4.900

Data Mining Toolkit

by Rebecca Shalfield

Data Mining Toolkit

The contents of this manual describe the product, the Data Mining Toolkit version 1.66, and are believed correct at the time of going to press. They do not embody a commitment or imply any liability on the part of Logic Programming Associates (LPA), who may from time to time make changes to the specification of the product, in line with their policy of continual improvement. No part of this manual may be stored, reproduced or transmitted in any form, electronic or mechanical, for any purpose other than the licensee's personal use, without the prior written agreement of LPA.

The Data Mining Toolkit was written by Dr Philip Vasey.

Logic Programming Associates Ltd
Studio 30
The Royal Victoria Patriotic Building
Trinity Road
London SW18 3SX
England

phone:	**+44 (0) 20 8871 2016**
fax:	**+44 (0) 20 8874 0449**
web site:	**http://www.lpa.co.uk**

21 July, 2004

Contents

Introduction to the LPA Data Mining Toolkit

What is the LPA Data Mining Toolkit?

The LPA data mining toolkit is a collection of routines, supplied in the form of an API, which provide advanced data analysis functionality. It is loosely based on association rule mining or rule induction and tries to detect 'things' in terms of interestingness.

The toolkit contains no graphics routines, and is designed for embedding within other applications or within a dedicated GUI of your own design.

It accesses its data sources via the ODBC manager and assumes that you have correctly set up your data sources using the ODBC Administrator.

The various routines often return lists and structures which can be manipulated easily by the Prolog developer. This makes the toolkit an ideal basis for Prolog developers to build their own data mining applications.

By combining the data mining toolkit and the Intelligence Server, it is possible to present the data mining toolkit as a COM object for embedding within, say, a VB-oriented application. By combining the data mining toolkit, ProWeb and ProData, it is possible to develop a web-based data mining application.

What's Included in the Toolkit?

The Data-Mining toolkit includes:

1. An API - A collection of routines and associated documentation for building DM solutions
2. A documented source-code example which uses the API and a variety of dialogs created using the Dialog Editor to construct a simple DM application demo. You can compile and run this code in source mode. You can modify and expand the code.
3. A way to run that demo as if it were a stand-alone application. You can use this to illustrate the concept on any DB.
4. A documented web-based source-code example which uses the API and the ProWeb Toolkit.

Whilst one and two are aimed at application developers and researchers interested in building applications which have some DM component, three is primarily aimed at educators who are engaged in teaching the basics of DM.

What is Data Mining?

Data mining can be defined as:

> The non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

The word *process* is crucial here, since data mining consists of a number of stages, and requires a particularly high degree of interaction between the system and the analyst. The key stages of data mining as supported by the LPA Data Mining toolkit are:

- **Selection of Data Source**: The first stage is to select the appropriate data for analysis. The toolkit assumes that all joins and exclusions and views are actioned outside of the toolkit. The toolkit works off a single table.
- **Constructing a Target**: This is a simple formula which represents what you are interested in, and is used to focus the data mining investigation.
- **Discovering Patterns:** The main stage of the data mining process, and is where the patterns in the data are found. The first patterns to be discovered are simple atomic conditions of the form:

  A -> Q and

  B -> Q and

  C -> Q etc.

  We can then look to combine these atomic 'rules' and investigate the combinations i.e.:

  A & B -> Q

  A & C -> Q

  B & C -> Q etc.

  This process can go on until the rules get 'too' complicated or coverage diminishes to zero.

The Data Mining toolkit is goal-driven. The whole of this process is iterative, with the analyst returning to previous stages when not satisfied with the results obtained, and requires continuous interaction with and exploration of the intermediate results.

Terminology

Given a rule:

```
If A & B & C then D
```

we can identify two important measures:

1. Truth or Accuracy – How often is the rule correct?
2. Coverage – How often does the rule apply?

Truth indicates how often the conclusion is true given that the conditions are true. i.e. How does:

```
LHS -> RHS
```

Compare to:

```
LHS -> ~RHS?
```

Coverage indicates how much of the database the rule potentially explains. What proportion of all RHS does LHS -> RHS account for?

Base	= Total number of records in DB
Conditional	= Number of records where conditions (i.e. LHS) hold in DB
Hit	= Number of records where both conditions (i.e. LHS) and conclusion (i.e. RHS) hold
LHS	= Left-hand side of the rule
RHS	= Right-hand side of the rule
Target	= Number of records where conclusion (i.e. RHS) holds in DB
Target%	= **Target** / **Base**
Miss	= **Target** - **Hit**
True	= **Hit**
False	= **Conditional** - **True**
Other	= **Base** - (**Target** + **Conditional** – **Hit**)
True%	= (**True** / **Conditional**) * 100
Hit%	= (**Hit** / **Target**) * 100

Base%	= (**Conditional** / **Base**) * 100
Significance%	= ((**Hit%** - **Base%**) / **Base%**) * 100 OR = (**True%** - **Target%**) / **Target%**
AbsoluteSignificance%	= abs(**Significance%**)
Entropy	= aln(**True**/**Base**) - aln(**False**/**Base**) - aln(**Miss**/**Base**) - aln(**Other**/**Base**)

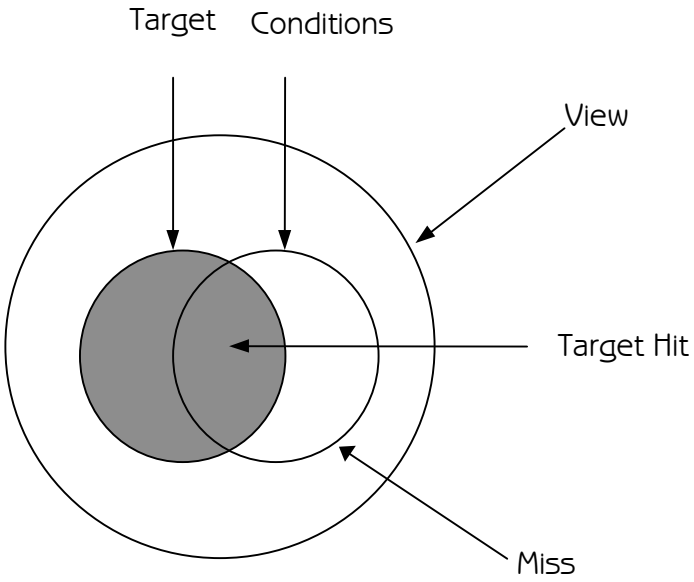
The principal purpose of data mining is to discover conditions which seem to bear an unnatural affinity with a nominated target condition. The data mining engine will search through all nominated columns and values looking for conditions which satisfy the threshold levels of 'interestingness'.

A high 'hit rate' is desirable; a high truth value is desirable and a reasonable coverage in both the base and target communities is required. Often there is a trade off between high truth (or accuracy) and high coverage (or target). We want rules (conditions) which explain as big a section of our target as possible without too many misses. We can always increase the truth by adding more conditions, but this typically reduces our coverage. By joining rules using OR, we can increase the coverage but at the risk of reducing accuracy.

The internal variables used include:

- view - the number of rows in the base view.
- target - the number of rows in which the target holds.
- condition - the number of rows in which the conditions hold.
- hit - the number of rows in which both the target and the conditions holds. Also known as a positive.
- miss - the number of rows in which the conditions hold, but the target is not true. Also known as a negative.

The following diagram shows graphically the meaning of the factors used in the display of influential conditions.


Target
Conditions
View
Target Hit
Miss

Two of the data mining toolkit's predicates (*dm_api_conditional_expression/2* and *dm_api_conditional_statement/4*) return the Prolog structure, *variables(...)*, which has 14 arguments; *variables/14* has the following form:

```
variables(
            BaseCount,
            TargetCount,
            ConditionalCount,
            HitCount,
            MissCount,
            TrueCount,
            FalseCount,
            OtherCount,
            True%,
            Hit%,
            Base%,
            Significance%,
            AbsSignificance%,
            Entropy
          ).
```

Significance is an indication of deviation from the norm. This can be either positive (i.e. over representation) or negative (i.e. under representation).

Entropy is a measure of interestingness. The default formula produces a relative value between the number of positives (condition and target holds) and the number of negatives (condition holds but the target does not).

Example

Let's suppose we are looking for people who bought product X and the following *variables/14* fact was returned to us:

```
variables(1000,300,280,62,238,62,218,482,22.1428571428571,20.666666666
6667,28,-26.1904761904762,26.1904761904762,-3.06764370101787)
```

The arguments are as follows:

```
variables(
            1st         1000,                   BaseCount
            2nd         300,                    TargetCount
            3rd         280,                    ConditionalCount
            4th         62                      HitCount
            5th         238,                    MissCount
            6th         62,                     TrueCount
            7th         218,                    FalseCount
            8th         482,                    OtherCount
            9th         22.1428571428571,       True%
            10th        20.6666666666667,       Hit%
            11th        28,                     Base%
            12th        -26.1904761904762,      Significance%
            13th        26.1904761904762,       AbsSignificance%
            14th        -3.06764370101787       Entropy
            ).
```

Argument	**Variable**
1st	BaseCount = 1000 i.e. We are looking at 1000 records.
2nd	TargetCount = 300 i.e. 300 of which bought X.
3rd	ConditionalCount = 280 e.g. We have found a condition, say, yellow hair, and there are 280 of them.
4th	HitCount = 62 i.e. 62 of the 280 also brought X.
5th	TargetCount - HitCount = MissCount 300 - 62 = 238 e.g. There's another 238 people who bought X but don't have, say, yellow hair.
6th	HitCount = TrueCount 62 = 62
7th	ConditionalCount - TrueCount = FalseCount 280 - 62 = 218 e.g. 218 people with say, yellow hair, did not buy X.
8th	BaseCount – (TargetCount + ConditionalCount – HitCount) = OtherCount 1000 – (300 + 280 – 62) = 482 e.g. There's 482 people without yellow hair or buying X.
9th	(TrueCount / ConditionalCount) * 100 = True% (i.e. Truth) (62 / 280) * 100 = 22.1428571428571% e.g. What's the percentage of people with yellow hair buying product X?
10th	(HitCount / TargetCount) * 100 = Hit% (i.e. Coverage) (62 / 300) * 100 = 20.6666666666667% e.g. What's the percentage of those buying product X and having, say, yellow hair?
11th	(ConditionalCount / BaseCount) * 100 = Base% (280 / 1000) * 100 = 28% e.g. What percentage of the 'world' have the condition, say, yellow hair?
12th	((Hit% - Base%) / Base%) * 100 = Significance% ((20.6666666666667 – 28) / 28) * 100 = -26.1904761904762%

	(Target / Base) * 100 = Target% (300 / 1000) * 100 = 30% e.g. Target% means what percentage of people buy product X? ((True% - Target%) / Target%) * 100 = Significance% ((22.1428571428571 – 30) / 30) * 100 = -26.1904761904762%
13th	abs(Significance%) = AbsSignificance% abs(-26.1904761904762) = 26.1904761904762%
14th	aln(TrueCount / BaseCount) – aln(FalseCount / BaseCount) - aln(MissCount / BaseCount) – aln(OtherCount / BaseCount) = Entropy aln(62 / 1000) – aln(218 / 1000) - aln(238 / 1000) – aln(482 / 1000) = -3.06764370101787

Installation

Introduction to ProData

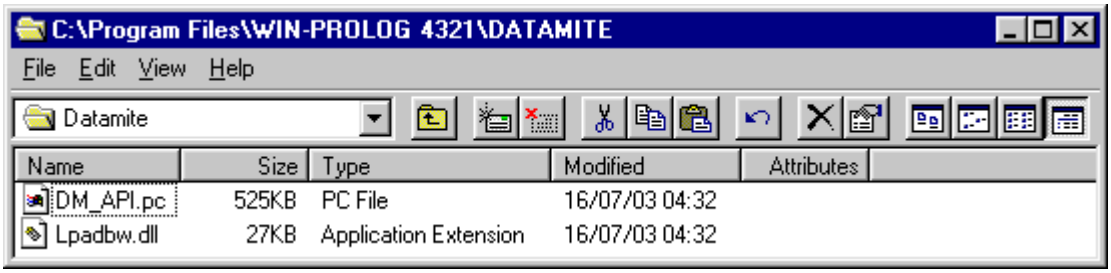
The data mining toolkit itself provides no predicates for general purpose programming access to ODBC and SQL, such as getting schema information or adding, deleting, updating records or tables.

It is assumed that the data mining toolkit will be used in conjunction with ProData. ProData provides general purpose programming access to ODBC and SQL. This allows you to interrogate data sources and their schemas, and build data dictionaries in Prolog.

You are referred to the "*ProData Interface*" manual for further information.

Installing the LPA Data Mining Toolkit

The data mining toolkit is installed along with ***WIN-PROLOG*** itself; just ensure that the 'Datamite API' component is enabled and selected. If installed correctly, you should have a Datamite folder, containing two files, within ***WIN-PROLOG***'s root folder.



Loading the LPA Data Mining Toolkit

To load the data mining toolkit, execute the following from the ***WIN-PROLOG*** command line:

```
?- ensure_loaded( prolog('datamite\dm_api') ). <enter>
```

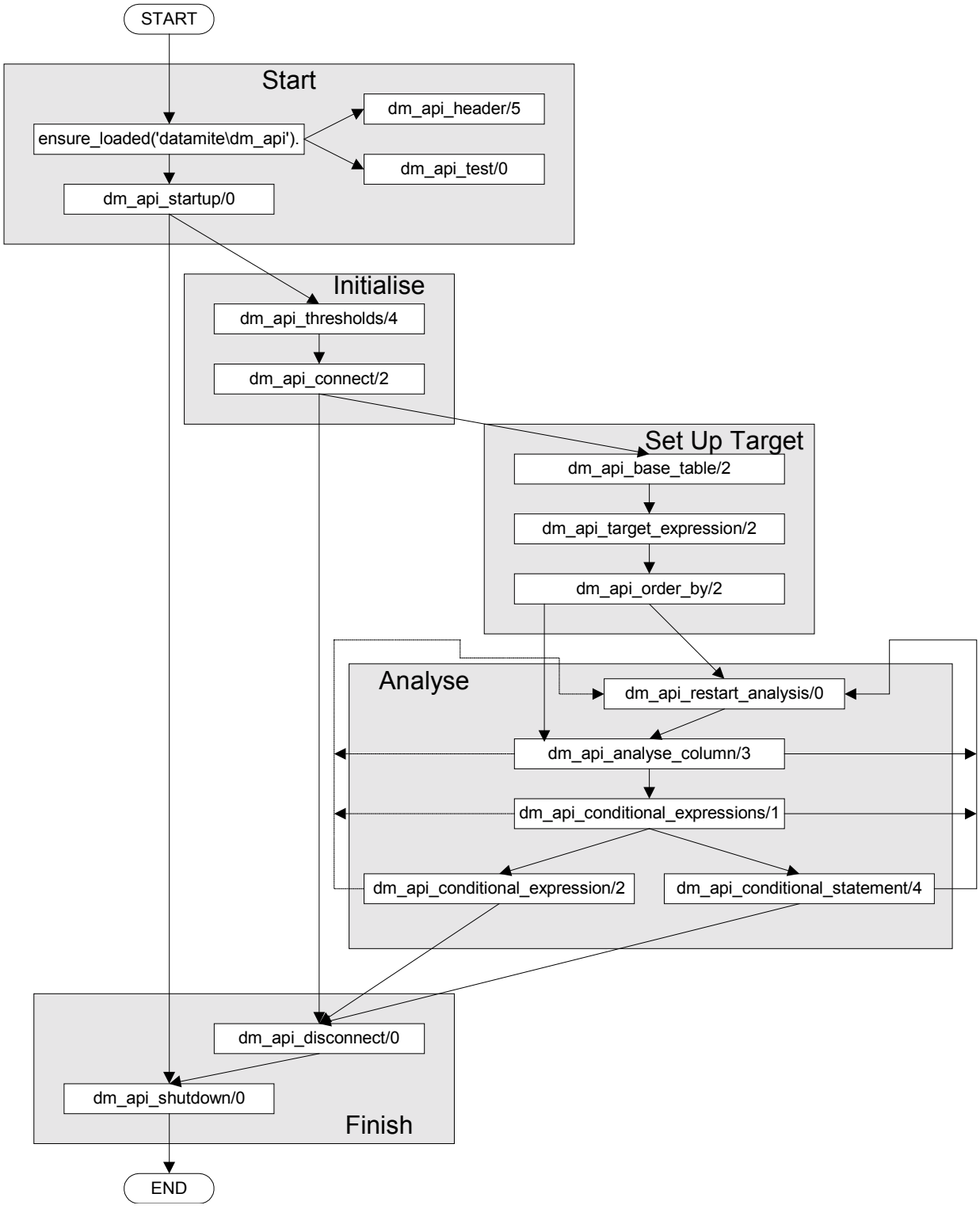
Getting the DLL in the Correct Place

The data mining toolkit requires the file, LPADBW.DLL, to be present in the ***WIN-PROLOG*** root directory.

When ProData is installed, LPADBW.DLL is automatically placed in the ***WIN-PROLOG*** root directory.

When installing the data mining toolkit without also installing ProData; LPADBW.DLL is placed in the Datamite directory within the ***WIN-PROLOG*** root directory; you just need to move LPADBW.DLL from the Datamite directory into the ***WIN-PROLOG*** root directory.

Predicate Flow Chart



Tutorial

First we need to load the Data Mining Toolkit into ***WIN-PROLOG***:

```
?- ensure_loaded( prolog('datamite\dm_api') ). <enter>
```

If you have the ProData Interface Toolkit, you might like to load this as well:

```
?- ensure_loaded( system(dblink) ). <enter>
```

We next need to start up the Data Mining Toolkit:

```
?- dm_api_startup. <enter>
```

If *dm_api_startup/0* fails, ensure you have the file, LPADBW.DLL, in ***WIN-PROLOG***'s root directory.

