui e que foram considerados valiosos para um primeiro estudo na área de Web Usage Mining [8]. Também tomou-se o cuidado de separar os campos em campos atômicos, ou seja, que não possuem duas informações juntas, como é o caso da data e hora, as quais foram separados em campos distintos. Também foram incluídos campos que não necessariamente têm dados neste primeiro instante, ou seja, não têm campos equivalentes nos logs, mas que consideramos importantes para serem estudados em trabalhos futuros.

Para nomear os campos do arquivo `.names`, foi utilizada uma variação da nomenclatura proposta pelo World-Wide-Web Consortium (W3C) [Hallam-Baker and Behlendorf, 1996]. Nessa nomenclatura, os nomes dos campos podem ter uma das seguintes formas:

- **<identifier>** O nome do campo está relacionado com a transação HTTP como um todo. O identificador `data`, por exemplo, indica o momento em que a transação, como um todo, foi executada.
- **<prefix>_<identifier>** Identifica uma informação específica que é transferida na transação. O prefixo indica de quem é essa informação ou o fluxo que é seguido. O IP do cliente (c), por exemplo, é indicado por `c_ip` e o do servidor (s) por `s_ip`.
- **<prefix>h_<header>** Identifica uma informação específica que é transferida através do cabeçalho (**h**eader) do protocolo HTTP. O prefixo tem o mesmo significado que no formato anterior.

Os prefixos que podem ser utilizados são os seguintes:

- **c** Cliente.
- **s** Servidor.
- **cs** Do cliente para o servidor.
- **sc** Do servidor para o cliente.
- **x** Identificador de uma aplicação específica.

Assim, um campo com nome `cs_method` indica qual o método de requisição enviado do cliente para o servidor enquanto que `csh_referer` refere-se ao campo `referer` transferido, do cliente para o servidor, no cabeçalho do protocolo HTTP.

A seguir, é dada uma lista dos campos que estão presentes no arquivo `.names` por nós definido:

[8] Web Usage Mining é a sub-área de Web Mining que trata, basicamente, das técnicas de aplicação de Mineração de Dados em logs de servidores Web.

c_ip O endereço de Internet do cliente que fez a requisição. Esse é o endereço para o qual a resposta do servidor será enviada. A maioria dos computadores em rede não têm endereços de Internet fixos. Em vez disso, o endereço é atribuído dinamicamente para o computador quando o usuário faz uma conexão com o seu provedor através do *modem*. Mesmo que o endereço seja dinâmico, ele permanece o mesmo durante uma sessão e pode ser utilizado para "amarrar" os eventos de uma sessão.

c_dns O filtro está preparado para traduzir os endereços IPs para o respectivo nome de domínio. O nome traduzido é armazenado neste campo. Nem sempre é possível fazer essa tradução. Nesses casos, o campo conterá um valor desconhecido, indicado pelo símbolo '`?`'.

c_userid O nome de usuário da pessoa requisitando o documento, conforme determinado pelo protocolo de autenticação do HTTP [Group, 1999b]. Se o documento sendo requisitado não está protegido por senha, então este campo armazenará um valor desconhecido.

date A data na qual a transação HTTP foi completada.

time A hora na qual a transação HTTP foi completada.

cs_method Este campo armazena o nome do método utilizado para solicitar um documento do servidor. Para melhor entender o que é o método de requisição, considere o seguinte exemplo de requisição HTTP:

```
GET /manual/index.html HTTP/1.0
```

Neste caso, `GET` é o método do HTTP sendo utilizado pelo navegador para a requisição do arquivo `/manual/index.html`. Os dois métodos mais comumente utilizados do HTTP são o `GET` e o `POST`. Os dois diferem apenas na forma como as informações de uma página são enviados para o servidor.

cs_uri URIs (Uniform Resource Identifier) são *strings* que identificam recursos (documentos, imagens, arquivos para download, serviços, caixas de e-mail, etc.) na Web. No caso do campo `cs_uri`, ele indica a localização do arquivo que o cliente deseja obter do servidor. Observando o exemplo de requisição mostrado no ítem anterior, observa-se que o cliente deseja obter o arquivo `index.html`. A string `/manual/index.html` é a URI deste arquivo. No caso do recurso sendo requisitado ser um serviço, é provável que a URI contenha parâmetros de entrada para o serviço, como no seguinte exemplo tirado do site Google[9]:

```
GET /search?hl=pt&ie=UTF-8&oe=UTF-8&q=URI&lr= HTTP/1.0
```

