ht://Miner Tutorial

Gabriele Bartolini `<g.bartolini@comune.prato.it>`

Table of Contents

C++ System for Web usage mining and data warehouse: it allows the discovery of knowledge from data (KDD) regarding users' usage on the Web (such as unique visitors, sessions, transactions) and organises it in a PostgreSQL multi-stage data warehouse.

Getting Started

Introduction

ht://Miner is an *open-source* system for Web usage mining.

Web usage mining is a specific branch of Web mining, first defined by *Etzioni* in 1996 as the "use of data mining techniques to automatically discover and extract information from World Wide Web documents and services".

The term Web usage mining first appeared in 1997, when *Cooley et al.* defined a taxonomy for the Web Mining discipline. Since then, we refer to Web usage mining as the *automatic discovery of user access*

patterns from Web servers.

ht://Miner is structured in order to perform the main operations of a system for *KDD* (*Knowledge Discovery from Data*), which are:

1. Data pre-processing (selection, filtering and coding)

2. Data mining

3. Analysis

4. Decisions making

Although the system is designed to be expanded and to interface itself with several DBMS, currently ht://Miner supports PostgreSQL. The data architecture is organised in two stages:

1. *Transactional data* for On-Line Transactional Processing purposes (OLTP)

2. *Data warehouse* for On-Line Analytical Processing purposes (OLAP)

The first data source is *fully normalised*, contains all the data and it is organised to perform *specific queries*. The second data source is organised to *privilege speed* rather than redundancy, in order to improve performances of *aggregate functions* over time periods such as days, months, semesters, years.

In order to produce reports, Web usage data produced by ht://Miner can be queried through:

1. *third party applications* for data mining and data reporting through ODBC or PostgreSQL client library

2. ht://Miner's *PHP middleware* library

3. customised applications that directly interface themselves with the database

Installation and configuration

Requirements

ht://Miner requires a GNU/Linux system with:

*   *kernel 2.4* or higher

*   *GNU C/C++* compiler (with *libstdc++*)

*   *PostgreSQL 8.x* [pgsql]

*   *GNU gettext* library [gettext]

*   *PCRE* (Perl Compatibile Regular Expressions) library [pcre]

- *zlib* library [zlib]

- *Open SSL* library [openssl]

- *eXpat* library (XML parser) [expat]

Although it has not been tested on different platform, ht://Miner has been written using ANSI C++ and should be easily ported on POSIX compliant systems.

GeoIP library support

ht://Miner is designed to support the GeoIP C library from Maxming [geoipcapi]. This library allows the system to locate the Internet visitors based on their IP address. The accuracy of this operation depends on the type of database used. Maxmind currently offers two databases for free:

1. GeoLite Country [geolitecountry]

2. GeoLite City [geolitecity]

ht://Miner supports both. For more information visit the section in the ht://Miner website.

The GeoIP support can be enabled through the *—enable-geoip* option of the configure script.

```
$ ./configure —enable-geoip [other options]
```

Installation from sources

Getting the sources

ht://Miner's sources are available from the SourceForge file releases system and are usually shipped as compressed TAR archives (bzip2 and gzip).

Uncompressing the sources

For tar archive files that have been compressed using the gzip algorithm, type:

```
tar xzvf filename.tar.gz
```

For tar archive files that have been compressed using the bzip2 algorithm, type:

```
tar xjvf filename.tar.bz2
```

Quick install

In case you performed a system wide installation of the required libraries (which means that these libraries stand in directories such as /usr/lib or /usr/local/lib and their headers in /usr/include or /usr/local/include), you should be able to install ht://Miner by simply typing:

```
$ cd htminer-version
```

```
$ ./configure
$ make
$ su
# make install
```

Configuring the source

By default ht://Miner is installed under the `/usr/local/htminer` directory. However, the system supports the standard *configure* methods for customising the application's locations in the system. For more information on the available options, type:

```
$ ./configure —help
```

For instance, if you need to install ht://Miner under a different directory (e.g. /usr), type:

```
$ ./configure —prefix=/usr
```

Compiling the source

Sources compilation is performed through the GNU make application. Once you have configured the sources, type:

```
$ make
```

In order to install the application, depending on the location, you may need to become superuser and then type:

```
$ make install
```

Preparing the database

Note

Currently ht://Miner supports solely the PostgreSQL RDBMS. Therefore the following information refer to this particular product.