We next need to connect to an ODBC data source. Should you need to get a list of ODBC data sources currently available, Prodata's *db_show_schema/1* predicate will help:

```
?- db_show_schema( sources ). <enter>
```

The Data Mining Toolkit's *dm_api_connect/2* predicate allows us to connect to an ODBC data source:

```
?- dm_api_connect( `MyDataBase`, `ACCESS` ). <enter>
```

We next need to specify a table within our chosen ODBC data source to be the base table. Should you need to get a list of tables within a specified ODBC data source, Prodata's *db_show_schema/1* predicate will help:

```
?- db_show_schema( user ). <enter>
```

Let's assume we want to mine the table, MyTable, which contains the following data:

Name	Sex	Pay
Fred	Male	17500
Bill	Male	20000
Jack	Male	17500
Wendy	Female	22000
Martin	Male	35000
Joan	Female	30000
Richard	Male	22000

The Data Mining Toolkit's *dm_api_base_table/2* predicate allows us to specify the table we

wish to mine within our chosen ODBC data source:

```
?- dm_api_base_table( `MyTable`, BaseCount ). <enter>
BaseCount = 7
```

The value assigned to the variable, BaseCount, is the number of records in the base table, MyTable.

Next, we need to specify our target expression:

```
?- dm_api_target_expression( `Sex = 'Male'`, TargetCount ). <enter>
TargetCount = 5
```

The value assigned to the variable, TargetCount, is the number of records in the MyTable table where the given expression (i.e. `Sex = 'Male'`) holds.

What we have done so far with the Data Mining Toolkit is equivalent to the following ProData *db_sql_select/2* call:

```
?- db_sql_select( `SELECT COUNT(*) FROM MyTable WHERE Sex = 'Male'`,
[TargetCount] ). <enter>
TargetCount = 5
```

We can now analyse the Pay column to see how it relates to our target expression:

```
?- dm_api_analyse_column( `discrete`, `Pay`, NumberOfSolutions ).
<enter>
NumberOfSolutions = 4
```

The value assigned to the variable, NumberOfSolutions, equals the number of discrete values in the Pay column (i.e. 17500, 20000, 22000 and 35000) where the target expression (i.e. `Sex = 'Male'`) holds.

We could also analyse the Name column to see how it relates to the target expression:

```
?- dm_api_analyse_column( `discrete`, `Name`, NumberOfSolutions ).
<enter>
NumberOfSolutions = 5
```

The value assigned to the variable, NumberOfSolutions, equals the number of discrete values in the Name column (i.e. 'Fred', 'Bill', 'Jack', 'Martin' and 'Richard') where the target expression (i.e. `Sex = 'Male'`) holds.

For completeness, we could also try analysing the Sex column to see how it relates to the target expression:

```
?- dm_api_analyse_column( `discrete`, `Sex`, NumberOfSolutions ).
<enter>
NumberOfSolutions = 1
```

As you would expect, the value assigned to the variable, NumberOfSolutions, equals the number of discrete values in the Sex column (i.e. 'Male') where the target expression (i.e. `Sex = 'Male'`) holds.

The Data Mining Toolkit is now holding a number of solutions for us. Before we can get such solutions back, we need to tell it in what order we want such solutions to be

returned:

```
?- dm_api_order_by( `hit`, `asc` ). <enter>
yes
```

The above *dm_api_order_by/2* call states that we want the solutions returned in ascending hit count order; hit being one of 15 possible values.

Let's ask the Data Mining Toolkit what the output will be when Pay = 17500:

```
?- dm_api_conditional_statement( 1, `Pay = 17500`,
ConditionalStatement, Variables ). <enter>
ConditionalStatement = `Pay = 17500` ,
Variables = variables(7,5,2,2,3,2,0,2,100,40,28.5714285714286,40,40,-
2.53506300925521)
```

The returned *variables/14* fact tells us the following information:

- There are 7 records in the table, MyTable.
- There are 5 records where the target expression (i.e. `Sex = 'Male'`) holds.
- There are 2 records where the condition (i.e. `Pay = 17500`) holds.
- There are 2 records where both the target expression (i.e. `Sex = 'Male'`) and the condition (i.e. `Pay = 17500`) hold.
- There are 3 records where the target expression (i.e. `Sex = 'Male'`) holds but the condition (i.e. `Pay = 17500`) did not hold.
- There are 2 records where both the target expression (i.e. `Sex = 'Male'`) and the condition (i.e. `Pay = 17500`) hold.
- There are 0 records where the condition (i.e. `Pay = 17500`) holds but the target expression (i.e. `Sex = 'Male'`) did not hold.
- The other counts equal 2.
- The true percentage is 100%.
- The hit percentage is 40%.
- The base percentage is 28.6%.
- 40 is the significance.
- 40 is the absolute significance.
- -2.53506300925521 is the entropy.

We next need to disconnect from the ODBC data source:

```
?- dm_api_disconnect. <enter>
```

We finally need to shut down the Data Mining Toolkit:

```
?- dm_api_shutdown. <enter>
```

The Predicates

dm_api_analyse_column/3

Predicate dm_api_analyse_column(+Mode, +ColumnName, -Solutions)

Description Analyse a particular column in the base (target) table (set using *dm_api_base_table/2*) with respect to the target condition (set using *dm_api_target_expression/2*).

Arguments

+Mode	STRING	The mode of analysis: `discrete` or `DISCRETE` - Find individual column values that are significant `continuous` or `CONTINUOUS`- Find sub-ranges of column values that are significant
+ColumnName	STRING	The name of the column in the target table to be analysed
-Solutions	INTEGER	The total number of individual values (discrete analysis) or sub-range of values (continuous analysis) that were found to be significant

Examples

```
?- dm_api_startup,
dm_api_connect(`StatLog`,`ACCESS`),
dm_api_base_table(`GermanCredit`,_),
dm_api_target_expression(`"LoanApproved"=2 AND
"Foreign"='Yes'`,_),
dm_api_analyse_column(`discrete`,
`PurposeOfLoan`,Solutions). <enter>
Solutions = 2

?-dm_api_analyse_column(`discrete`,`Housing`,
Solutions). <enter>
Solutions = 3

?-dm_api_analyse_column(`continuous`, `Duration`,
Solutions ). <enter>
Solutions = 4
```

Comments Use a binary-chop algorithm to determine the sub-ranges of a continuous column.

Binary chop algorithm to determine the sub-ranges of a floating-point or timestamp column which meet the thresholds.

Stitch together contiguous sub-ranges providing they result in a NEW candidate condition! N.B. This operation may result in a new duplicate!

Remove redundant sub-ranges which are subsumed by more specific sub-ranges.

Sequence The predicates, *dm_api_base_table/2* and *dm_api_target_expression/2*, must be called before this one.

Trouble Shooting If the number of solutions returned is zero, ensure you have declared a target expression (using *dm_api_target_expression/2*) and/or revise your threshold settings with *dm_api_thresholds/4*.

You will get a type error 23 if the column being analysed does not support continuous mode. Try placing the *dm_api_analyse_column/3* call inside a catch:

```
?- ( catch( Error, dm_api_analyse_column( `continuous`,
`ProductName`, Solutions ) ),
     Error = 0,
     !
   ; dm_api_analyse_column( `discrete`,   `ProductName`,
Solutions )
   ). <enter>
```

dm_api_base_table/2

Predicate dm_api_base_table(+TableName, -BaseCount)

Description Declare which table in the connected database will be mined

Arguments

+TableName	STRING	The name of the table which is to be mined.
-BaseCount	INTEGER	The number of records (rows) in the table

Example

```
?-dm_api_startup, dm_api_connect(`StatLog`,`ACCESS`),
dm_api_base_table(`GermanCredit`,BaseCount). <enter>
BaseCount = 1000
?-listing(data_dm_api_global). <enter>
% data_dm_api_global/3
data_dm_api_global(told,dsn,'StatLog').
data_dm_api_global(count,base,1000).
data_dm_api_global(told,base_table,`GermanCredit`).
Yes
```

Sequence The predicates, *dm_api_startup/0* and *dm_api_connect/2*, must be called before this one.

Trouble Shooting The first argument, TableName, must be an LPA string and NOT an atom.

dm_api_conditional_expression/2

Predicate dm_api_conditional_expression(-ConditionalExpression,-Variables)

Description Return, through backtracking, each "atomic level" SQL conditional expression (e.g. PurposeOfLoan = 'New Car') that has been found to be 'interesting' following the analysis (using *dm_api_analyse_column/3*) of one or more columns. The order in which they are returned is determined by *dm_api_order_by/2*.

Arguments

-*ConditionalExpression*	STRING	An SQL statement, representing a significant "atomic level" condition, of the form: "ColumnName" = Value or "ColumnName" BETWEEN LowerValue AND UpperValue
-*Variables*	STRUCTURE	A Prolog structure of the form: **variables(...)** containing the variables.

Example

```
?-dm_api_startup, dm_api_connect(`StatLog`,`Access`),
dm_api_base_table(`GermanCredit`,_),
dm_api_target_expression(`"LoanApproved" = 2`,_),
dm_api_order_by( `entropy`, `asc` ),
dm_api_analyse_column(`discrete`,`PurposeOfLoan`,_),
dm_api_conditional_expression( Cond, Var ). <enter>

Cond = `"PurposeOfLoan" = 'Television'` ,
Var = variables( 1000, 300, 280, 62, 238, 62, 218,
482, 22.1428571428571, 20.6666666666667, 28,
-26.1904761904762, 26.1904761904762, -
3.06764370101787 ) ;

Cond = `"PurposeOfLoan" = 'New Car'` ,
Var = variables( 1000, 300, 234, 89, 211, 89, 145,
555, 38.034188034188, 29.6666666666667, 23.4,
26.7806267806268, 26.7806267806268,
-3.03981225377716) ;

no

findall( (Cond,Var),
         dm_api_conditional_expression(Cond,Var),
         Solutions
       ).
```

Sequence The predicate, *dm_api_analyse_column/3*, must be called one or more times before this predicate.

dm_api_conditional_expressions/1

Predicate dm_api_conditional_expressions(-Solutions)

Description Return the number of separate conditional expressions that were found to be significant during the analysis of one or more columns. Knowing the number of separate conditional expressions can help when ascertaining the integer to enter as the first argument in a *dm_api_conditional_statement/4* call.

Argument

-Solutions	INTEGER	The number of significant conditional expressions

Example

```
?-dm_api_startup,
dm_api_connect(`StatLog`,`ACCESS`),
dm_api_base_table(`GermanCredit`,_),
dm_api_target_expression(`"LoanApproved"=2`,_),
dm_api_analyse_column(`discrete`,`PurposeOfLoan`,_),
dm_api_conditional_expressions(Solutions). <enter>
Solutions = 2

?- dm_api_conditional_expressions( NoOfCondExp ),
dm_api_conditional_statement( NoOfCondExp,
`"PurposeOfLoan" = 'Television'`, Cond, Var ).
<enter>
```

Sequence The predicate, *dm_api_analyse_column/3*, must be called before this one.

dm_api_conditional_statement/4

Predicate dm_api_conditional_statement(+TopSolutions, +Mask, -ConditionalStatement, -Variables)

Description Return, through backtracking, each significant SQL statement that matches the mask supplied. The order in which they are returned is determined by *dm_api_order_by/2*.

Arguments

+TopSolutions	INTEGER > 0	The number of conditional expressions, in the order stated, that should be considered when forming the conditional statements
+Mask	STRING	A mask using the wildcards ? (replace with a single conditional expression) and * (replace with any combination of conditional expressions using the AND operator) which defines the form of the conditional statement. A possible mask might be: `?`, `*`, `"CreditHistory" <> 'Critical' AND ?` or `"PurposeOfLoan" = 'New Car' AND *`.
-ConditionalStatement	STRING	An SQL statement matching the mask which is deemed to be significant
-Variables	STRUCTURE	A Prolog structure of the form: **variables(...)** containing the variables.

Examples

```
?-dm_api_startup, dm_api_connect(`StatLog`,`Access`),
dm_api_base_table(`GermanCredit`,_),
dm_api_target_expression(`"LoanApproved" = 2`,_),
dm_api_order_by( `entropy`, `asc` ),
dm_api_analyse_column( `discrete`, `PurposeOfLoan`, _
), dm_api_conditional_expressions( NoOfCondExp ),
dm_api_conditional_statement( NoOfCondExp,
`"PurposeOfLoan" = 'Television'`, Cond, Var ).
<enter>

Cond = `"PurposeOfLoan" = 'Television'` ,

Var = variables( 1000, 300, 280, 62, 238, 62, 218,
482, 22.1428571428571, 20.6666666666667, 28,
-26.1904761904762, 26.1904761904762,
-3.06764370101787 )

?-dm_api_conditional_statement( 8, `"CreditHistory"
<> 'Critical' AND ?`, Cond, Vars ). <enter>
Cond = `"CreditHistory" <> 'Critical' AND "Duration"
BETWEEN 18 AND 21` ,
Vars = variables( 1000, 296, 105, 39, 257, 39, 66,
638, 37.14, 13.17, 10.5, 25.48, 25.48, -3.21 ) ;
Cond = `"CreditHistory" <> 'Critical' AND "Duration"
= 12` ,
Vars = variables( 1000, 296, 126, 42, 254, 42, 84,
620, 33.33, 14.18, 12.6, 12.61, 12.61, -3.19 ) ;
Cond = `"CreditHistory" <> 'Critical' AND
"PurposeOfLoan" = 'Furniture'` ,
Vars = variables( 1000, 296, 131, 47, 249, 47, 84,
620, 35.87, 15.87, 13.1, 21.20, 21.20, -3.18 ) ;
Cond = `"CreditHistory" <> 'Critical' AND "Housing" =
'Rent'` ,
Vars = variables( 1000, 296, 142, 60, 236, 60, 82,
622, 42.25, 20.27, 14.2, 42.74, 42.74, -3.15 ) ;
Cond = `"CreditHistory" <> 'Critical' AND
"PurposeOfLoan" = 'New Car'` ,
Vars = variables( 1000, 296, 156, 69, 227, 69, 87,
617, 44.23, 23.31, 15.6, 49.42, 49.42, -3.12 ) ;
Cond = `"CreditHistory" <> 'Critical' AND "Age"
BETWEEN 19 AND 26` ,
Vars = variables( 1000, 296, 201, 82, 214, 82, 119,
585, 40.79, 27.70, 20.1, 37.82, 37.82, -3.07 ) ;
```

Given the following `*`-masked program:

```
star_test :-
   dm_api_startup,
   dm_api_connect( `statlog`, `ACCESS` ),
   dm_api_base_table( `GermanCredit`, _ ),
   dm_api_target_expression( `"LoanApproved" = 2`, _ ),
   dm_api_thresholds( 1, 1, 1, 1 ),
   dm_api_order_by( `base`, `asc` ),
   dm_api_analyse_column( `discrete`, `PurposeOfLoan`, _ ),
   dm_api_analyse_column( `discrete`, `StatusSex`,     _ ),
   dm_api_conditional_expressions( NoOfCondExps ),
   forall( data_dm_api_global( conditional_expression(_), _,
CondExp-_ ),
           ( write( CondExp ),
             nl
           )
         ),
   nl,
   dm_api_conditional_statement( NoOfCondExps, `*`, CondState,
_ ),
   write( CondState ),
   nl,
   fail.

star_test :-
   dm_api_shutdown,
   dm_api_disconnect.
```

The following conditional expressions and conditional statements (shown in unsorted order due to order by `base` and `asc`) are generated:

```
?- star_test. <enter>
"PurposeOfLoan" = 'Appliance'
"PurposeOfLoan" = 'Business'
"PurposeOfLoan" = 'Education'
"PurposeOfLoan" = 'Furniture'
"PurposeOfLoan" = 'New Car'
"PurposeOfLoan" = 'Other'
"PurposeOfLoan" = 'Repairs'
"PurposeOfLoan" = 'Television'
"PurposeOfLoan" = 'Used Car'
"StatusSex"     = 'Divorced Male'
"StatusSex"     = 'Married Male'
"StatusSex"     = 'Married/Divorced Female'
"StatusSex"     = 'Single Male'

"PurposeOfLoan" = 'Business' AND  "StatusSex" = 'Divorced Male'
"PurposeOfLoan" = 'Furniture' AND "StatusSex" = 'Divorced Male'
"PurposeOfLoan" = 'New Car' AND   "StatusSex" = 'Divorced Male'

"PurposeOfLoan" = 'New Car' AND    "StatusSex" = 'Married Male'
"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Married Male'

"PurposeOfLoan" = 'Business' AND
                        "StatusSex" = 'Married/Divorced Female'
"PurposeOfLoan" = 'Education' AND
                        "StatusSex" = 'Married/Divorced Female'
"PurposeOfLoan" = 'Furniture' AND
                        "StatusSex" = 'Married/Divorced Female'
"PurposeOfLoan" = 'New Car' AND
                        "StatusSex" = 'Married/Divorced Female'
"PurposeOfLoan" = 'Television' AND
                        "StatusSex" = 'Married/Divorced Female'
"PurposeOfLoan" = 'Used Car' AND
                        "StatusSex" = 'Married/Divorced Female'

"PurposeOfLoan" = 'Business' AND    "StatusSex" = 'Single Male'
"PurposeOfLoan" = 'Education' AND   "StatusSex" = 'Single Male'
"PurposeOfLoan" = 'Furniture' AND   "StatusSex" = 'Single Male'
"PurposeOfLoan" = 'New Car' AND     "StatusSex" = 'Single Male'
"PurposeOfLoan" = 'Repairs' AND     "StatusSex" = 'Single Male'
"PurposeOfLoan" = 'Television' AND  "StatusSex" = 'Single Male'
"PurposeOfLoan" = 'Used Car' AND    "StatusSex" = 'Single Male'
no
```

Each combination of one "PurposeOfLoan" = <value> and one "StatusSex" = <value> is being considered by the Datamining toolkit but some fail to meet the thresholds and are internally rejected. The following 'trace' gives you a feel for what is happening inside *dm_api_conditional_statement/4* when given a `*` as the mask; the failed combinations are shown in bold:

```
TRYING :"PurposeOfLoan" = 'Appliance'  AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Business'   AND "StatusSex" = 'Divorced Male'
SUCCESS:"PurposeOfLoan" = 'Business'   AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Education'  AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Furniture'  AND "StatusSex" = 'Divorced Male'
SUCCESS:"PurposeOfLoan" = 'Furniture'  AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'New Car'    AND "StatusSex" = 'Divorced Male'
SUCCESS:"PurposeOfLoan" = 'New Car'    AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Other'      AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Repairs'    AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Used Car'   AND "StatusSex" = 'Divorced Male'
TRYING :"PurposeOfLoan" = 'Appliance'  AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Business'   AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Education'  AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Furniture'  AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'New Car'    AND "StatusSex" = 'Married Male'
SUCCESS:"PurposeOfLoan" = 'New Car'    AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Other'      AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Repairs'    AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Married Male'
SUCCESS:"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Used Car'  AND "StatusSex" = 'Married Male'
TRYING :"PurposeOfLoan" = 'Appliance' AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Business'  AND "StatusSex" = 'Married/Divorced Female'
SUCCESS:"PurposeOfLoan" = 'Business'  AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Education' AND "StatusSex" = 'Married/Divorced Female'
SUCCESS:"PurposeOfLoan" = 'Education' AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Furniture' AND "StatusSex" = 'Married/Divorced Female'
SUCCESS:"PurposeOfLoan" = 'Furniture' AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'New Car'   AND "StatusSex" = 'Married/Divorced Female'
SUCCESS:"PurposeOfLoan" = 'New Car'   AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Other'     AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Repairs'   AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Married/Divorced
Female'
SUCCESS:"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Married/Divorced
Female'
TRYING :"PurposeOfLoan" = 'Used Car'  AND "StatusSex" = 'Married/Divorced Female'
SUCCESS:"PurposeOfLoan" = 'Used Car'  AND "StatusSex" = 'Married/Divorced Female'
TRYING :"PurposeOfLoan" = 'Appliance'  AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Business'   AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'Business'   AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Education'  AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'Education'  AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Furniture'  AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'Furniture'  AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'New Car'    AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'New Car'    AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Other'      AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Repairs'    AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'Repairs'    AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'Television' AND "StatusSex" = 'Single Male'
TRYING :"PurposeOfLoan" = 'Used Car'   AND "StatusSex" = 'Single Male'
SUCCESS:"PurposeOfLoan" = 'Used Car'   AND "StatusSex" = 'Single Male'
```

Sequence The predicates, *dm_api_order_by/2* and *dm_api_analyse_columns/3*, must be called before this one.