[9] http://www.google.com.br

Nesse caso, o serviço sendo requisitado é `search` e os parâmetros estão na *string* que seque o caractere '`?`' — `hl=pt&ie=UTF-8&oe=UTF-8&q=URI&lr=`. Nota-se que os parâmetros são separados por caracteres '`&`'. Páginas dinâmicas também seguem o mesmo esquema de parâmetro, ou seja, um caractere '`?`' separando o recurso sendo requisitado dos parâmetros que serão enviados para o recurso[10].

cs_uri_stem A URI sem os parâmetros.

cs_uri_query Somente os parâmetros da URI.

cs_version A versão do protocolo HTTP sendo utilizado. Nota-se que quem indica qual versão do protocolo utilizar é o cliente, pois essa informação trafega do cliente para o servidor.

sc_status O resultado da transação, como definido no protocolo HTTP ([Group, 1999a]).

sc_bytes O número de bytes transferidos do servidor para o cliente.

csh_referer O `referer` é uma informação que trafega no cabeçalho do protocolo e indica a URI de origem da URI (`cs_uri`) sendo requisitada, como descrito na seção 3.1 página 9.

csh_referer_stem A URI do `referer` sem os parâmetros.

csh_referer_query Somente os parâmetros passados para o `referer`.

csh_user_agent O nome e/ou a versão do navegador sendo utilizado pelo cliente.

Os próximos campos contém somente o valor '`?`' (valor desconhecido) pois foi decidido incluí-los para serem considerados em trabalhos futuro.

csh_cookie Algumas versões Apache (nome de um servidor Web muito utilizado) têm a possibilidade de ativar um módulo que permite rastrear um usuário através de *cookies*. *Cookies* são arquivos enviados pelo servidor e que o navegador guarda no disco da máquina cliente. Depois, toda vez que o navegador visitar *este* site, ele envia este arquivo de volta para o servidor. Assim, o campo `csh_cookie` serve para guardar o número utilizado para rastrear o usuário, na tentativa de se identificar, no log, qual a origem da requisição. O motivo de considerar este campo como trabalho futuro é que o módulo que permite rastrear um usuário não é ativado por *default* no Apache.

x_session Este campo não está presente no arquivo de log, mas é uma informação secundária que tem que ser extraída de uma combinação de registros do log. A sessão é uma forma de identificar o usuário que está acessando o servidor. Se o campos `csh_cookie`

[10] Observar que o símbolo '`?`' na *string* de requisição não tem relação com o símbolo utilizado como valor desconhecido na sintaxe padrão.

for usado, não há necessidade de se usar o `x_session`. Mas, no caso do `csh_cookie` não puder ser usado ou o servidor Web não suportar esta funcionalidade, o campo `x_session` é uma solução para rastrear os usuários. O problema com este campo é que não é fácil extrair esta informação, normalmente sendo infererida por heurísticas aplicadas no arquivo de log, não sendo, assim, uma informação precisa. No presente momento, o filtro não infere esta informação (trabalho futuro).

x_uri_type Este campo indica qual o tipo de recurso que foi requisitado. Este recurso pode ser um arquivo HTML, uma figura, um arquivo de *script* (como uma página dinâmica), entre outros. Esta informação não está no arquivo de log, devendo ser obtida de uma base de informações secundária, criada, por exemplo, pelo *webmaster* do site.

x_uri_subtype Este campo indica o sub-tipo do recurso sendo requisitado. Ele é uma informação que complementa o campo `x_uri_type`. Assim, se no campo `x_uri_type` tem-se a indicação de que o recurso requisitado é uma página HTML, o campo `x_uri_subtype` pode indicar que esta página é uma página de descrição de um produto, por exemplo.

x_referer_type Semelhante ao campo `x_uri_type`, mas aplicado ao `referer`.

x_referer_subtype Semelhante ao campo `x_uri_subtype`, mas aplicado ao `referer`.