Users' creation

The first step is to create the *owner* of the ht://Miner data sources. The username to be created is *htminer*. You can use your favourite method for the user's creation:

- `createuser` console application

- `CREATE USER` SQL command given via the `psql` console application

- pgAdmin

- phppgadmin's web interface

For more information on users' creation in PostgreSQL, read the documentation about the CREATE USER

[http://www.postgresql.org/docs/8.1/interactive/sql-createuser.html] command or the 'Database Roles and Privileges' chapter [http://www.postgresql.org/docs/8.1/interactive/user-manag.html].

Usually you may want to create a user that has *login* privileges but does not have the right to either create a database or a user. You can do this by following the steps you find below (please change the password):

```
$ su - postgres
$ psql
postgres=> CREATE ROLE htminer LOGIN NOSUPERUSER
    NOCREATEDB NOCREATEROLE ENCRYPTED PASSWORD 'changeme';
```

Warning

It is important that you understand the basics of PostgreSQL administration, especially what *roles* are and how they work. In particular, you need to understand how access restrictions work (through the `pg_hba.conf` file).

The second step is the creation of the user that can access data for reporting only. By default the user is called *htreport*:

```
postgres=> CREATE ROLE htreport LOGIN NOSUPERUSER
    NOCREATEDB NOCREATEROLE ENCRYPTED PASSWORD 'changeme';
```

Tablespaces' creation

One of the most important features that PostgreSQL 8.x has brought are *tablespaces*. As PostgreSQL documentation says, "tablespaces allow administrators to select different file systems for storage of individual tables, indexes, and databases. This improves performance and control over disk space usage".

The use of tablespaces however is strictly tied to the size of your organisation. Depending on the number of requests you may want to spread your data over multiple disks, in order to improve efficiency and data integrity. For our purposes, we have decided to store the two data source stages in two different locations on two different disks. This can be easily achieved in PostgreSQL.

Suppose you have the `/disk1` and `/disk2` locations for the first and second disk respectively. You can create and register two tablespaces in PostgreSQL by simply performing the following steps.

```
$ su
$ mkdir /disk1/htminer_oltp
$ mkdir /disk2/htminer_warehouse
$ chown postgres /disk1/htminer_oltp
$ chmod 770 /disk1/htminer_oltp
$ chown postgres /disk2/htminer_warehouse
$ chmod 770 /disk2/htminer_warehouse
$ su - postgres
$ psql
postgres=> CREATE TABLESPACE htminer_oltp OWNER htminer
  LOCATION '/disk1/htminer_oltp';
postgres=> CREATE TABLESPACE htminer_warehouse OWNER htminer
  LOCATION '/disk1/htminer_warehouse';
```

Warning

Before proceeding with tablespaces creation, it is recommended that you read PostgreSQL documentation on the topic, in particular the CREATE TABLESPACE [http://www.postgresql.org/docs/8.1/interactive/sql-createtablespace.html] SQL command.

Database creation

ht://Miner stores everything in one single database. However, tables are organised using PostgreSQL's *schemas*, which allows to group tables into logically connected relations.

The final step for database preparation is the creation of the database itself. The database, by default, will be called *htminer*.

```
$ su - postgres
$ psql
postgres=> CREATE DATABASE htminer WITH OWNER = htminer
   ENCODING = 'LATIN1';
```

If you want, you can specify a default tablespace (it is recommended that you choose the warehouse tablespace, as it is the one where new tables and indexes are created more often due to horizontal partitioning of tables [http://en.wikipedia.org/wiki/Partition_(database)] - per month and per year).

```
postgres=> CREATE DATABASE htminer WITH OWNER = htminer
   ENCODING = 'LATIN1'
   TABLESPACE = htminer_warehouse;
```

The *htminer* database is now ready to be used by the suite of tools of ht://Miner. Each of them involves a particular step of the KDD process.

Step one - Log collection

One of the fundamental parts of ht://Miner is log collection. In particular, ht://Miner is able to handle both *Apache* [http://httpd.apache.org] and Microsoft Internet Information Server formats for access logs.

Apache web server's log format

ht://Miner needs more information than a common log format, in particular:

1. resource's content type

2. server's IP address

3. server's host name (*virtual host*)

4. server's port

5. resource's content language

Here you can find the instructions that need to be added to Apache's `httpd.conf` file in order to prepare ht://Miner compliant log files.

```
LogFormat \"%h\t%u\t%t\t\"%r\"\t%\>s\t%b\t\"%{Referer}i\"\t\"%{User-Agent}i\"\t%A\t%v\t%p\t%T
CustomLog admin/access_log htminer
```

Note

You can change the location and the file name for the log file as you like.