Trouble Shooting This predicate will fail if no conditional expressions exist in memory.

dm_api_connect/2

Predicate dm_api_connect(+DSN, +Driver)

Description Connect to a named data source using the named driver

Arguments

+DSN	STRING	The name of the ODBC data source.
+Driver	STRING	The name of the ODBC database driver: ANSI : American National Standards Institute ACCESS : Microsoft Access(r) SQL SERVER : Microsoft SQL Server(r) ORACLE : Oracle(r) DB2 : IBM DB2(r)

Examples

```
?-dm_api_startup, dm_api_connect(`StatLog`,`ACCESS`),
dm_api_disconnect, dm_api_shutdown. <enter>
yes
```

```
?-dm_api_connect(`StatLog`,`ACCESS`). <enter>
yes
?-listing(data_dm_api_global). <enter>
data_dm_api_global(told,dsn,'StatLog').
?-dm_api_connect(`Northwind`,`ACCESS`). <enter>
yes
?-listing(data_sql_dsn_connection). <enter>
% data_sql_dsn_connection/3
data_sql_dsn_connection('Northwind',ms_jet,`2`).
data_sql_dsn_connection('StatLog',ms_jet,`1`).
Yes
```

```
?-dm_api_connect(`Test;PWD=rhubarb`,`ACCESS`).
<enter>
yes
```

Sequence The predicate, *dm_api_startup/0,* must be called before this one.

Trouble Shooting The two arguments must be entered as LPA string data types and NOT atoms.

dm_api_disconnect/0

Predicate dm_api_disconnect

Description Disconnect from the last data source connected to via a call to *dm_api_connect/2*

Example

```
?-dm_api_startup,
dm_api_connect(`StatLog`,`ACCESS`),
dm_api_disconnect, dm_api_shutdown. <enter>
yes
```

Sequence The predicate, *dm_api_connect/2*, must be called before this one.

Trouble Shooting If you nest two or more *dm_api_connect/2* ... *dm_api_disconnect/0* calls, you may need to re-execute the outer *dm_api_connect/2* call(s) before *dm_api_disconnect/0* will work.

dm_api_header/5

Predicate dm_api_header(-Application, -Copyright, -Author, -VersionNumber, -VersionDate)

Purpose Returns the application's name, copyright information, author's name, version number and version's date.

Arguments

-Application	STRING	The name of the application.
-Copyright	STRING	The copyright notice.
-Author	STRING	The name of the author(s).
-VersionNumber	STRING	The current version number.
-VersionDate	STRING	The date of the current version.

Example

```
?-dm_api_header( Application, Copyright, Author, Version, Date ).
<enter>
Application = `DataMite(TM) Application Programmer's Interface` ,
Copyright = `Copyright Logic Programming Associates 1995-2000` ,
Author = `Dr Phil Vasey` ,
Version = `1.66` ,
Date = `15 JUN 2000`
```

Sequence This predicate can be called at any time.

dm_api_order_by/2

Predicate dm_api_order_by(+Variable, +AscDesc)

Description Declare the order in which conditional expressions, etc. will be returned

Arguments

+*Variable*	STRING	The name of a variable in the set { `base`, `target`, `conditional`, `hit`, `miss`, `true`, `false`, `other`, `true%`, `hit%`, `base%`, `significance%`, `absolute_significance%`, `|significance%|`, `entropy` }
+*AscDesc*	STRING	Either `ASC` for an ascending order or `DESC` for a descending order

Examples

```
?-dm_api_startup, dm_api_connect(`StatLog`,`Access`),
dm_api_base_table(`GermanCredit`,_),
dm_api_target_expression(`"LoanApproved" = 2`,_),
dm_api_order_by( `entropy`, `asc` ),
dm_api_analyse_column(`discrete`,`PurposeOfLoan`,_),
dm_api_conditional_expression( Cond, Var ). <enter>

Cond = `"PurposeOfLoan" = 'Television'` ,
Var = variables( 1000, 300, 280, 62, 238, 62, 218,
482, 22.1428571428571, 20.6666666666667, 28,
-26.1904761904762, 26.1904761904762,
-3.06764370101787 ) ;

Cond = `"PurposeOfLoan" = 'New Car'` ,
Var = variables( 1000, 300, 234, 89, 211, 89, 145,
555, 38.034188034188, 29.6666666666667, 23.4,
26.7806267806268, 26.7806267806268,
-3.03981225377716 ) ;

no

?-dm_api_order_by(`ENTROPY`,`DESC`). <enter>
yes
```

Sequence This predicate can be called at any time but must be executed prior to calling *dm_api_conditional_expression/2* or *dm_api_conditional_statement/4*.

dm_api_restart_analysis/0

Predicate dm_api_restart_analysis

Description Restart the mining process for the target condition in the target table of the connected database. *dm_api_restart_analysis/0* actually retracts all the *data_dm_api_global(conditional_expression(_),_,_)* facts, such as those asserted by *dm_api_analyse_column/3*.

Example

```
?-dm_api_analyse_column( `discrete`, `PurposeOfLoan`,
_ ), dm_api_restart_analysis. <enter>
yes
```

Sequence The predicate, *dm_api_analyse_column/3*, must be called before this one.

dm_api_shutdown/0

Predicate dm_api_shutdown

Description Shutdown procedure which terminates the data mining toolkit

Side-Effect ODBC Closes the ODBC library

Example

```
?-dm_api_startup, dm_api_shutdown. <enter>
yes
```

Sequence The predicate, *dm_api_startup/0*, must be called before this one, otherwise an error will be generated. Placing the call to *dm_api_shutdown/0* inside a *catch/2* may solve your problem:

```
?- catch( Error, dm_api_shutdown ). <enter>
```

dm_api_startup/0

Predicate dm_api_startup

Description Startup procedure which initialises the data mining toolkit

Side-Effects

Prolog	Creates all the dynamic predicates used to store information.
ODBC	Opens the ODBC library

Example

```
?-dm_api_startup, dm_api_shutdown. <enter>
yes
```

Sequence This is the first Datamining toolkit predicate to be called.

dm_api_target_expression/2

Predicate dm_api_target_expression(+TargetExpression, -TargetCount)

Description Declare the SQL target expression

Arguments

+TargetExpression	STRING	The SQL expression which forms the target of the mining process.
-TargetCount	INTEGER	The number of rows in the base table where the target expression applies

Examples

```
?- dm_api_startup,
dm_api_connect(`StatLog`,`Access`),
dm_api_base_table(`GermanCredit`, BaseCount ),
dm_api_target_expression( `"LoanApproved" = 2`,
TargetCount ). <enter>
BaseCount = 1000,
TargetCount = 300
```

```
?- dm_api_startup,
dm_api_connect(`StatLog`,`Access`),
dm_api_base_table(`GermanCredit`,_),
dm_api_target_expression( `"LoanApproved" = 2 AND
"Foreign" = 'Yes'`, TargetCount ). <enter>
TargetCount = 296
```

Comparison To ProData This is equivalent to the ProData predicate:

```
?- db_sql_select( `SELECT COUNT(*) FROM GermanCredit
WHERE LoanApproved = 2`, [TargetCount] ). <enter>
TargetCount = 300
```

Sequence The predicate, *dm_api_base_table/2*, must be called before this one.

dm_api_thresholds/4

Predicate dm_api_thresholds(+SignificanceThreshold, +BaseThreshold, +HitThreshold, +TrueThreshold)

Description Set the minimum threshold percentages that determine whether or not something is deemed to be 'interesting'

Arguments

+*SignificanceThreshold*	NUMBER [0-100]	The minimum threshold for the AbsoluteSignificance% variable (the variables fact's 13th argument)
+*BaseThreshold*	NUMBER [0-100]	The minimum threshold for the Base% variable (the variables fact's 11th argument)
+*HitThreshold*	NUMBER [0-100]	The minimum threshold for the Hit% variable (the variables fact's 10th argument)
+*TrueThreshold*	NUMBER [0-100]	The minimum threshold for the True% variable (the variables fact's 9th argument)

Example

```
?-dm_api_thresholds(25,10,5,75). <enter>
yes
?-listing(data_dm_api_global). <enter>
% data_dm_api_global/3
data_dm_api_global( threshold,
'absolute_significance%', 25 ).
data_dm_api_global( threshold, 'base%', 10 ).
data_dm_api_global( threshold, 'hit%', 5 ).
data_dm_api_global( threshold, 'true%', 75 ).
```

Sequence This predicate can be called at any time.

Additional Predicates

This section gives definitions for a number of additional predicates you may find useful.

dm_api_analyse_columns/1

This *dm_api_analyse_columns(-AllSolutions)* predicate, defined as follows, allows you to analyse all the columns in the base table and get back a count of all the conditional expressions found (just like *dm_api_conditional_expressions/1*).

```
dm_api_analyse_columns( AllSolutions ) :-
   data_dm_api_global( told, base_table, BaseTable ),
   db_get_schema( columns( BaseTable ), ColumnNames ),
   forall( member( ColumnName, ColumnNames ),
           ( atom_string( ColumnName, ColumnNameAsString ),
             ( catch( Error, dm_api_analyse_column( `continuous`,
                                        ColumnNameAsString, _ ) ),
               Error = 0,
               !
             ; dm_api_analyse_column( `discrete`, ColumnNameAsString,
                                                                _ )
             )
           )
         ),
   dm_api_conditional_expressions( AllSolutions ).
```

This predicate will only work if ProData is loaded and you have connected to the data source with db_connect/2. *db_get_schema/2* is a ProData predicate which returns a list of the columns within a given table.

dm_api_base_table/1

This *dm_api_base_table(-BaseTable)* predicate, defined as follows, allows you to get the name of the base table.

```
dm_api_base_table( BaseTable ) :-
   one( data_dm_api_global( told, base_table, BaseTable ) ).

?- dm_api_base_table( BaseTable ). <enter>
BaseTable = `products`
```

dm_api_conditional_expressions/2

This *dm_api_conditional_expressions(-NumberOfSolutions, -ConditionalExpressions)* predicate, defined as follows, can be called after one or more *dm_api_analyse_column/3* calls to pick up all the solutions in a single list:

```
dm_api_conditional_expressions( NumberOfSolutions,
ConditionalExpressions ) :-
   dm_api_conditional_expressions( NumberOfSolutions ),
   findall( ConditionalExpression,
            data_dm_api_global( conditional_expression(_), _,
ConditionalExpression - _ ),
            ConditionalExpressions
          ).

?- dm_api_conditional_expressions( NumberOfSolutions,
ConditionalExpressions ). <enter>
NumberOfSolutions = 4 ,
ConditionalExpressions = [`"ProductID" BETWEEN 1 AND 39`,`"ProductID"
BETWEEN 1 AND 20`,`"ProductID" BETWEEN 24 AND 39`,`"ProductID" BETWEEN
28 AND 39`]
```

---

dm_api_connected/1

This *dm_api_connected(-DSNs)* predicate, defined as follows, allows you to ascertain which data sources you are currently connected to and in what order.

```
dm_api_connected( DSNs ) :-
   findall( DSNAsString,
            ( data_sql_dsn_connection( DSN, _, _ ),
              atom_string( DSN, DSNAsString )
            ),
            ReversedDSNs
          ),
   reverse( ReversedDSNs, DSNs ).

?- dm_api_connect( `Statlog`, `ACCESS` ). <enter>
yes

?- dm_api_connected( DSNs ). <enter>
DSNs = [`Statlog`]

?- dm_api_connect( `nwind`, `ACCESS` ). <enter>
yes

?- dm_api_connected( DSNs ). <enter>
DSNs = [`Statlog`,`nwind`]
```

---

dm_api_ordered_by/2

This *dm_api_ordered_by(-Variable, -AscDesc)* predicate, defined as follows, allows you to ascertain the order in which condition expressions, etc. will be returned.

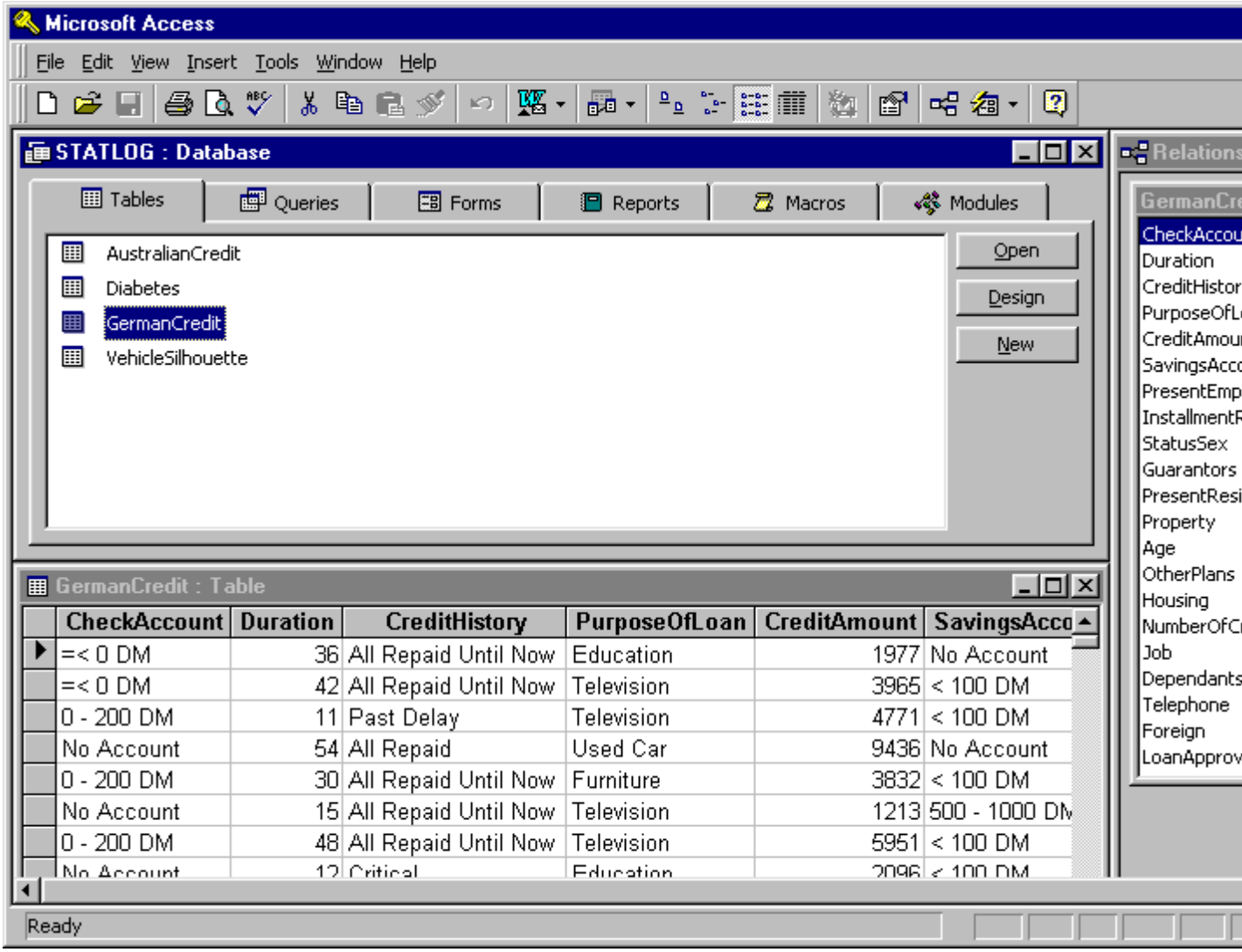
```
dm_api_ordered_by( Variable, AscDesc ) :-
   data_dm_api_global( told, order_by_position, OrderByPosition ),
   OrderByPositionMinusOne is OrderByPosition - 1,
   member( Variable, [ `base`,
                       `target`,
                       `conditional`,
                       `hit`,
                       `miss`,
                       `true`,
                       `false`,
                       `other`,
                       `true%`,
                       `hit%`,
                       `base%`,
                       `significance%`,
                       `absolute_significance%`,
                       `entropy`
                     ], OrderByPositionMinusOne ),
   data_dm_api_global( told, order_by_direction, OrderByDirection ),
   (  OrderByDirection = 1
   -> AscDesc = `asc`
   ;  AscDesc = `desc`
   ),
   !.

?- dm_api_order_by(`entropy`,`desc`). <enter>
yes

?- dm_api_ordered_by( Variable, AscDesc ). <enter>
Variable = `entropy` ,
AscDesc = `desc`
```

Built-In Test Predicate

The DM_API.PC file has a built-in test predicate, *dm_api_test/0*. This predicate performs a test of the data mining toolkit using the StatLog MS Access(r) database.