4.2 Implementação

O filtro implementado lê um arquivo de log (entrada) e produz os dois arquivos de saída: o arquivo `.names` e o arquivo `.data` na sintaxe do Sistema Discover, conforme mostra a Figura 8.

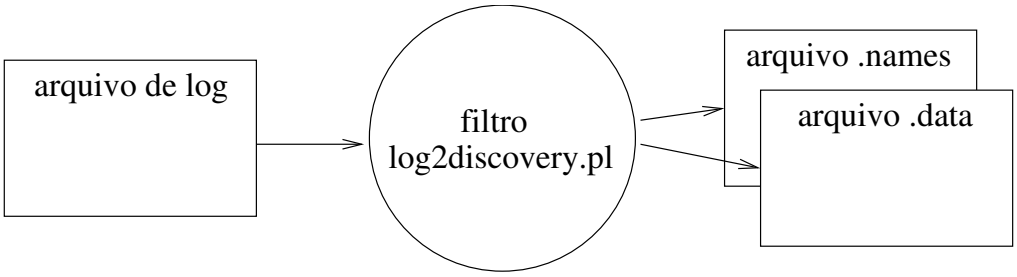

Figura 8: Entrada e saídas do filtro `log2discovery.pl`.

Assim, na implementação, o código do filtro foi dividido, basicamente, em três partes:

1. Obtenção dos parâmetros de entrada.
2. Criação do arquivo `.names`.
3. Criação do arquivo `.data`.

Essas partes estão descritas nas seções seguintes.

4.2.1 Obtenção dos parâmetros de entrada

O filtro `log2discovery.pl` aceita como entrada os seguintes parâmetros, listados a seguir:

-if (**i**nput **f**ile). Este é um parâmetro obrigatório que especifica qual o nome do arquivo de log que deve ser lido e processado.

-pr (**pr**efix). O prefixo indica qual o nome dos arquivos de saída que serão gerados. Se nada for especificado, o filtro utilizará o nome `log` e os arquivos de saída serão: `log.names` e `log.data`.

-id (**i**nitial **d**ate). Se uma data for especificada, serão processados somente os registros do arquivo de log com data posterior à especificada.

-fd (**f**inal **d**ate). Se uma data for especificada, serão processados somente os registro do arquivo de log com data anterior à especificada.

Para fazer o *parsing* da linha de comando, foi utilizado um pacote para Perl chamado `Getopt::Long`.

4.2.2 Criação do arquivo `.names`

Como já mencionado, o arquivo `.names` é o arquivo que descreve, no Sistema Discover, os campos dos registros que serão processados pelos algoritmos de Aprendizado de Máquina. Não há nada de especial na codificação desta parte do filtro: ele cria um arquivo cujo nome encontra-se especificado no parâmetro de entrada `-pr` e escreve nele um texto contendo os nomes e os tipos dos campos no formato padrão para arquivos de entrada do Sistema Discover.

O conteúdo final deste arquivo é mostrado na Figura 9:

4.2.3 Criação do arquivo `.data`

Para criar o arquivo `.data`, o filtro processa cada registro do arquivo de log, transformando esses registros em registros no formato padrão do Sistema Discover. O processamento de cada linha do arquivo de log envolve os seguintes passos:

1. tenta casar a linha do registro atual do log com o padrão que foi configurado no filtro. Esse padrão é, na verdade, uma expressão regular que casa com um registro do log gerado pelo servidor Apache (configuração padrão).

2. uma vez que o registro casou com a expressão regular, pode-se saber a data em que esse registro foi criado, filtrando-o de acordo com as datas iniciais e finais especificadas nos parâmetros `-id` e `-fd` de linha de comando.

```
| Class Attribute
null.

| Attributes
c_ip:               nominal.
c_dns:              string.
c_userid:           nominal.
date:               date.
time:               time.
cs_method:          nominal.
cs_uri:             string.
cs_uri_stem:        nominal.
cs_uri_query:       string.
cs_version:         nominal.
sc_status:          nominal.
sc_bytes:           integer.
csh_referer:        string.
csh_referer_stem:   nominal.
csh_referer_query:  string.
csh_user_agent:     string.
csh_cookie:         string.
x_session:          nominal.
x_uri_type:         nominal.
x_uri_subtype:      nominal.
x_referer_type:     nominal.
x_referer_subtype:  nominal.
```

Figura 9: Conteúdo do arquivo `.names` gerado pelo filtro.