Internet Information Server's log format

ht://Miner is able to detect Internet Information Server's log format on the fly, by performing a runtime parsing of the IIS headers. You just need to enable all the fields from the control panel for the logging module of IIS.

Step two - Log import

The very first phase of the ht://Miner's discovery process is to import the access log data in a temporary table. This process allows to:

1. bulk load accesses from different web servers

2. organise requests in sessions by simply grouping the available data by *user*, *origin IP address* and *user agent* (browser)

3. order requests within a single session by request time

This process has a huge impact in the potential of the information that is discovered, as it allows to identify sessions coming from different servers (managed by the same organisation) where requests are not tied together.

The application responsible for performing data import is called *htminer_importer*. For more information regarding its usage, you can type:

```
$ htminer_importer —help
```

The application by default uses the `importer.xml` configuration file which sits in the *configuration* directory (by default `/usr/local/htminer/conf`).

For example, in order to load the `access_log.20061107` file, you simply type:

```
$ htminer_importer access_log.20061107
```

In case you want more verbosity, you add the `-v` option:

```
$ htminer_importer -v access_log.20061107
$ htminer_importer -vv access_log.20061107
$ htminer_importer -vvv access_log.20061107
```

The more *v*'s you add, the more verbosity you get.

The first time you launch the application, you want it to initialise the data source (create the tables and the indexes). You can do it by adding the `-i` option:

```
$ htminer_importer -vi access_log.20061107
```

Importer's configuration file

The importer configuration is an XML file which is made up of 5 sections:

1. input control

2. output control

3. exclusions (filtering)

4. mime types file (for IIS)

5. logger control

Input control

This section controls the configuration of the input module. In particular, it allows to specify the Apache log format, using regular expressions (PCRE syntax) and to associate blocks to fields that ht://Miner recognises.

By default, the importer file is configured to manage the standard ht://Miner's log format for Apache.

Output control

This section allows to control the output database for the importer. This database serves for temporary purposes. For the moment, the allowed type for the output data source is `postgresql`, as it is the only RDBMS currently supported.

You need to modify the credentials to connect to the database, such as host, port, database name, user, password and tablespace (for instance `htminer_oltp` or `htminer_warehouse`).

Exclusions

ht://Miner's importer can filter input data by applying regular expressions on the following fields:

1. content types (i.e. images, CSS files, etc.)

2. user agents (for instance internal spiders)

3. server names

MIME Types file

Unfortunately, IIS cannot store the *Content-Type* that servers return to the client. It is necessary that ht://Miner performs an automatic detection by mapping the resource extension to a specific content-type. For instance:

```
text/html    htm html shtml xht xhtml phtml pht php php3 php4 asp cfm
```

Logger

Currently it only supports the standard output channel.

Step three - IP Address resolution

Once the data is imported in the temporary table (*temporary.access_log*), ht://Miner needs to *resolve* the IP addresses into *host names*. The way ht://Miner performs this task is very efficient:

1. retrieves the list of IP addresses (performs a *SELECT DISTINCT* on the table to get just the IP addresses list without duplicates)

2. supports multi-threading: you can decide the number of threads that perform IP address resolution using the `gethostbyaddr()` system call

3. for each IP address that is resolved, performs an UPDATE of the host name in the database temporary table

4. supports a dual method for IP resolution:

    a. distributed: every UPDATE is embedded into a single atomic transaction (slower but crash-safe)

    b. standalone: every thread performs all its operations within a single transaction (therefore in case you run 10 threads there will be 10 open transaction); this approach is faster but it is not crash-safe (if the application crashes, the transaction is not committed and you do not perform any update to the database)

The application responsible for performing the IP addresses resolution is called *htminer_ipresolver*. For more information regarding its usage, you can type:

```
$ htminer_ipresolver —help
```

The application currently does not support any configuration file. Therefore you need to provide access credentials through command line:

```
$ htminer_ipresolver -H HOSTNAME -U USERNAME -D DATABASE -t THREADSNUMBER -v
```

Step four - Data processing (OLTP)

The data processing step of ht://Miner is responsible for organising Web usage data stored in the temporary table in *user's sessions*.