Source Code

The source code of *dm_api_test/0* is as follows:

```
dm_api_test :-
        dm_api_test( `StatLog`, `ACCESS` ).

dm_api_test( DSN, Driver ) :-
        dm_api_call_startup,
        dm_api_call_connect( DSN, Driver ),
        dm_api_call_base_table( `GermanCredit` ),
        dm_api_call_target_expression( `"LoanApproved" = 2 AND
"Foreign" = 'Yes'` ),
        dm_api_call_thresholds( 5, 10, 10, 25 ),
        dm_api_call_restart_analysis,
        dm_api_call_analyse_column( `DISCRETE`,   `PurposeOfLoan` ),
        dm_api_call_analyse_column( `DISCRETE`,   `StatusSex`     ),
        dm_api_call_analyse_column( `DISCRETE`,   `Housing`       ),
        dm_api_call_analyse_column( `CONTINUOUS`, `Duration`      ),
        dm_api_call_analyse_column( `CONTINUOUS`, `Age`           ),
        dm_api_call_analyse_column( `CONTINUOUS`, `DateOfBirth`   ),
        dm_api_call_conditional_expressions,
        dm_api_call_order_by( `ENTROPY`, `ASC` ),
        ( dm_api_call_conditional_expression, fail ; true ),
        dm_api_call_order_by( `|SIGNIFICANCE%|`, `DESC` ),
        ( dm_api_call_conditional_expression, fail ; true ),
        ( dm_api_call_conditional_statement( 8, `*` ), fail ; true ),
        ( dm_api_call_conditional_statement( 8, `"CreditHistory" <>
'Critical' AND ?` ), fail ; true ),
        true.

dm_api_call_startup :-
        (  dm_api_startup ~> _
        -> write( true )
        ;  write( fail )
        ),
        nl.

dm_api_call_connect( DSNstring, Driverstring ) :-
        (  dm_api_connect( DSNstring, Driverstring ) ~> _
        -> write( true )
        ;  write( fail )
        ),
        nl.


dm_api_call_base_table( Table ) :-
        (  dm_api_base_table( Table, BaseCount ) ~> _
        -> write( true ),
           fwrite( r, 16, 10, BaseCount )
        ;  write( fail )
        ),
        nl.
```

```
dm_api_call_target_expression( TargetExpression ) :-
        (  dm_api_target_expression( TargetExpression, TargetCount )
~> _
        -> write( true ),
           fwrite( r, 16, 10, TargetCount )
        ;  write( fail )
        ),
        nl.

dm_api_call_thresholds( SignificanceThreshold, BaseThreshold,
HitThreshold, TrueThreshold ) :-
        (  dm_api_thresholds( SignificanceThreshold, BaseThreshold,
HitThreshold, TrueThreshold ) ~> _
        -> write( true )
        ;  write( fail )
        ),
        nl.

dm_api_call_restart_analysis :-
        (  dm_api_restart_analysis ~> _
        -> write( true )
        ;  write( fail )
        ),
        nl.

dm_api_call_analyse_column( ModeString, Column ) :-
        (  dm_api_analyse_column( ModeString, Column, Solutions ) ~> _
        -> write( true ),
           fwrite( r, 16, 10, Solutions )
        ;  write( fail )
        ),
        nl.

dm_api_call_conditional_expressions :-
        (  dm_api_conditional_expressions( Solutions )
        -> write( true ),
           fwrite( r, 16, 10, Solutions )
        ;  write( fail )
        ),
        nl.

dm_api_call_conditional_expression :-
        Variables = variables(
                                  _BaseCount,
                                  _TargetCount,
                                  ConditionalCount,
                                  HitCount,
                                  MissCount,
                                  TrueCount,
                                  FalseCount,
                                  OtherCount,
                                  TruePercent,
                                  HitPercent,
                                  BasePercent,
                                  SignificancePercent,
                                  AbsSignificancePercent,
                                  Entropy
                                ),
        (
           dm_api_conditional_expression( ConditionalExpressionString,
Variables ) ~> _,
```

```
            len( ConditionalExpressionString, LenString ),
            write( true ),
            fwrite( r,  4, 10, LenString              ),
            write( ConditionalExpressionString        ),
            fwrite( r, 16, 10, ConditionalCount       ),
            fwrite( r, 16, 10, HitCount               ),
            fwrite( r, 16, 10, MissCount              ),
            fwrite( r, 16, 10, TrueCount              ),
            fwrite( r, 16, 10, FalseCount             ),
            fwrite( r, 16, 10, OtherCount             ),
            fwrite( f, 16,  3, TruePercent            ),
            fwrite( f, 16,  3, HitPercent             ),
            fwrite( f, 16,  3, BasePercent            ),
            fwrite( f, 16,  3, SignificancePercent    ),
            fwrite( f, 16,  3, AbsSignificancePercent ),
            fwrite( f, 16,  3, Entropy                )
        ;
            write( fail )
        ),
        nl.

dm_api_call_order_by( VariableString, AscDescString ) :-
        (  dm_api_order_by( VariableString, AscDescString ) ~> _
        -> write( true )
        ;  write( fail )
        ),
        nl.

dm_api_call_conditional_statement( TopSolutions, Mask ) :-
        Variables = variables(
                                _BaseCount,
                                _TargetCount,
                                ConditionalCount,
                                HitCount,
                                MissCount,
                                TrueCount,
                                FalseCount,
                                OtherCount,
                                TruePercent,
                                HitPercent,
                                BasePercent,
                                SsignificancePercent,
                                AbsSignificancePercent,
                                Entropy
                              ),
        (
            dm_api_conditional_statement( TopSolutions, Mask,
ConditionalStatementString, Variables ) ~> _,
            len( ConditionalStatementString, LenString ),
            write( true ),
            fwrite( r,  4, 10, LenString              ),
            write( ConditionalStatementString         ),
            fwrite( r, 16, 10, ConditionalCount       ),
            fwrite( r, 16, 10, HitCount               ),
            fwrite( r, 16, 10, MissCount              ),
            fwrite( r, 16, 10, TrueCount              ),
            fwrite( r, 16, 10, FalseCount             ),
            fwrite( r, 16, 10, OtherCount             ),
            fwrite( f, 16,  3, TruePercent            ),
            fwrite( f, 16,  3, HitPercent             ),
            fwrite( f, 16,  3, BasePercent            ),
```

```
            fwrite( f, 16,  3, SignificancePercent    ),
            fwrite( f, 16,  3, AbsSignificancePercent ),
            fwrite( f, 16,  3, Entropy                )
        ;
            write( fail )
        ),
        nl.
```

Execution

We will now execute *dm_api_test/0* for you and document its output in great detail.

```
?- dm_api_test. <enter>
```

Calling dm_api_call_startup/0

```
true
```

Calling dm_api_call_connect(`STATLOG`, `ACCESS`)

```
true
```

Calling dm_api_call_base_table(`GermanCredit`)

```
true0000000000001000
```

Note: 1000 records in total.

Calling dm_api_call_target_expression(`"LoanApproved"=2 AND "Foreign"='Yes'`)

```
true0000000000000296
```

Note: 296 records which match the target expression.

Calling dm_api_call_thresholds(5, 10, 10, 25)

```
true
```

Note: Set up the minimum values for 'interestingness'.

Calling dm_api_call_restart_analysis

```
true
```

Calling dm_api_call_analyse_column(`DISCRETE`, `PurposeOfLoan`)

```
true0000000000000002
```

Note: 2 unique 'bins'/categories found.

Calling dm_api_call_analyse_column(`DISCRETE`, `StatusSex`)

```
true0000000000000002
```

Note: 2 unique 'bins'/categories found.

Calling dm_api_call_analyse_column(`DISCRETE`, `Housing`)

```
true0000000000000003
```

Note: 3 unique 'bins'/categories found.

Calling dm_api_call_analyse_column(`CONTINUOUS`, `Duration`)

```
true0000000000000004
```

Note: 4 'bins' found.

Calling dm_api_call_analyse_column(`CONTINUOUS`, `Age`)

```
true0000000000000004
```

Note: 4 'bins' found.

Calling dm_api_call_analyse_column(`CONTINUOUS`, `DateOfBirth`)

```
true0000000000000000
```

Note: Nothing found.

Calling dm_api_call_conditional_expressions

```
true0000000000000015
```

Note: 2 + 2 + 3 + 4 + 4 + 0 = 15.

Calling dm_api_call_order_by(`ENTROPY`, `ASC`)

```
true
```

Calling (dm_api_call_conditional_expression, fail ; true)

```
true0018"Housing" =
'Free'0000000000000108000000000000004400000000000002520000000000000044
0000000000000064000000000000640          40.741          14.865
10.800          37.638          37.638          -3.204

true0015"Duration" =
1800000000000001130000000000000420000000000002540000000000000420000
0000000007100000000000633          37.168          14.189
11.300          25.568          25.568          -3.203
```

```
true0028"Duration" BETWEEN 18 AND
210000000000000151000000000000005100000000000002450000000000000005100000
000000001000000000000000604          33.775          17.230
15.100          14.104          14.104          -3.160

true0015"Duration" =
120000000000000179000000000000004900000000000002470000000000000004900000
000000001300000000000000574          27.374          16.554
17.900          -7.519           7.519          -3.144

true0029"PurposeOfLoan" =
'Furniture'000000000000181000000000000005700000000000002390000000000000
000570000000000001240000000000000580          31.492          19.257
18.100           6.391           6.391          -3.129

true0018"Housing" =
'Rent'000000000000017900000000000000690000000000002270000000000000069
000000000000011000000000000000594          38.547          23.311
17.900          30.228          30.228          -3.111

true0027"PurposeOfLoan" = 'New
Car'000000000000023400000000000000870000000000002090000000000000008700
000000000001470000000000000557          37.179          29.392
23.400          25.606          25.606          -3.045


true0023"Age" BETWEEN 19 AND
260000000000000240000000000000009400000000000020200000000000000940000
000000001460000000000000558          39.167          31.757
24.000          32.320          32.320          -3.030

true0023"Age" BETWEEN 39 AND
750000000000000320000000000000008300000000000021300000000000000830000
000000002370000000000000467          25.938          28.041
32.000         -12.373          12.373          -3.013

true0039"StatusSex" = 'Married/Divorced
Female'000000000000031000000000000001070000000000001890000000000000010
700000000000002030000000000000501          34.516          36.149
31.000          16.609          16.609          -2.971

true0023"Age" BETWEEN 19 AND
290000000000000037100000000000001360000000000001600000000000000136000000
000000002350000000000000469          36.658          45.946
37.100          23.844          23.844          -2.891

true0027"StatusSex" = 'Single
Male'000000000000054800000000000000146000000000000150000000000000001460
000000000000402000000000000302          26.642          49.324
54.800          -9.992           9.992          -2.852

true0017"Housing" =
'Own'000000000000071300000000000000183000000000000113000000000000001830
000000000000530000000000000174          25.666          61.824
71.300         -13.290          13.290          -2.808
```

```
        true0028"Duration" BETWEEN 18 AND
        40000000000000048600000000000001660000000000000130000000000000001660000
        00000000032000000000000000384          34.156          56.081
        48.600          15.393          15.393          -2.804

        true0023"Age" BETWEEN 19 AND
        33000000000000051600000000000001780000000000000118000000000000001780000
        00000000033800000000000000366          34.496          60.135
        51.600          16.541          16.541          -2.775
        fail
```

Note: 15 listed.

Calling dm_api_call_order_by(`|SIGNIFICANCE|`, `DESC`)

```
        true
```

Note: Order now by significance in descending order.

Calling (dm_api_call_conditional_expression, fail ; true)

```
        true0018"Housing" =
        'Free'00000000000001080000000000000044000000000000025200000000000000044
        00000000000000640000000000000640          40.741          14.865
        10.800          37.638          37.638          -3.204

        true0023"Age" BETWEEN 19 AND
        26000000000000024000000000000000940000000000000202000000000000000940000
        00000000014600000000000000558          39.167          31.757
        24.000          32.320          32.320          -3.030

        true0018"Housing" =
        'Rent'00000000000001790000000000000006900000000000002270000000000000069
        00000000000001100000000000000594          38.547          23.311
        17.900          30.228          30.228          -3.111

        true0027"PurposeOfLoan" = 'New
        Car'00000000000002340000000000000087000000000000020900000000000000087000
        00000000001470000000000000557          37.179          29.392
        23.400          25.606          25.606          -3.045

        true0015"Duration" =
        18000000000000011300000000000000420000000000000254000000000000000420000
        00000000071000000000000000633          37.168          14.189
        11.300          25.568          25.568          -3.203

        true0023"Age" BETWEEN 19 AND
        29000000000000037100000000000000136000000000000016000000000000000136000000
        00000000235000000000000000469          36.658          45.946
        37.100          23.844          23.844          -2.891

        true0039"StatusSex" = 'Married/Divorced
        Female'00000000000003100000000000000107000000000000018900000000000000107
        00000000000000203000000000000000501          34.516          36.149
        31.000          16.609          16.609          -2.971
```

```
true0023"Age" BETWEEN 19 AND
330000000000000516000000000000017800000000000001180000000000001780000
000000000338000000000000366          34.496          60.135
51.600          16.541          16.541          -2.775

true0028"Duration" BETWEEN 18 AND
400000000000000486000000000000016600000000000001300000000000001660000
000000000320000000000000384          34.156          56.081
48.600          15.393          15.393          -2.804

true0028"Duration" BETWEEN 18 AND
210000000000000151000000000000005100000000000024500000000000000510000
000000000100000000000000604          33.775          17.230
15.100          14.104          14.104          -3.160

true0017"Housing" =
'Own'000000000000071300000000000001830000000000001130000000000001830
000000000000530000000000000174          25.666          61.824
71.300          -13.290          13.290          -2.808

true0023"Age" BETWEEN 39 AND
750000000000000320000000000000008300000000000002130000000000000830000
000000000237000000000000467          25.938          28.041
32.000          -12.373          12.373          -3.013

true0027"StatusSex" = 'Single
Male'000000000000054800000000000001460000000000000150000000000000001460
000000000000402000000000000302          26.642          49.324
54.800          -9.992          9.992          -2.852

true0015"Duration" =
120000000000000179000000000000004900000000000002470000000000000490000
000000000130000000000000574          27.374          16.554
17.900          -7.519          7.519          -3.144

true0029"PurposeOfLoan" =
'Furniture'000000000000018100000000000000570000000000000239000000000000
000570000000000000124000000000000580          31.492          19.257
18.100          6.391          6.391          -3.129

fail
```

Note: 15 listed.

Calling (dm_api_call_conditional_statement(8, `*`), fail ; true)

Note: '*' means replace with any combination of conditional expressions using the AND operator.

```
true0046"Housing" = 'Rent' AND "Age" BETWEEN 19 AND
290000000000000113000000000000004900000000000002470000000000000490000
000000000064000000000000640          43.363          16.554
11.300          46.496          46.496          -3.193

true0067"Age" BETWEEN 19 AND 26 AND "StatusSex" = 'Married/Divorced
Female'000000000000012500000000000000540000000000000242000000000000005
400000000000000710000000000000633          43.200          18.243
12.500          45.946          45.946          -3.175
```

```
true0067"Age" BETWEEN 19 AND 29 AND "StatusSex" = 'Married/Divorced
Female'0000000000000171000000000000000700000000000000226000000000000000700
0000000000000010100000000000000603          40.936          23.649
17.100          38.296          38.296          -3.115

true0046"Housing" = 'Rent' AND "Age" BETWEEN 19 AND
330000000000000130000000000000005700000000000002390000000000000005700000
0000000007300000000000000631          43.846          19.257
13.000          48.129          48.129          -3.167

true0055"PurposeOfLoan" = 'New Car' AND "Age" BETWEEN 19 AND
330000000000000104000000000000005300000000000002430000000000000053000000
0000000005100000000000000653          50.962          17.905
10.400          72.167          72.167          -3.194

true0067"StatusSex" = 'Married/Divorced Female' AND "Age" BETWEEN 19
AND
330000000000000205000000000000008300000000000002130000000000000083000000
0000000012200000000000000582          40.488          28.041
20.500          36.783          36.783          -3.070

fail
```

Note: 6 listed.