3. além da data, os outros campos que formam um registro do log são separados e transformados para o formato padrão, utilizando-se funções apropriadas. Essas funções são específicas para cada tipo de campo e agem como mini-filtros, lidando, também, com valores desconhecidos. Por exemplo, para um campo do tipo `nominal` é chamada a função que sabe como lidar com valores nominais.

4. finalmente, depois de processados todos os campos do registro atual, o resultado é escrito no arquivo de saída.

4.3 Exemplos de Entradas e Saídas

Nesta seção é mostrada a ação do filtro sobre alguns registros de log, na criação do arquivo `.data`. Para isso, foi criado um arquivo de log de nome `log.test` com o seguinte conteúdo:

1. `143.107.183.226 - - [02/Jul/2002:15:47:20 -0300] "GET /manuals/sce183/proc.html HTTP/1.1" 200 1329 "http://www.icmc.sc.usp.br/manuals/sce183/conteudo.html" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"`

2. `143.107.183.67 - - [02/Jul/2002:15:47:10 -0300] "GET /imagens/pessoas1.gif HTTP/1.1" 304 - "http://www.icmc.sc.usp.br/" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"`

3. `200.131.15.49 - - [02/Jul/2002:19:46:21 -0300] "GET /~poscomp/ HTTP/1.1" 200 4410 "http://mail.passosuemg.br/webmail/src/read_body.php?mailbox=INBOX&passed_id=9&startMessage=1&show_more=0" "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)"`

4. `143.107.231.61 - - [02/Jul/2002:17:02:57 -0300] "GET /intranet/serv/altser.php?rowid=715 HTTP/1.1" 200 7545 "http://www.icmc.sc.usp.br/intranet/serv/pesqser.php" "Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; Win 9x 4.90)"`

A seguir, o filtro foi executado com a seguinte linha de comando:

```
./log2discovery.pl -if log.test
```

criando, assim, os arquivos `log.names` e `log.data`. O arquivo `log.names` é o mostrado na Figura 9 e o arquivo `log.data` gerado contém os seguintes quatro registros correspondentes:

1. `143.107.183.226, ?, ?, 2002/07/02, 15:47:20, GET, "/manuals/sce183/proc.html", /manuals/sce183/proc.html, ?, HTTP/1.1, 200, 1329, "http://www.icmc.sc.usp.br/manuals/sce183/conteudo.html", http://www.icmc.sc.usp.br/manuals/sce183/conteudo.html, ?, "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)", ?, ?, ?, ?, ?, ?`

2. 143.107.183.67, ?, ?, 2002/07/02, 15:47:10, GET, "/imagens/pessoas1.gif" , /imagens/pessoas1.gif, ?, HTTP/1.1, 304, ?, "http://www.icmc.sc.usp.br /", http://www.icmc.sc.usp.br/, ?, "Mozilla/4.0 (compatible; MSIE 6.0; W indows NT 5.0)", ?, ?, ?, ?, ?, ?

3. 200.131.15.49, ?, ?, 2002/07/02, 19:46:21, GET, "/~poscomp/", /~poscomp/ , ?, HTTP/1.1, 200, 4410, "http://mail.passosuemg.br/webmail/src/read_bo dy.php?mailbox=INBOX&passed_id=9&startMessage=1&show_more=0", http://mai l.passosuemg.br/webmail/src/read_body.php, "mailbox=INBOX&passed_id=9&st artMessage=1&show_more=0", "Mozilla/4.0 (compatible; MSIE 5.5; Windows 9 8)", ?, ?, ?, ?, ?, ?

4. 143.107.231.61, ?, ?, 2002/07/02, 17:02:57, GET, "/intranet/serv/altser. php?rowid=715", /intranet/serv/altser.php, "rowid=715", HTTP/1.1, 200, 7 545, "http://www.icmc.sc.usp.br/intranet/serv/pesqser.php", http://www.i cmc.sc.usp.br/intranet/serv/pesqser.php, ?, "Mozilla/4.0 (compatible; MS IE 6.0; Windows 98; Win 9x 4.90)", ?, ?, ?, ?, ?, ?