This process is achieved through:

1. *user's identification* (anonymous or authenticated)

2. *sessions' identification*: user's requests are first ordered by access time then grouped into logical sets

called *sessions*, where the time difference between a particular request and the following (if available) is less than *30 minutes* (this value can be changed at run-time)

3. *transactions' identification*: requests in a session can be grouped in smaller subset called *transactions*. Currently ht://Miner supports 3 algorithms for transactions discovery [gb2001]:

    a. *time-window length* (*TWL*): requests are grouped into transactions of length *t* (e.g. 15 seconds)

    b. *maximal forward reference* (*MFR*): grouping is based on the navigational path that is performed by the user. A user makes *forward references* and *backward references* during the surfing activity and the algorithm *automatically* attempts to detect these subsets. The algorithm is suitable for *tree-like* websites, but highly discouraged for *network-like* sites

    c. *reference length* (*RL*): requests are grouped according to the *reference length* parameter, which represents the inactivity time (e.g. 15 seconds). This technique allows the classification of requests into *navigational* or *content* ones. If a user spend more time than the reference length parameter on a page, this page is classified as *content* request and the previous requests within the transaction as *navigational* (or *auxiliary*) requests. This is also the current *default method* for transaction discovery in ht://Miner.

4. *spiders detection*: ht://Miner is able to detect spiders both *automatically* and *supervisedly*. The latter one is performed using *regular expressions* to match the *User Agent* string. The first one is partially based on the algorithm by Tan and Kumar [tan00modeling] and allows the classification of sessions in *spiders sessions* based on the characteristics of the requests

5. *localisation*: in case you enabled the GeoIP support the processing phase attempts for discover the location of an IP address (country and possibly region and city)

At the end of the process the available web usage data is organised in the first stage data source (*OLTP*): every information is stored in tables within the *oltp* schema of the PostgreSQL database.

The application responsible for performing the OLTP database loading operation is *htminer_oltp_loader*. For more information regarding its usage, you can type:

```
$ htminer_oltp_loader —help
```

The application by default uses the `oltp.xml` configuration file which sits in the *configuration* directory (by default `/usr/local/htminer/conf`).

A typical execution, which performs initialisation tasks as well (`-i`), is:

```
$ htminer_oltp_loader -vpi
```

OLTP loader configuration file

The OLTP loader file for configuration is an XML file. It allows the definition of several options, including input channel, output channel, session length, transaction discovery algorithm, etc.

Here follows a brief description of the main features.

Input channel definition

The *input* element is responsible for defining an input channel. An input channel is a data source that expects the following information:

1. data source type (currently *postgresql*)

2. host (*host name*, *host IP address*, *Unix socket file* - e.g. `/tmp`)

3. port

4. data source name (database name)

5. connection user

6. connection password

7. connection pool size (e.g. 10)

8. session length (in seconds, by default 1800 seconds - 30 minutes)

9. transaction discovery algorithm (by default *reference length* with 15 seconds inactivity time)

10. timeout for threads wait operations (in seconds): when waiting for connections from the pool, threads can hang for up to *timeout* seconds

11. date range:

    a. begin date: first date to be processed (specific date such as *2006-11-01*, *first*, *last*, *yesterday*, *today*)

    b. end date (as above): last date to be processed

12. requirements for the input channel: in case of IP resolution, accepts only those requests where the IP resolution has been performed

Output channel definition

The *input* element is responsible for defining the output channel. An output channel is a data source that expects the following information:

1. data source type (currently *postgresql*)

2. host (*host name*, *host IP address*, *Unix socket file* - e.g. `/tmp`)

3. port

4. data source name (database name)

5. connection user

6. connection password

7. tablespace to be used for tables creation

8. connection pool size (e.g. 20)

Spider control configuration

The *spidercontrol* element is responsible for configuring the spider detection process. It serves both the *automatic* process and the *supervised* one.

Automatic spider detection

The automatic spider detection is configured by setting the following attributes for the `spidercontrol` element:

- `robotstxt="`*`boolean value`*`"`: if a session contains a request to the `/robots.txt` file, the session is classified as *spider session*

- `headrequests="`*`boolean value`*`"`: if a session contains an HTTP HEAD request, it is classified as *spider session*

- `emptyreferers="`*`percentage`*`"`: percentage of allowed empty referers (if a session contains more than the specified percentage it is classified as *spider session*)

- `minimumlength="`*`integer value`*`"`: minimum sessions length (number of requests) for the application of the automatic spiders detection algorithm. Sessions with a length shorter than this value are not considered for the automatic detection algorithm.