Calling (dm_api_call_conditional_statement(8, `"CreditHistory" <> 'Critical' AND ?`), fail ; true)

```
true0057"CreditHistory" <> 'Critical' AND "Age" BETWEEN 19 AND
260000000000000201000000000000008200000000000002140000000000000082000000
0000000011900000000000000585          40.796          27.703
20.100          37.824          37.824          -3.075

true0052"CreditHistory" <> 'Critical' AND "Housing" =
'Rent'0000000000000142000000000000000600000000000000236000000000000006000
0000000000000082000000000000000622          42.254          20.270
14.200          42.748          42.748          -3.152

true0061"CreditHistory" <> 'Critical' AND "PurposeOfLoan" = 'New
Car'0000000000000156000000000000000690000000000000227000000000000006900
0000000000008700000000000000617          44.231          23.311
15.600          49.428          49.428          -3.128

true0057"CreditHistory" <> 'Critical' AND "Age" BETWEEN 19 AND
290000000000000029900000000000001180000000000000178000000000000011800000
0000000018100000000000000523          39.465          39.865
29.900          33.327          33.327          -2.955

true0073"CreditHistory" <> 'Critical' AND "StatusSex" =
'Married/Divorced
Female'0000000000000231000000000000009100000000000002050000000000000091
0000000000000014000000000000000564          39.394          30.743
23.100          33.088          33.088          -3.040

true0057"CreditHistory" <> 'Critical' AND "Age" BETWEEN 19 AND
330000000000000040200000000000001470000000000000149000000000000014700000
0000000025500000000000000449          36.567          49.662
40.200          23.538          23.538          -2.860
```

```
fail
yes
```

Note: CreditHistory was not one of the columns we initially analysed.

Desktop Data Mining Toolkit Example

One of the supplied examples is a Windows Desktop data mining toolkit example (DIALOGDM.PL).

Loading And Running The Program

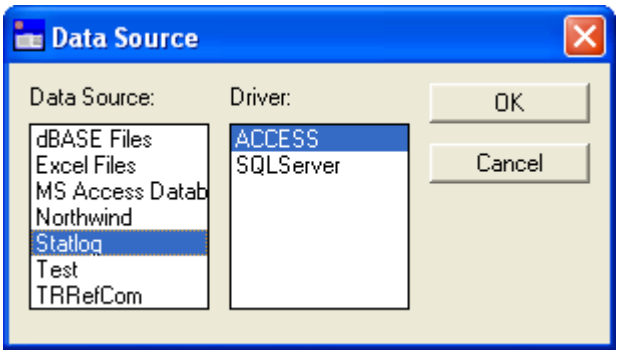
Launch *WIN-PROLOG* and load (via the File\Load menu option) the file DIALOGDM.PL from the EXAMPLES\DATAMITE directory; execute the following goal from the Prolog prompt:

```
?- run. <enter>
```

Here are a few screenshots of the example in use.

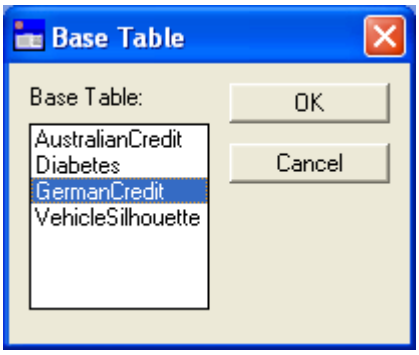
Selecting The Data Source

First of all, you need to select the data source and a driver. The driver field defaults to `ACCESS`, the driver for Microsoft Access.



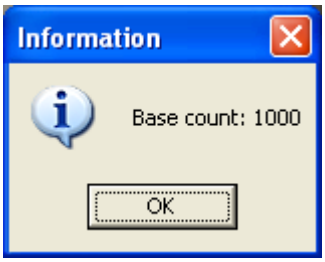
Selecting The Base Table

Next, you need to select the base table. This is the table you wish to mine.



Confirmation Of The Base Count

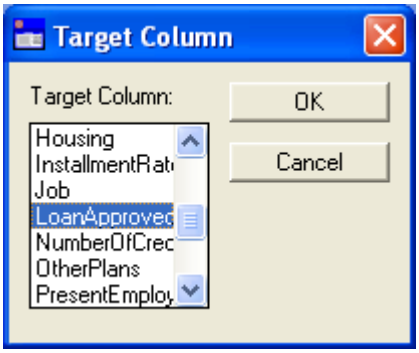
This dialog gives confirmation of the base count.



The base count is the number of records in the base table.

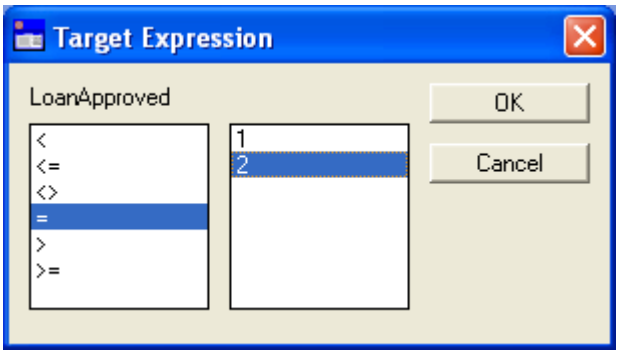
Selecting The Target Column

Next, you need to select the target column.



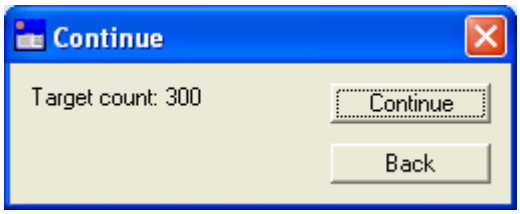
Specifying The Target Expression

Having selected the target column on the previous dialog, you now need to complete the target expression by selecting an SQL comparison type and a value.



Confirmation Of The Target Count

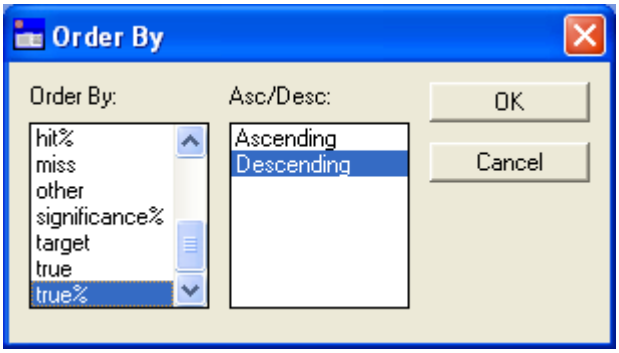
This dialog gives confirmation of the target count.



Click [Continue] to continue or [Back] to return to the Target Column dialog and set a different target expression.

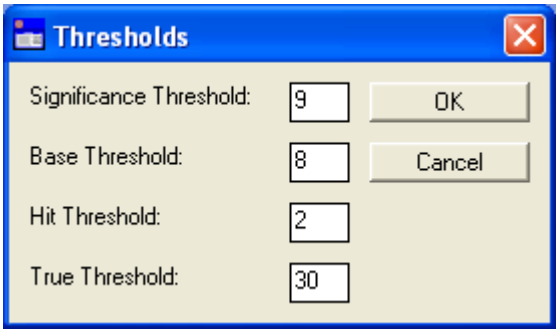
Ordering The Solutions

You next need to specify how you want the solutions ordered when returned to you. Here we have sorted them into ***desc***ending order by ***true%***.



Setting The Thresholds

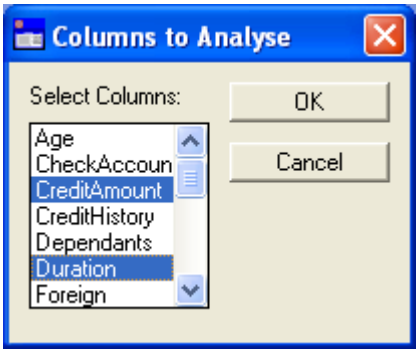
You next need to set the four thresholds.



The True threshold is calculated automatically using the formula, (Target Count / Base Count) * 100.

Selecting The Columns To Analyse

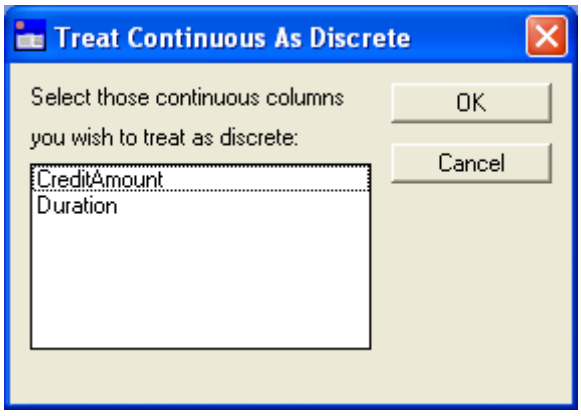
You next need to select one or more columns that you wish to analyse.



We have chosen CreditAmount and Duration.

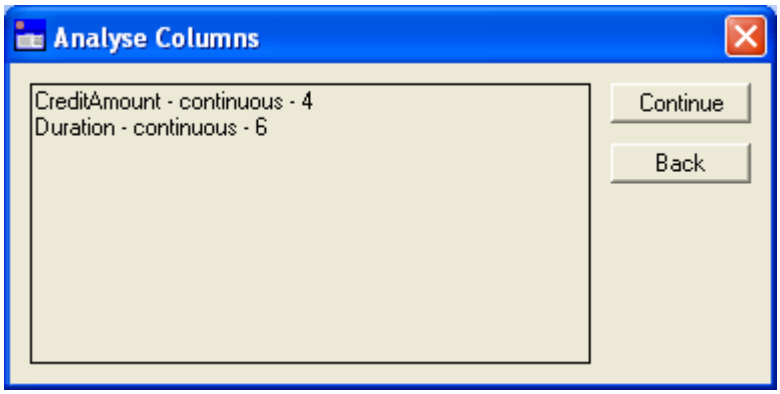
Internal Handling Of Each Column

The values in each selected column are automatically internally handled as either discrete or continuous. This dialog allows you to override the automatically assigned setting for a continuous column in order to treat it as discrete.



Solutions Found

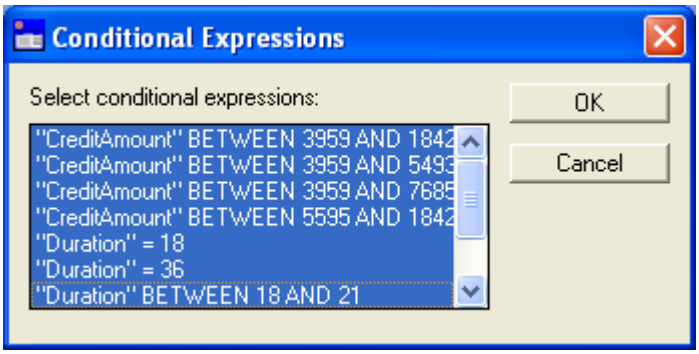
This dialog calls *dm_api_analyse_column/3* one or more times and then tells you how many conditional expressions were found for a particular column during the analysis phase.



Click [Continue] to continue or [Back] to return to the Select Columns dialog and select a different set of columns.

Conditional Expressions

The next screen lists all the conditional expressions found.



Select all the conditional expressions that you are interested in; by default, all conditional

expressions are selected.

Export Confirmation

The next dialog asks whether you want to export the selected conditional expressions' data to a comma-separated-value (CSV) file.

Clicking the [Yes] button takes you to the Export dialog.

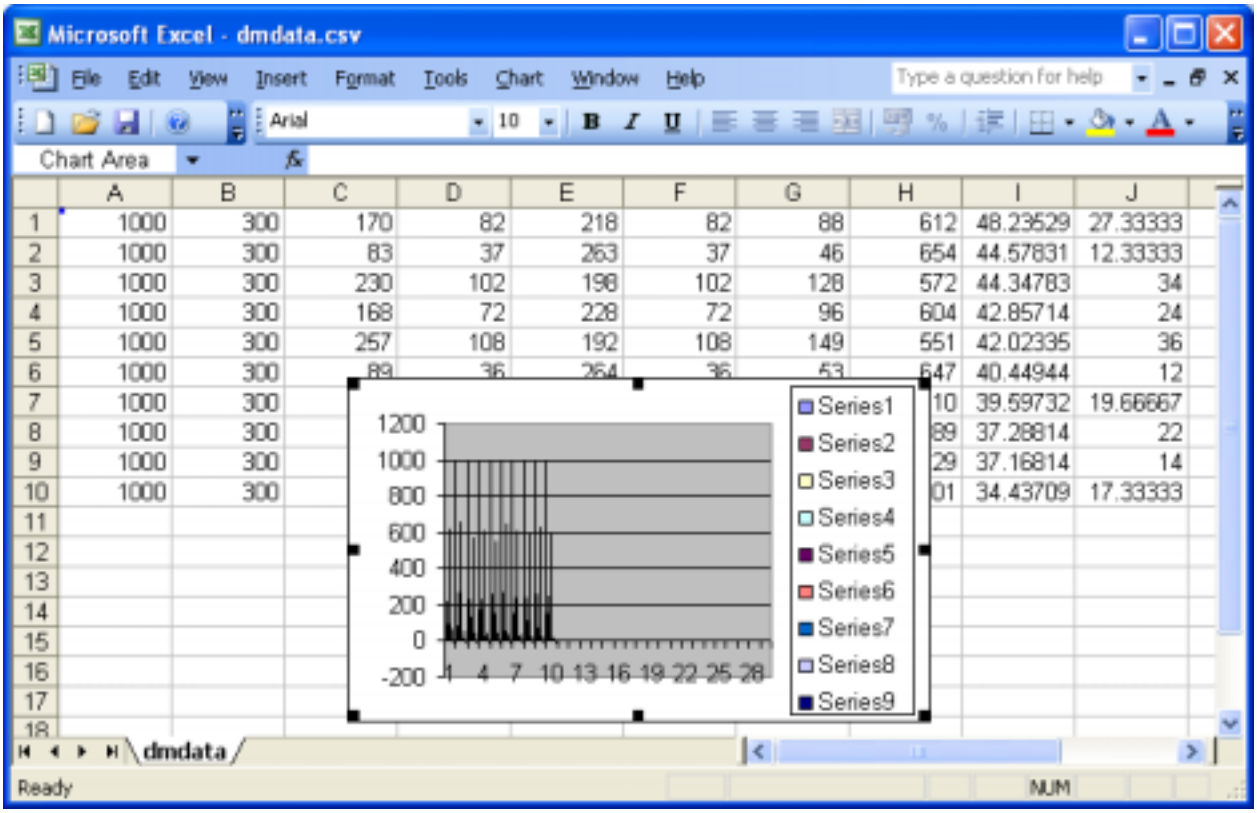
Selecting The Columns To Be Exported

Select the columns that you want to export.

When you click the [OK] button, the data for the selected columns will be saved as a CSV file named DMDATA.CSV. You will then be taken to the Conditional Expression dialog.

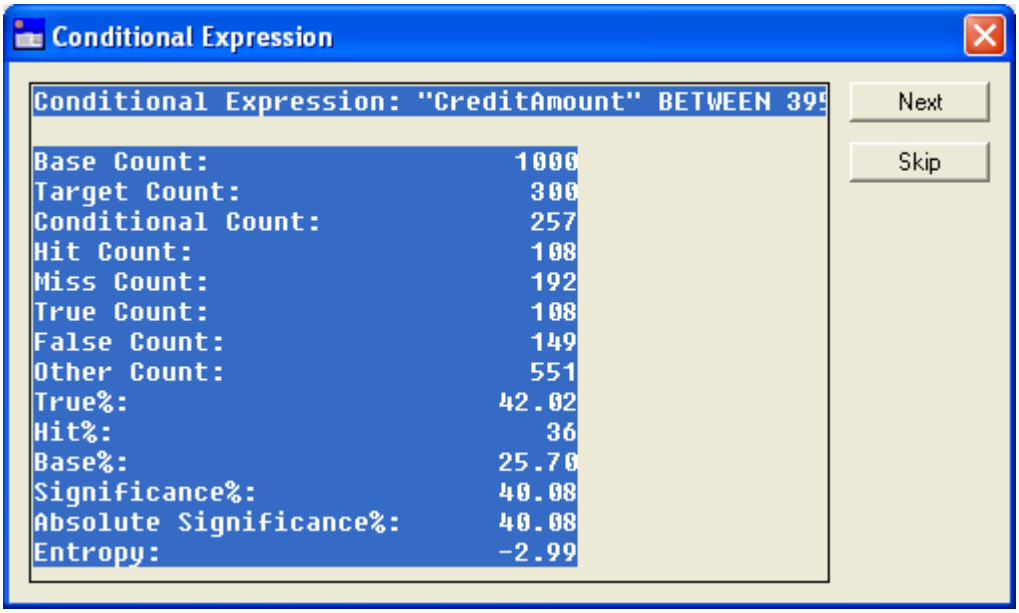
Viewing The Conditional Expressions In Excel

The DMDATA.CSV file can be viewed in Excel.



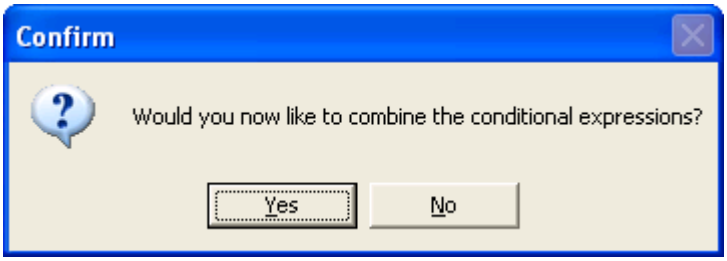
Conditional Expression

Each conditional expression will be shown in its own dialog. Clicking 'Next' takes you to the next conditional expression's dialog. Clicking 'Skip' takes you to the combination stage.