4.4 Alterando o Filtro

Atualmente, há uma variedade de servidores Web sendo utilizados e cada um gera um arquivo de log em um formato diferente. Além disso, alguns servidores Web (como o Apache, mencionado anteriormente) permitem que o administrador do site crie seu próprio formato de log.

O filtro `log2discovery.pl` foi desenvolvido levando-se em consideração essa diversidade de servidores e permite que se altere a forma que ele lê o arquivo de log, permitindo assim, ler, virtualmente, qualquer tipo de arquivo de log, alterando um mínimo do código implementado.

Para se fazer uma alteração, deve-se modificar somente as primeiras partes do código do filtro, onde se encontram as partes de código que configuram o formato do arquivo de log, descritas a seguir.

1. A primeira parte do código contém a configuração dos padrões dos campos do log. Ou seja, ela contém expressões regulares que devem casar com cada um dos campos que existem. Por exemplo, a seguinte linha de código Perl mostra como identificar uma data em uma linha de registro do arquivo de log:

   ```
   my $pat_date = "(\\d{2}/\\w{3}/\\d{4})";
   ```

   Note que deve-se colocar uma "\" onde for necessário, como é o caso da própria "\". Isso acontece porque, na verdade, a variável `$pat_date` é uma *string* que contém uma

expressão regular e essa expressão contém algumas "instruções" que necessitam da "\", como "\d" que casa com um dígito. Assim, se quer-se uma string que contenha a expressão regular que casa com um dígito ("\d") é necessário colocar uma "\" após a "\" do padrão "\d", ou seja:

- O padrão que se quer: `\d`
- A *string* que corresponde ao padrão que se quer: `\\d`

Também é importante notar que há parênteses envolvendo cada padrão. Eles são utilizados para criar as variáveis em Perl (`$1`, `$2`, etc.) que referenciam cada *string* que casa com cada uma das expressões. Elas serão passadas como argumentos para as funções que sabem como lidar com cada tipo de dado.

2. Nesta parte do código tem-se a variável que junta todos os padrões de campos em um único padrão, o qual mostra como é o formato de toda uma linha do arquivo de log.

```
my $pattern  = "^$pat_ip $pat_identd $pat_userid ";
   $pattern .= "\\[$pat_date:$pat_time $pat_zone\\] ";
   $pattern .= "\"$pat_method $pat_uri $pat_version\" ";
   $pattern .= "$pat_status $pat_bytes ";
   $pattern .= "\"$pat_referer\" \"$pat_user_agent\"\$";
```

A variável `$pattern` é uma *string* que contém o formato de uma linha do log. Portanto, para configurar o filtro para ler arquivos de logs de outros servidores Web diferentes do Apache, basicamente deve ser alterada a variável `$pattern`. A observação anterior sobre os caracteres "\" também é válida aqui.

3. A última parte da configuração é uma variável do tipo *hash table* que contém informações de como cada campo deve ser processado. Cada entrada dessa tabela contém uma referência para a função que sabe como lidar com cada tipo de campo específico. Uma parte do código implementado ilustra isto:

```
my %fields = (
    'c_ip'              => '&nominal($1)',
    'c_dns'             => '&string(&calcDNS($1))',
    'c_userid'          => '&nominal($3)',
    'date'              => '&date($4)',
    'time'              => '&time($5, $6)',
(...)
    'x_referer_subtype' => '&nominal("-")' );
```

Nota-se que, para cada campo, é indicada qual função chamar e que os parâmetros destas funções são variáveis na forma $\$n$ onde n é a ordem em que o campo sendo configurado aparece na variável $\$pattern$.

Portanto, para realmente trocar o tipo de arquivo de log que o filtro aceita como entrada, basta configurar a variável `$pattern` e suas variáveis auxiliares, ou seja, as variáveis que mostram a expressão regular para cada campo individual. Também deve-se tomar o cuidado de alterar a variável do tipo *hash table* para indicar os parâmetros passados para as funções, bem como se é a função apropriada para manipular o tipo de campo específico.

5 Conclusão

Neste trabalho foi descrita a implementação de um filtro que transforma logs de servidores Web em arquivos de dados na sintaxe padrão do Sistema Discover, o que permitirá gerenciar a grande quantidade de dados que um arquivo de log armazena. Após realizar a transformação, é possível iniciar o processo de extração de conhecimento desses logs utilizando as facilidades implementadas no Sistema Discover.