A *boolean value* can be one of the following:

- *true*, *t*, *yes*, *y*

- *false*, *f*,*no*, *n*

A *percentage value* is an integer value from 0 to 100.

Supervised spider detection

Currently, the supervised detection of spiders is performed through *regular expressions* that are applied to the *user agent* string that the browser sends to the HTTP server through the `User-Agent` HTTP header.

You can specify regular expressions using the `regexp` element within the `useragents` element. See the default configuration file for examples.

Step five - First stage data warehouse

loading

Currently, the loading process of the first stage data warehousing is performed by a `Perl` script that you can find in the `contrib/scripts` directory. The script name is `htminer_warehouse_loader`.

For more information regarding its usage, you can type:

```
$ htminer_warehouse_loader —help
```

A very common way of invoking the script is:

```
$ htminer_warehouse_loader -a -v
```

Step six - Custom data warehouse loading (second stage)

The second stage data warehouse is for *custom* purposes. Theoretically there could be infinite applications of custom warehouses which are built using the first stage data warehouse information.

Currently, ht://Miner implements the creation of *anonymous day facts* but it is designed to easily plug different algorithms (for instance anonymous sessions facts, spider facts, entry points, crackers attempts, association rules, etc.).

Also the custom warehouse is designed to handle *hierarchies* through the *hierarchy manager* application.

Hierarchy manager

ht://Miner comes with an application that manages URLs classification in hierarchies of categories, allowing the creation of user-friendly reports and more powerful description of available data for business purposes.

Using an XML file and regular expressions that are applied to URLs, the hierarchy manager creates database tables (including the powerful warehouse *bridge hierarchy tables* for multi-level hierarchies) and manages them over time accounting data changes. This is an important feature for a data warehouse, because data changes over time and it is fundamental to keep historical data.

Note

Consider this example: the C category is under the T category. On the D day, the website structure is reorganised and the C category is moved under the Y category. Accesses to the C category before the D day must be counted under the T category, whereas from that day on (unless other changes occur) requests must be assigned to the Y category as well.

The `htminer_hierarchy_manager` is the application that is responsible for this process. For more information:

```
$ htminer_hierarchy_manager —help
```

The application by default uses the `hierarchy.xml` configuration file which sits in the *configuration* directory (by default `/usr/local/htminer/conf`). For more information, look at the default file.

Anonymous Day facts

An anonymous day fact is an aggregation of requests performed by an anonymous user (every information about the possible user is detached to the requests) once spiders requests have been filtered.

These facts are organised in day tables and monthly tables, with different levels of aggregation (*granularity*).

Data is also organised in categories, according to a specific *hierarchy* generated through the *hierarchy manager* application.

Step seven - PHP Interface

ht://Miner's report layer has a PHP interface for online reporting. The PHP interface is currently under development phase and is made up of a middleware library, a set of classes to be used by custom PHP applications.

Copyright

ht://Miner has been developed by the Comune di Prato. Its sources are property of the Comune di Prato and have been released under GNU/GPL license. Other authors have contributed to the project and the sources contain their names when this is applicable.

© Copyright 2003-2006 Comune di Prato, Prato, Tuscany, Italy

References

[expat] The Expat XML Parser. http://expat.sourceforge.net/

[gb2001] "Web usage mining and discovery of association rules from HTTP servers logs", Gabriele Bartolini, 2001. http://www.prato.linux.it/~gbartolini/en/view-a/2/pdf/wum.pdf

[geoipcapi] GeoIP C API. http://www.maxmind.com/app/c

[geolitecountry] GeoLite Country. http://www.maxmind.com/app/geoip_country

[geolitecity] GeoLite City. http://www.maxmind.com/app/geoip_city

[gettext] GNU gettext. http://www.gnu.org/software/gettext/

[openssl] The Open Source toolkit for SSL/TLS. http://www.openssl.org/

[pcre] Perl Compatibile Regular Expressions library. http://www.pcre.org/

[pgsql] PostgreSQL. http://www.postgresql.org/

[tan00modeling] "Modeling of Web Robot Navigational Patterns", P. Tan and V. Kumar, 2000. http://citeseer.ist.psu.edu/tan00modeling.html

[zlib] Zlib. http://www.zlib.net/