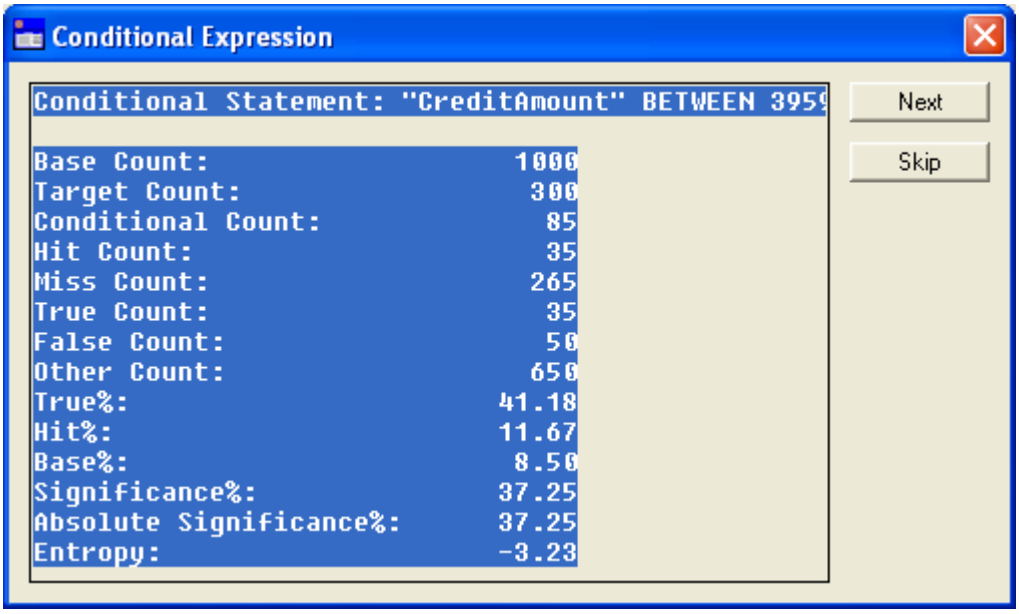
Combining The Conditional Expressions

Confirmation is asked for before combining the conditional expressions to make one or more conditional statements.



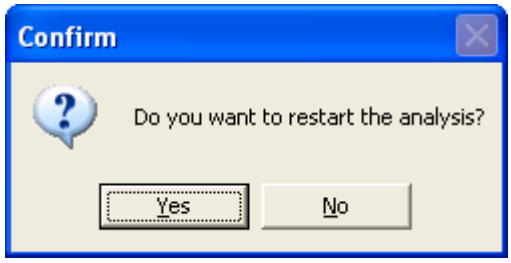
Conditional Statement

Each conditional statement will be shown in its own dialog. Clicking 'Next' takes you to the next conditional statement's dialog. Clicking 'Skip' takes you to the restart analysis dialog.



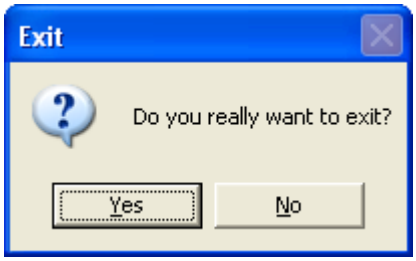
Restarting The Analysis

This dialog allows you to restart the analysis by returning you to the Thresholds dialog.



Exiting The Program

If you click [No], you will be taken to the following dialog:

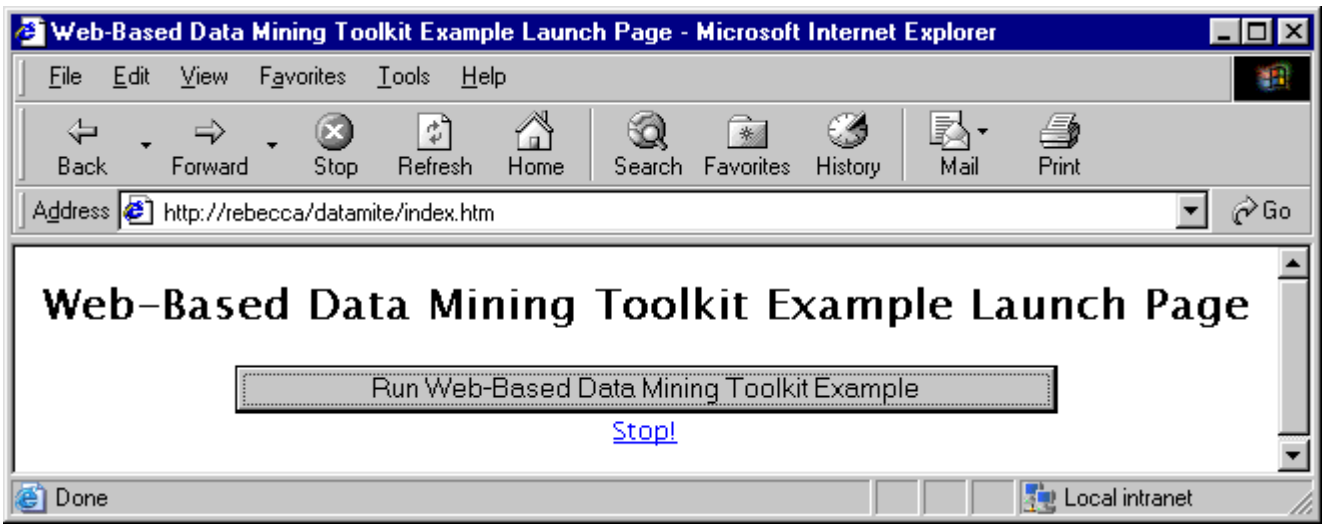


Clicking [Yes] will exit the program; clicking [No] will rerun the program right from the very beginning.

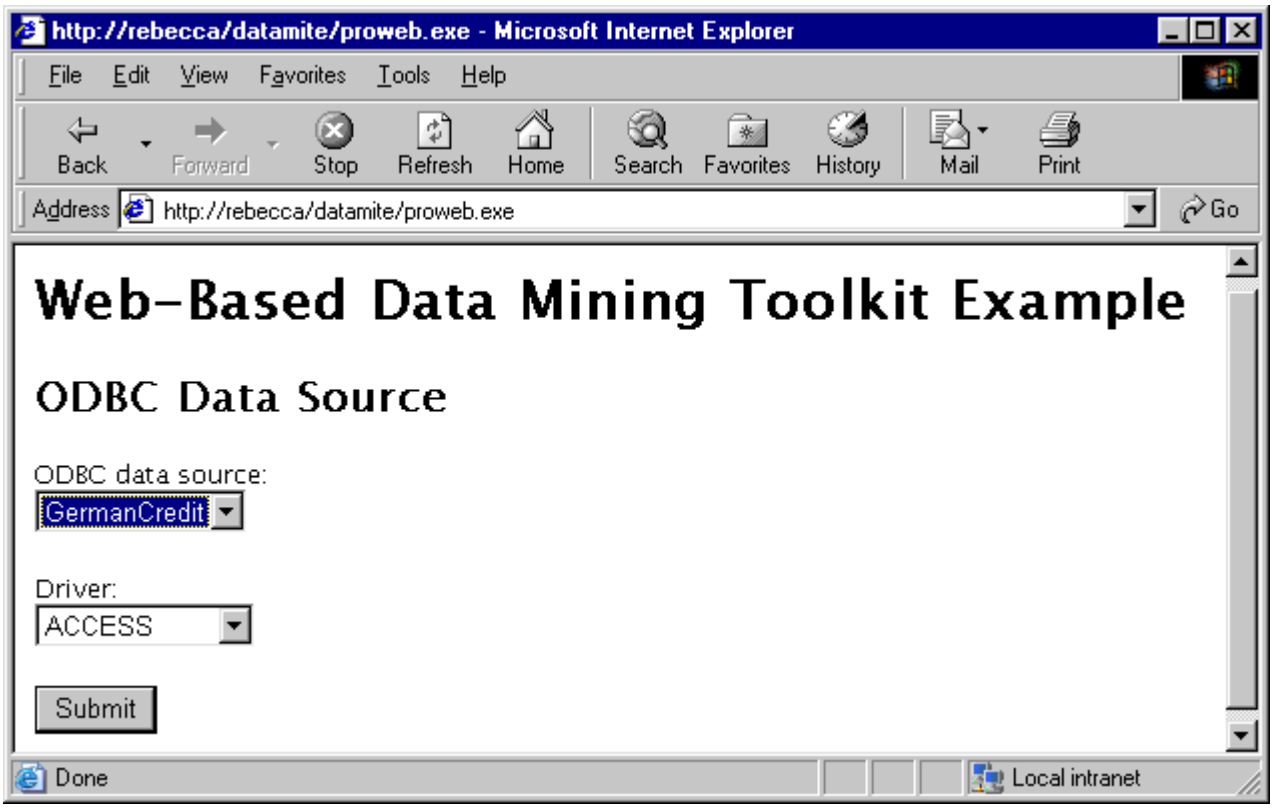
ProWeb Data Mining Toolkit Example

One of the supplied examples is a web-based data mining toolkit example, utilising the LPA products, ProWeb and Prodata, and using Scalable Vector Graphics (SVG). The example is in the file, PW_DMAPI.PL, in the ProWeb\Examples directory. A free SVG plug-in for your web browser is available from the Adobe web site. Here are a few screenshots of the example in use:

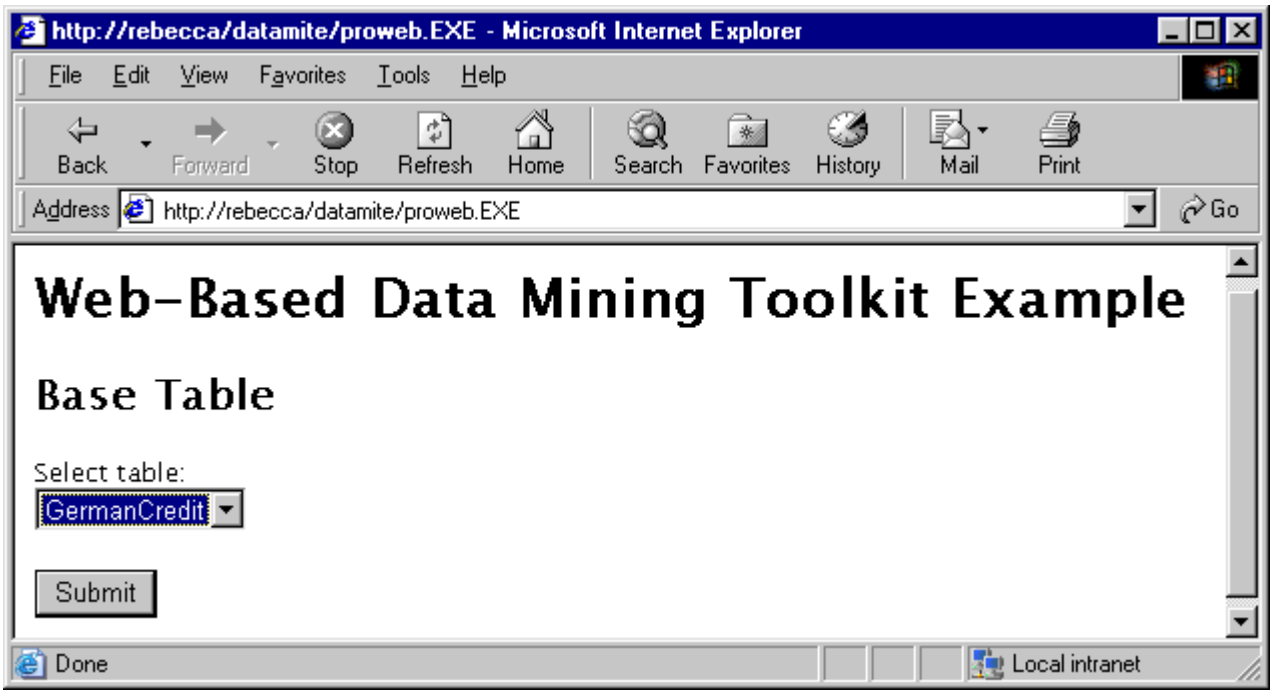
Launch Page



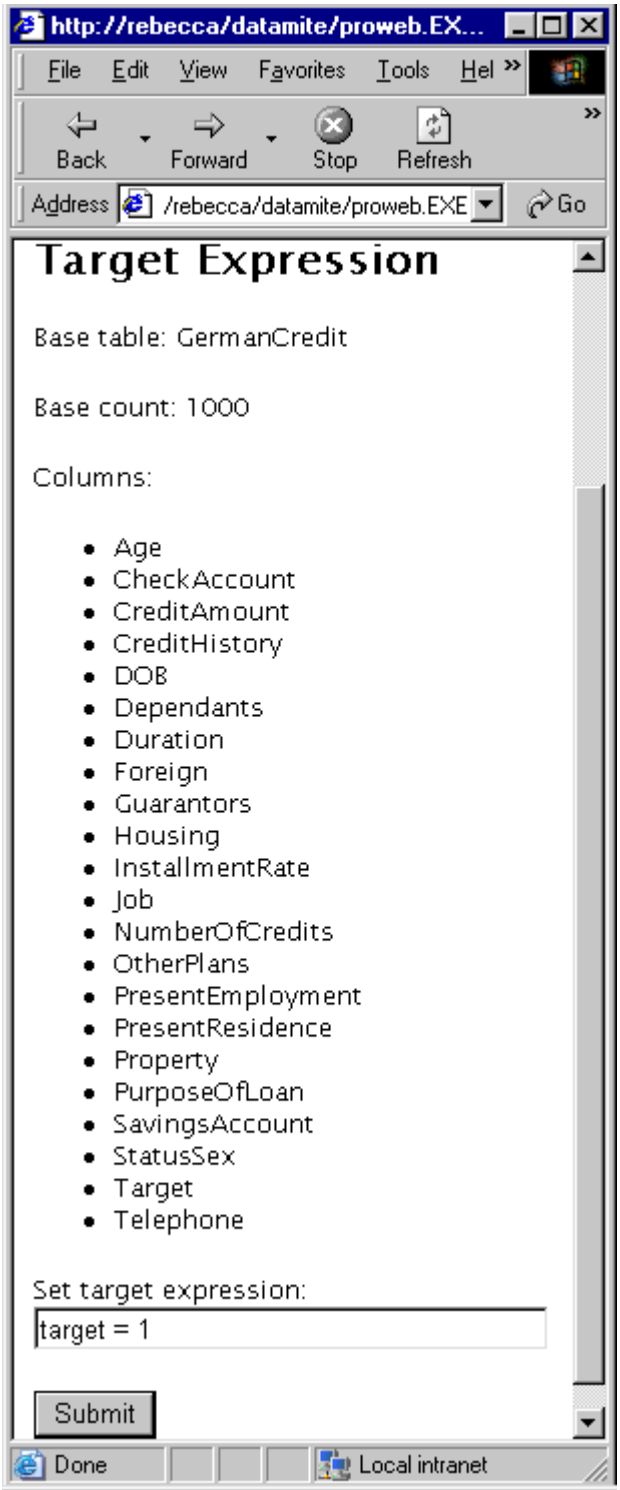
Selecting The Data Source



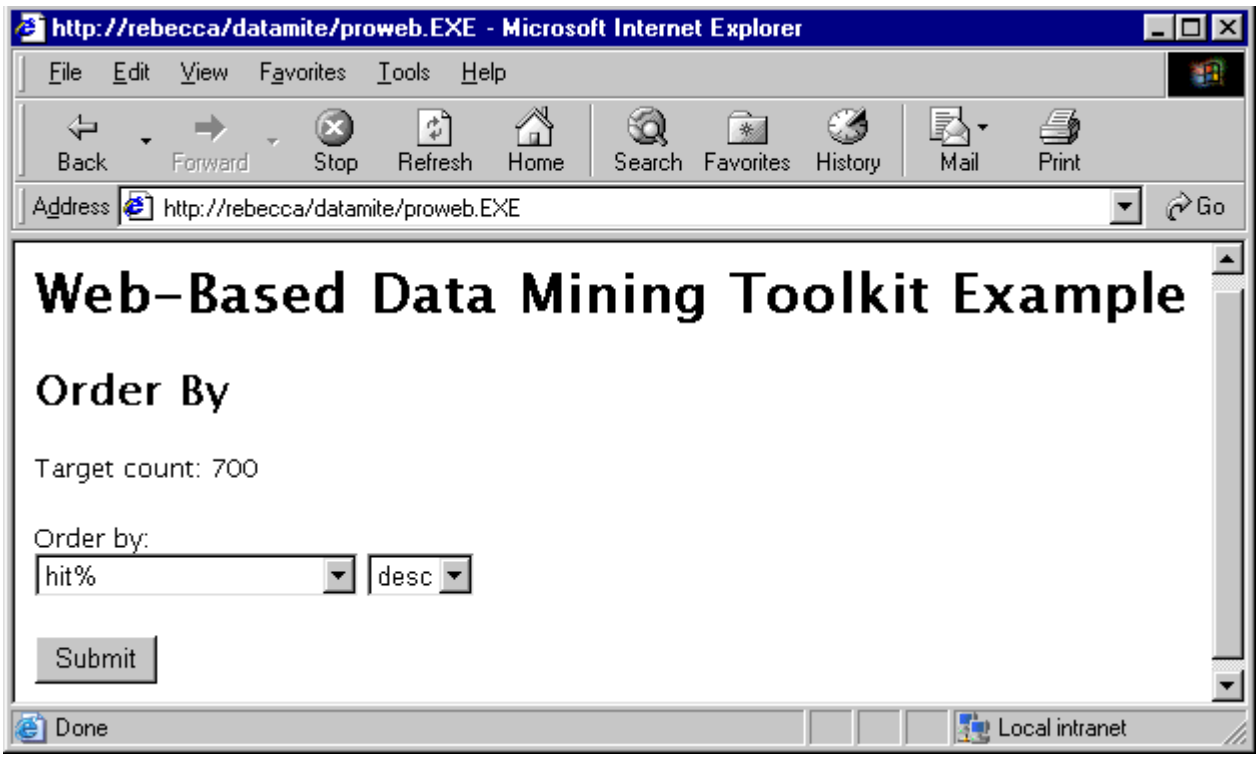
Selecting The Base Table



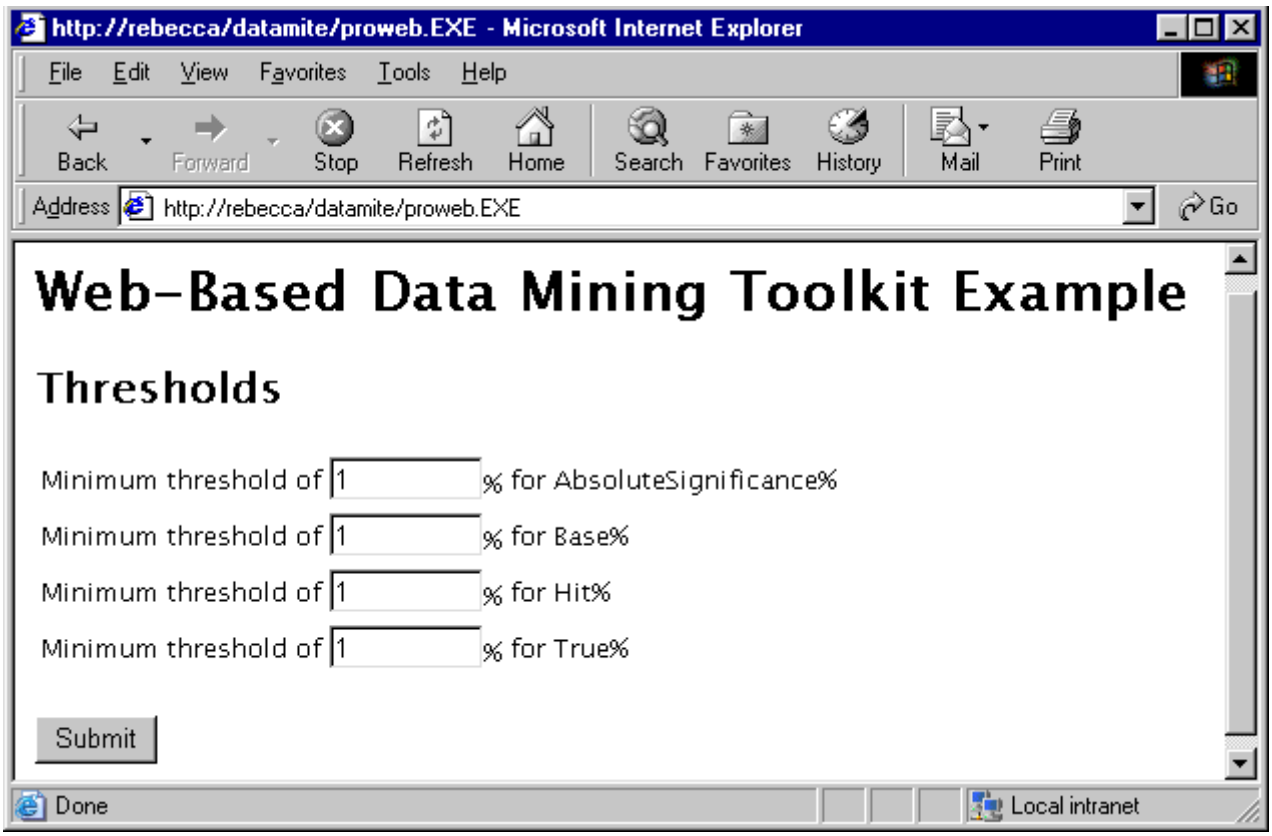
Specifying The Target Expression


http://rebecca/datamite/proweb.EX...
File Edit View Favorites Tools Hel
Back Forward Stop Refresh
Address /rebecca/datamite/proweb.EXE
Go
Target Expression
Base table: GermanCredit
Base count: 1000
Columns:
Age
CheckAccount
CreditAmount
CreditHistory
DOB
Dependants
Duration
Foreign
Guarantors
Housing
InstallmentRate
Job
NumberOfCredits
OtherPlans
PresentEmployment
PresentResidence
Property
PurposeOfLoan
SavingsAccount
StatusSex
Target
Telephone
Set target expression:
target = 1
Submit
Done
Local intranet

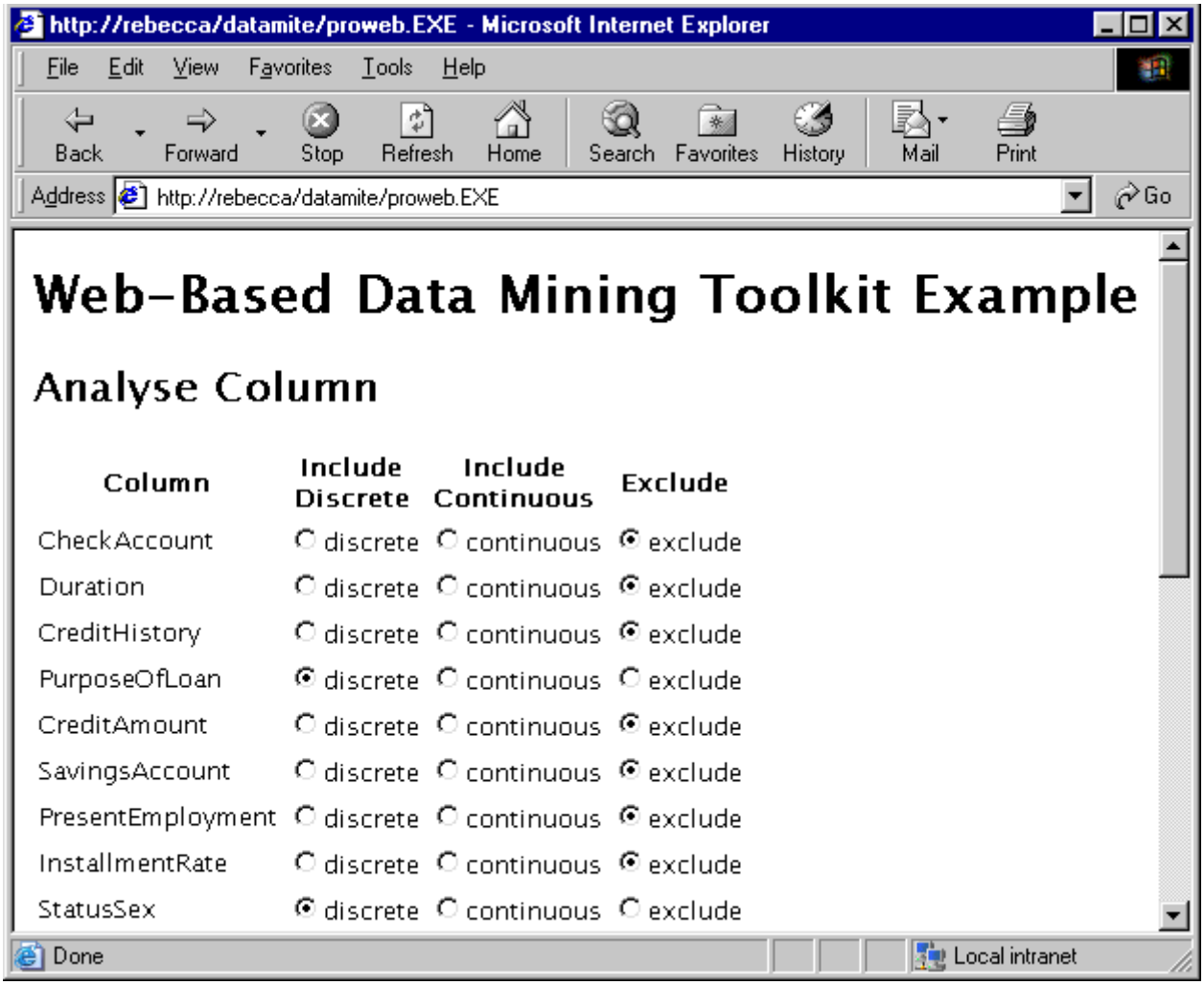
Ordering The Solutions



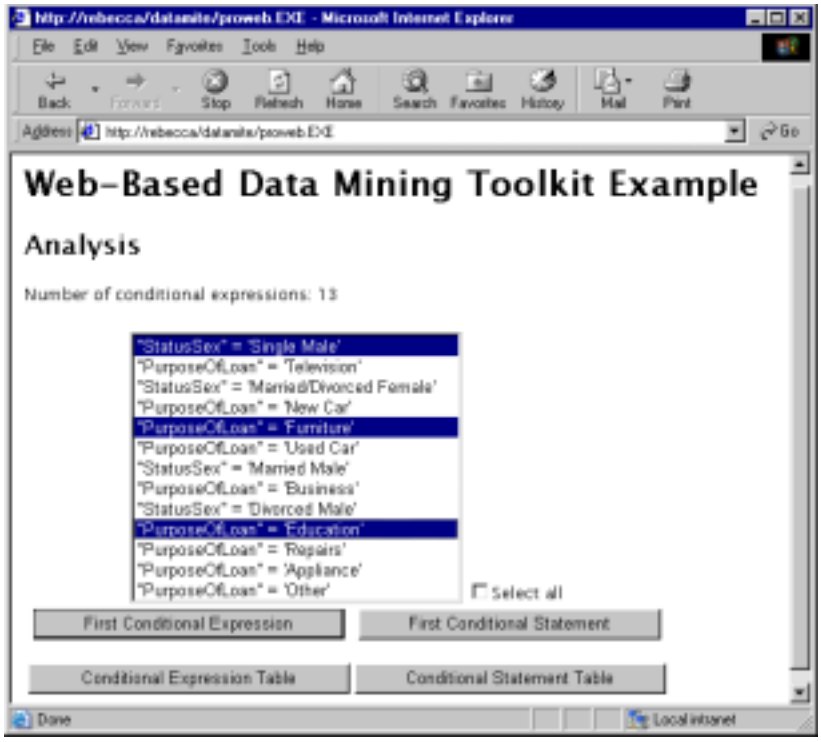
Setting The Thresholds



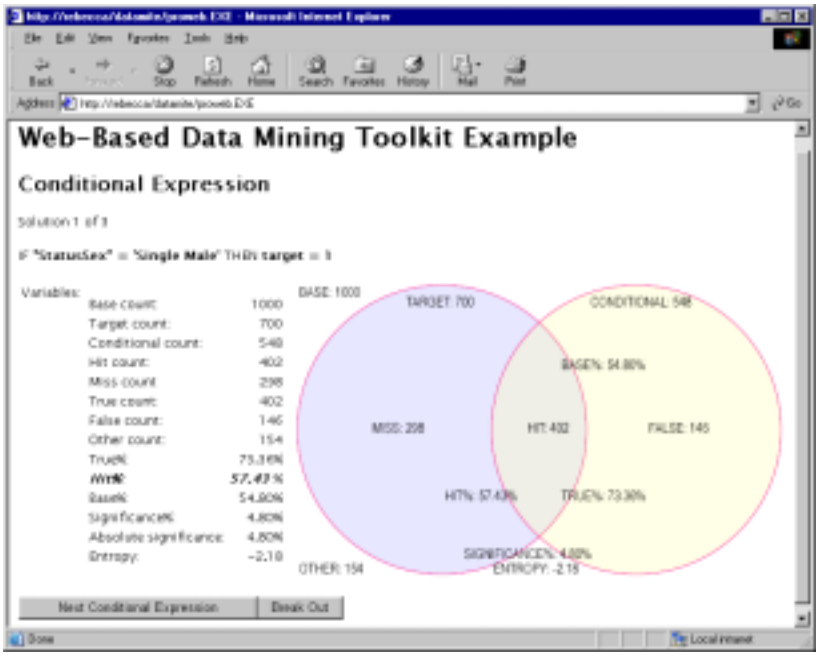
Selecting The Columns To Analyse



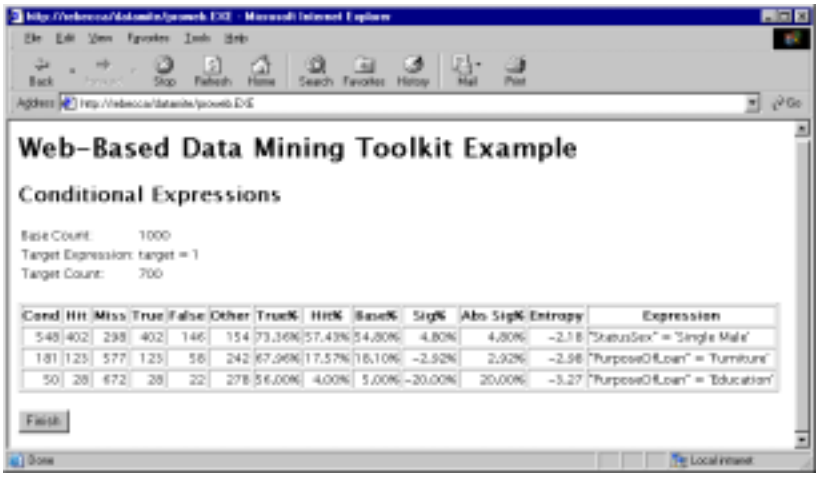
Solutions Found



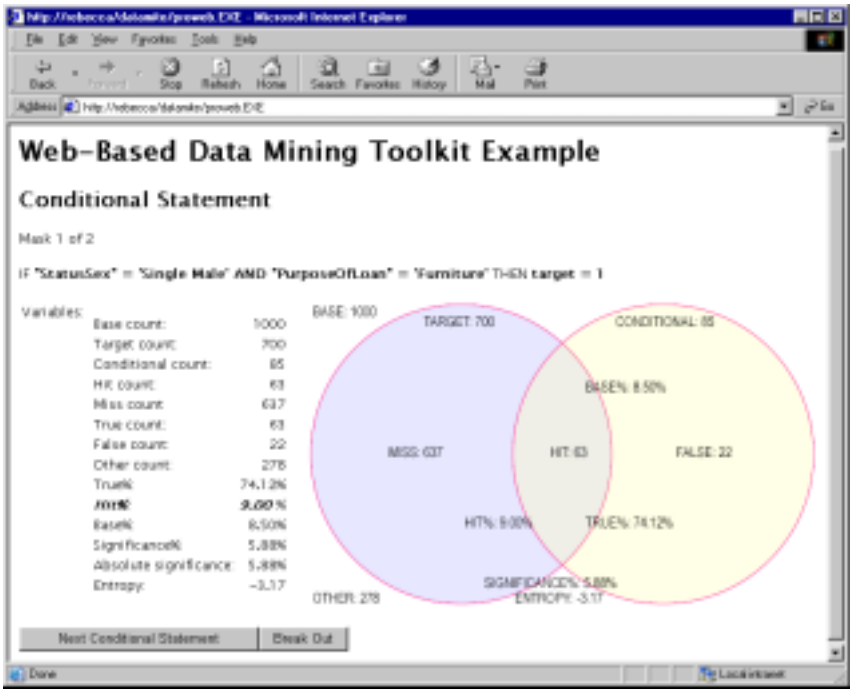
Conditional Expression



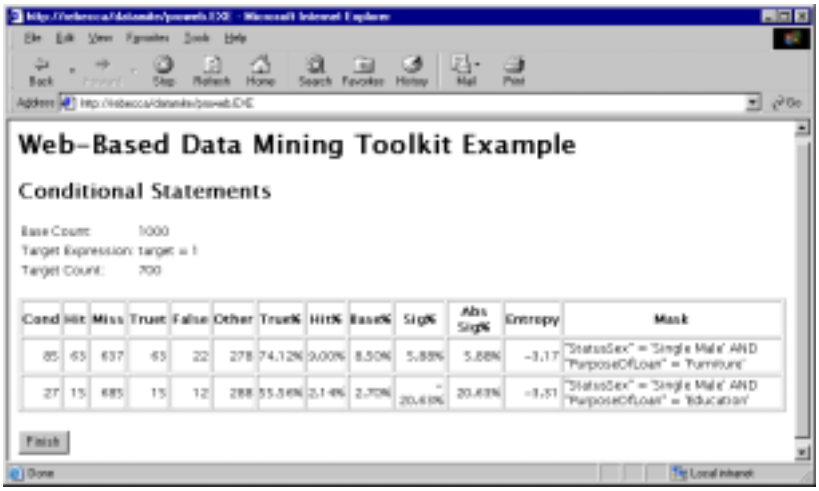
Conditional Expressions

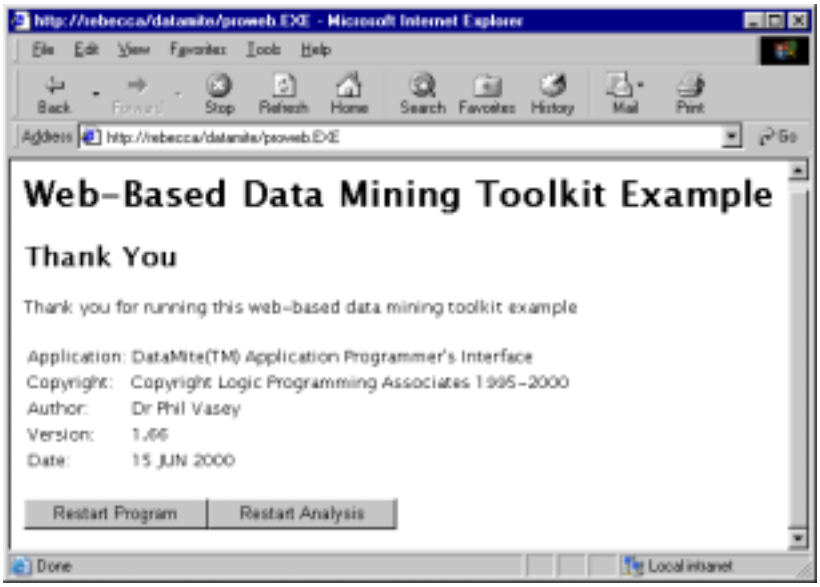


Conditional Statement


Web-Based Data Mining Toolkit Example
Conditional Statement
Rule 1 of 2
IF "StatusSex" = 'Single Male' AND "PurposeOfLoan" = 'Furniture' THEN target = 1
Variables:
Base count: 1000
Target count: 700
Conditional count: 85
Hit count: 63
Miss count: 637
True count: 63
False count: 22
Other count: 278
True%: 74.12%
Hit%: 9.00 %
Base%: 8.50%
Significance%: 5.88%
Absolute significance: 5.88%
Entropy: -3.17
BASE: 1000
TARGET: 700
CONDITIONAL: 85
BASE%: 8.50%
MISS: 637
HIT: 63
FALSE: 22
HIT%: 9.00%
TRUE%: 74.12%
SIGNIFICANCE%: 5.88%
ENTROPY: -3.17
OTHER: 278
Next Conditional Statement
Break Out
Done