Também foi descrito o funcionamento da Web de forma que se tenha uma referência para o entendimento dos dados que existem nos arquivos de log. Foi mostrado que há limitações na quantidade de informações que são registradas nos logs e, dessa forma, algumas dificuldades serão encontradas ao se realizar uma mineração muito abrangente dos mesmos pois essa mineração será restringida pela limitação de informações nos dados.

Uma breve descrição do Sistema Discover foi feita, mostrando a idealização do projeto e o porquê que se criar uma sintaxe padrão para os arquivos de entrada dos algoritmos de Aprendizado de Máquina.

Referências

[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). Sintaxe padrão do arquivo de exemplos do projeto Discover. http://www.icmc.sc.usp.br/~gbatista/Discover/SintaxePadraoFinal.htm.

[Batista, 2003] Batista, G. E. A. P. A. (2003). A biblioteca da sintaxe padrão para arquivos de exemplos do Sistema Discover. Technical report, ICMC-USP. Trabalho em Andamento.

[Chiara, 2001] Chiara, R. (2001). Aplicação de data mining em logs de servidores web. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP.

[Dosualdo, 2002] Dosualdo, D. G. (2002). Investigação de regressão para data mining. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP.

[Geromini, 2002] Geromini, M. R. (2002). Projeto e desenvolvimento de uma interface gráfica para o ambiente de descoberta de conhecimento DISCOVER. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP.

[Group, 1999a] Group, N. W. (1999a). *RFC2616 - Hypertext Transfer Protocol – HTTP/1.1*.

[Group, 1999b] Group, N. W. (1999b). *RFC2617 - HTTP Authentication: Basic and Digest Access Authentication*.

[Hallam-Baker and Behlendorf, 1996] Hallam-Baker, P. M. and Behlendorf, B. (1996). Extended log file format. Technical report, W3C. http://www.w3c.org/pub/WWW/TR/WD-logfile.html.

[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 DISCOVER. Technical Report 143, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_143.ps.zip.

[Kemp et al., 2002] Kemp, A. H., Batista, G. E. A. P. A., and Monard, M. C. (2002). Descrição da implementação dos filtros para recuperação da taxa de erro dos algoritmos de aprendizado de máquina usado no ambiente DISCOVER. Technical Report 175, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_175.ps.zip.

[Kimball and Merz, 2000] Kimball, R. and Merz, R. (2000). *The Data Webhouse Toolkit*. John Wiley and Sons, Inc.

[Kosala and Blockeel, 2000] Kosala, R. and Blockeel, H. (2000). Web mining reseach: A survey. In *SIGKDD Explorations*, volume 2, pages 1–15.

[Martins, 2001] Martins, C. A. (2001). Interpretação de clusters em Aprendizado de Máquina. Monografia para o Exame de Qualificação de Doutorado, ICMC-USP.

[Melanda, 2002] Melanda, E. (2002). Pós-processamento de conhecimento de regras de associação. Monografia para o Exame de Qualificação de Doutorado, ICMC-USP.

[Prati, 2002] Prati, R. C. (2002). Projeto e implementação do framework de integração do Sistema DISCOVER. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP.

[Prati et al., 2001a] Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2001a). 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.

[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.

[Project, 2001] Project, A. H. S. D. (2001). *Apache Web Server Documentation*. Apache Foundation.

[Spiliopoulou et al., 1999] Spiliopoulou, M., Pohle, C., and Faulstich, L. (1999). Improving the effectiveness of a web site with web usage mining. In *Proceedings of the Workshop on Web Usage Analysis and User Profiling, WEBKDD '99*, pages 51–56.

[Wall and Schwartz, 1991] Wall, L. and Schwartz, R. (1991). *Programming PERL*. O´Reilly & Associates.

[Weiss and Indurkhya, 1998] Weiss, S. and Indurkhya, N. (1998). *Predictive Data Mining*. Morgan Kaufmann Publishers.

[Zaïane et al., 1998] Zaïane, O. R., Xin, M., and Han, J. (1998). Discovering web access patterns and trends by applying OLAP and data mining technology on web logs. In *Advances in Digital Libraries*, pages 19–29.