Conditional Statements





Glossary Of Terms

?	A complete or partial **mask**.
*	A complete or partial **mask**.
absolute significance%	Formula = abs(**Significance%**)
ACCESS	The name of the **driver** used to connect to a Microsoft Access-based data source.
asc	Short for ascending order.
Base Count	The number of **row**s/**record**s in the **base table**.
base table	The table in the ODBC data source chosen to be data mined.
base threshold	The minimum **threshold** for the **Base%** variable
Base%	The percentage of the **Base** having the **condition**.
bin	A container for a column's unique value or range of values.
column name	The name of a column in the **base table**. Used within the **target expression** or a **conditional expression**.
combining	ANDing two or more **conditional expression**s to make a **conditional statement**.
ConditionalCount	The number of **row**s/**record**s in the **base table** where the **condition** holds.
conditional expression	An SQL statement representing a significant condition of the form: "ColumnName" = Value or "ColumnName" BETWEEN LowerValue AND UpperValue.
conditional statement	Two or more **conditional expression**s ANDed together.
continuous	The values in a single column are continuous (e.g. the integers from 1 through to 10). As opposed to **discrete**.
desc	Short for descending order.
discrete	The values in a single column are discrete (e.g. apple, pear and banana). As opposed to **continuous**.
Entropy	A measure of interestingness. The default formula

	aln(**True** / **Base**) – aln(**False** / **Base**) -aln(**Miss** / **Base**) – aln(**Other** / **Base**)
	produces a relative value between the number of **positives** and the number of **negatives**.
FalseCount	Formula = **Conditional** - **True**
Hit	Same as **True**
hit count	The number of **row**s in which both the **target** and the candidate **condition** hold. Same as **true count**.
hit threshold	The minimum **threshold** for the **Hit%** variable.
Hit%	Formula = (**Hit** / **Target**) * 100
mask	Used to obtain none, one or more **conditional statement**s.
Miss	Formula = **Target** - **Hit**
miss count	The number of **row**s in which the candidate **condition** holds but the **target** does not. Same as **negatives**.
negatives	The number of **row**s in which the candidate **condition** holds but the **target** does not. Same as **miss count**.
order by	Specifies in what order the solutions are returned.
OtherCount	The number of **row**s in which neither the **target** or the candidate **condition** hold.
	Formula = **Base** – (**Target** + **Conditional** – **Hit**)
positives	The number of **row**s in which both the **target** and the candidate **condition** hold. Same as **hit count**.
record	A **row** within the **base table**
row	A **record** within the **base table**
significance	An indication of deviation from the norm. This can be either positive (i.e. over representation) or negative (i.e. under representation)
significance threshold	The minimum **threshold** for the **AbsoluteSignificance%** variable.
Significance%	Formula = ((**Hit%** - **Base%**) / **Base%**) * 100
	Formula = (**True%** - **Target%**) / **Target%**
solutions	The number of significant **conditional expression**s.
SQL	Structured Query Language.
string	A data type in Win-Prolog denoted by `...` quote marks.
target count	The number of **record**s/**row**s in the **base table** where the **target**

	expression applies.
target expression	The SQL expression (e.g. `` `"LoanApproved" = 2 AND "Foreign" = 'Yes'` ``) which forms the target of the data mining process.
Target%	Formula = **Target** / **Base**
thresholds	The minimum values for 'interestingness'.
True	Same as **Hit**
true count	The number of **row**s in which both the **target** and the candidate **condition** hold. Same as **hit count**.
true threshold	The minimum **threshold** for the **True%** variable
True%	Formula = (**True** / **Conditional**) * 100
value	Something in a column/row intersection cell.
variables	A Prolog structure.
view	The columns in the **base table** being analysed.

Appendix A: Extract From 'An Overview Of Data Mining Techniques'

This Appendix contains an extract from 'An Overview of Data Mining Techniques' (http://www.thearling.com/text/dmtechniques/dmtechniques.htm), which itself, is excerpted from the book 'Building Data Mining Applications for CRM' by Alex Berson, Stephen Smith and Kurt Thearling.

2.4. Rule Induction

Rule induction is one of the major forms of data mining and is perhaps the most common form of knowledge discovery in unsupervised learning systems. It is also perhaps the form of data mining that most closely resembles the process that most people think about when they think about data mining, namely "mining" for gold through a vast database. The gold in this case would be a rule that is interesting - that tells you something about your database that you didn't already know and probably weren't able to explicitly articulate (aside from saying "show me things that are interesting").

Rule induction on a data base can be a massive undertaking where all possible patterns are systematically pulled out of the data and then an accuracy and significance are added to them that tell the user how strong the pattern is and how likely it is to occur again. In general these rules are relatively simple such as for a market basket database of items scanned in a consumer market basket you might find interesting correlations in your database such as:

- If bagels are purchased then cream cheese is purchased 90% of the time and this pattern occurs in 3% of all shopping baskets.
- If live plants are purchased from a hardware store then plant fertilizer is purchased 60% of the time and these two items are bought together in 6% of the shopping baskets.

The rules that are pulled from the database are extracted and ordered to be presented to the user based on the percentage of times that they are correct and how often they apply.

The bane of rule induction systems is also its strength - that it retrieves all possible interesting patterns in the database. This is a strength in the sense that it leaves no stone unturned but it can also be viewed as a weaknes because the user can easily become overwhelmed with such a large number of rules that it is difficult to look through all of them. You almost need a second pass of data mining to go through the list of interesting rules that have been generated by the rule induction system in the first place in order to find the most valuable gold nugget amongst them all. This overabundance of patterns can also be problematic for the simple task of prediction because all possible patterns are

culled from the database there may be conflicting predictions made by equally interesting rules. Automating the process of culling the most interesting rules and of combing the recommendations of a variety of rules are well handled by many of the commercially available rule induction systems on the market today and is also an area of active research.

Applying Rule induction to Business

Rule induction systems are highly automated and are probably the best of data mining techniques for exposing all possible predictive patterns in a database. They can be modified to for use in prediction problems but the algorithms for combining evidence from a variety of rules comes more from rules of thumbs and practical experience.

In comparing data mining techniques along an axis of explanation neural networks would be at one extreme of the data mining algorithms and rule induction systems at the other end. Neural networks are extremely proficient and saying exactly what must be done in a prediction task (e.g. who do I give credit to / who do I deny credit to) with little explanation. Rule induction systems when used for prediction on the other hand are like having a committee of trusted advisors each with a slightly different opinion as to what to do but relatively well grounded reasoning and a good explanation for why it should be done.

The business value of rule induction techniques reflects the highly automated way in which the rules are created which makes it easy to use the system but also that this approach can suffer from an overabundance of interesting patterns which can make it complicated in order to make a prediction that is directly tied to return on investment (ROI).

What is a rule?

In rule induction systems the rule itself is of a simple form of "if this and this and this then this". For example a rule that a supermarket might find in their data collected from scanners would be: "if pickles are purchased then ketchup is purchased'. Or

- If paper plates then plastic forks
- If dip then potato chips
- If salsa then tortilla chips

In order for the rules to be useful there are two pieces of information that must be supplied as well as the actual rule:

- Accuracy - How often is the rule correct?
- Coverage - How often does the rule apply?

Just because the pattern in the data base is expressed as rule does not mean that it is true all the time. Thus just like in other data mining algorithms it is important to recognize and make explicit the uncertainty in the rule. This is what the accuracy of the rule

means. The coverage of the rule has to do with how much of the database the rule "covers" or applies to. Examples of these two measure for a variety of rules is shown in Table 2.2.

In some cases accuracy is called the confidence of the rule and coverage is called the support. Accuracy and coverage appear to be the preferred ways of naming these two measurements.

Rule	Accuracy	Coverage
If breakfast cereal purchased then milk purchased.	85%	20%
If bread purchased then swiss cheese purchased.	15%	6%
If 42 years old and purchased pretzels and purchased dry roasted peanuts then beer will be purchased.	95%	0.01%

Table 2.2 Examples of Rule Accuracy and Coverage

The rules themselves consist of two halves. The left hand side is called the antecedent and the right hand side is called the consequent. The antecedent can consist of just one condition or multiple conditions which must all be true in order for the consequent to be true at the given accuracy. Generally the consequent is just a single condition (prediction of purchasing just one grocery store item) rather than multiple conditions. Thus rules such as: "if x and y then a and b and c".

What to do with a rule

When the rules are mined out of the database the rules can be used either for understanding better the business problems that the data reflects or for performing actual predictions against some predefined prediction target. Since there is both a left side and a right side to a rule (antecedent and consequent) they can be used in several ways for your business.

Target the antecedent. In this case all rules that have a certain value for the antecedent are gathered and displayed to the user. For instance a grocery store may request all rules that have nails, bolts or screws as the antecedent in order to try to understand whether discontinuing the sale of these low margin items will have any effect on other higher margin. For instance maybe people who buy nails also buy expensive hammers but wouldn't do so at the store if the nails were not available.

Target the consequent. In this case all rules that have a certain value for the consequent can be used to understand what is associated with the consequent and perhaps what affects the consequent. For instance it might be useful to know all of the interesting rules that have "coffee" in their consequent. These may well be the rules that affect the purchases of coffee and that a store owner may want to put close to the coffee in order to increase the sale of both items. Or it might be the rule that the coffee manufacturer uses to determine in which magazine to place their next coupons.

Target based on accuracy. Some times the most important thing for a user is the

accuracy of the rules that are being generated. Highly accurate rules of 80% or 90% imply strong relationships that can be exploited even if they have low coverage of the database and only occur a limited number of times. For instance a rule that only has 0.1% coverage but 95% can only be applied one time out of one thousand but it will very likely be correct. If this one time is highly profitable that it can be worthwhile. This, for instance, is how some of the most successful data mining applications work in the financial markets - looking for that limited amount of time where a very confident prediction can be made.

Target based on coverage. Some times user want to know what the most ubiquitous rules are or those rules that are most readily applicable. By looking at rules ranked by coverage they can quickly get a high level view of what is happening within their database most of the time.

Target based on "interestingness". Rules are interesting when they have high coverage and high accuracy and deviate from the norm. There have been many ways that rules have been ranked by some measure of interestingness so that the trade off between coverage and accuracy can be made.

Since rule induction systems are so often used for pattern discovery and unsupervised learning it is less easy to compare them. For example it is very easy for just about any rule induction system to generate all possible rules, it is, however, much more difficult to devise a way to present those rules (which could easily be in the hundreds of thousands) in a way that is most useful to the end user. When interesting rules are found they usually have been created to find relationships between many different predictor values in the database not just one well defined target of the prediction. For this reason it is often much more difficult to assign a measure of value to the rule aside from its interestingness. For instance it would be difficult to determine the monetary value of knowing that if people buy breakfast sausage they also buy eggs 60% of the time. For data mining systems that are more focused on prediction for things like customer attrition, targeted marketing response or risk it is much easier to measure the value of the system and compare it to other systems and other methods for solving the problem.

Caveat: Rules do not imply causality

It is important to recognize that even though the patterns produced from rule induction systems are delivered as if then rules they do not necessarily mean that the left hand side of the rule (the "if" part) causes the right hand side of the rule (the "then" part) to happen. Purchasing cheese does not cause the purchase of wine even though the rule if cheese then wine may be very strong.

This is particularly important to remember for rule induction systems because the results are presented as if this then that as many causal relationships are presented.

Types of databases used for rule induction

Typically rule induction is used on databases with either fields of high cardinality (many different values) or many columns of binary fields. The classical case of this is the super market basket data from store scanners that contains individual product names and

quantities and may contain tens of thousands of different items with different packaging that create hundreds of thousands of SKU identifiers (Stock Keeping Units).

Sometimes in these databases the concept of a record is not easily defined within the database - consider the typical Star Schema for many data warehouses that store the supermarket transactions as separate entries in the fact table. Where the columns in the fact table are some unique identifier of the shopping basket (so all items can be noted as being in the same shopping basket), the quantity, the time of purchase, whether the item was purchased with a special promotion (sale or coupon). Thus each item in the shopping basket has a different row in the fact table. This layout of the data is not typically the best for most data mining algorithms which would prefer to have the data structured as one row per shopping basket and each column to represent the presence or absence of a given item. This can be an expensive way to store the data, however, since the typical grocery store contains 60,000 SKUs or different items that could come across the checkout counter. This structure of the records can also create a very high dimensional space (60,000 binary dimensions) which would be unwieldy for many classical data mining algorithms like neural networks and decision trees. As we'll see several tricks are played to make this computationally feasible for the data mining algorithm while not requiring a massive reorganization of the database.

Discovery

The claim to fame of these ruled induction systems is much more so for knowledge discovers in unsupervised learning systems than it is for prediction. These systems provide both a very detailed view of the data where significant patterns that only occur a small portion of the time and only can be found when looking at the detail data as well as a broad overview of the data where some systems seek to deliver to the user an overall view of the patterns contained n the database. These systems thus display a nice combination of both micro and macro views:

- Macro Level - Patterns that cover many situations are provided to the user that can be used very often and with great confidence and can also be used to summarize the database.
- Micro Level - Strong rules that cover only a very few situations can still be retrieved by the system and proposed to the end user. These may be valuable if the situations that are covered are highly valuable (maybe they only apply to the most profitable customers) or represent a small but growing subpopulation which may indicate a market shift or the emergence of a new competitor (e.g. customers are only being lost in one particular area of the country where a new competitor is emerging).

Prediction

After the rules are created and their interestingness is measured there is also a call for performing prediction with the rules. Each rule by itself can perform prediction - the consequent is the target and the accuracy of the rule is the accuracy of the prediction. But because rule induction systems produce many rules for a given antecedent or

consequent there can be conflicting predictions with different accuracies. This is an opportunity for improving the overall performance of the systems by combining the rules. This can be done in a variety of ways by summing the accuracies as if they were weights or just by taking the prediction of the rule with the maximum accuracy.

Table 2.3 shows how a given consequent or antecedent can be part of many rules with different accuracies and coverages. From this example consider the prediction problem of trying to predict whether milk was purchased based solely on the other items that were in the shopping basket. If the shopping basket contained only bread then from the table we would guess that there was a 35% chance that milk was also purchased. If, however, bread and butter and eggs and cheese were purchased what would be the prediction for milk then? 65% chance of milk because the relationship between butter and milk is the greatest at 65%? Or would all of the other items in the basket increase even further the chance of milk being purchased to well beyond 65%? Determining how to combine evidence from multiple rules is a key part of the algorithms for using rules for prediction.

Antecedent	Consequent	Accuracy	Coverage
bagels	cream cheese	80%	5%
bagels	orange juice	40%	3%
bagels	coffee	40%	2%
bagels	eggs	25%	2%
bread	milk	35%	30%
butter	milk	65%	20%
eggs	milk	35%	15%
cheese	milk	40%	8%

Table 2.3 Accuracy and Coverage in Rule Antecedents and Consequents

The General Idea

The general idea of a rule classification system is that rules are created that show the relationship between events captured in your database. These rules can be simple with just one element in the antecedent or they might be more complicated with many column value pairs in the antecedent all joined together by a conjunction (item1 and item2 and item3 … must all occur for the antecedent to be true).

The rules are used to find interesting patterns in the database but they are also used at times for prediction. There are two main things that are important to understanding a rule:

Accuracy - Accuracy refers to the probability that if the antecedent is true that the precedent will be true. High accuracy means that this is a rule that is highly dependable.

Coverage - Coverage refers to the number of records in the database that the rule applies to. High coverage means that the rule can be used very often and also that it is less likely to be a spurious artifact of the sampling technique or idiosyncrasies of the database.

The business importance of accuracy and coverage

From a business perspective accurate rules are important because they imply that there is useful predictive information in the database that can be exploited - namely that there is something far from independent between the antecedent and the consequent. The lower the accuracy the closer the rule comes to just random guessing. If the accuracy is significantly below that of what would be expected from random guessing then the negation of the antecedent may well in fact be useful (for instance people who buy denture adhesive are much less likely to buy fresh corn on the cob than normal).

From a business perspective coverage implies how often you can use a useful rule. For instance you may have a rule that is 100% accurate but is only applicable in 1 out of every 100,000 shopping baskets. You can rearrange your shelf space to take advantage of this fact but it will not make you much money since the event is not very likely to happen. Table 2.4. Displays the trade off between coverage and accuracy.

	Accuracy Low	Accuracy High
Coverage High	Rule is rarely correct but can be used often.	Rule is often correct and can be used often.
Coverage Low	Rule is rarely correct and can be only rarely used.	Rule is often correct but can be only rarely used.

Table 2.4 *Rule coverage versus accuracy*.

Trading off accuracy and coverage is like betting at the track

An analogy between coverage and accuracy and making money is the following from betting on horses. Having a high accuracy rule with low coverage would be like owning a race horse that always won when he raced but could only race once a year. In betting, you could probably still make a lot of money on such a horse. In rule induction for retail stores it is unlikely that finding that one rule between mayonnaise, ice cream and sardines that seems to always be true will have much of an impact on your bottom line.

How to evaluate the rule

One way to look at accuracy and coverage is to see how they relate so some simple statistics and how they can be represented graphically. From statistics coverage is simply the a priori probability of the antecedent and the consequent occurring at the same time. The accuracy is just the probability of the consequent conditional on the precedent. So, for instance the if we were looking at the following database of super market basket scanner data we would need the following information in order to calculate the accuracy and coverage for a simple rule (let's say milk purchase implies eggs purchased).

T = 100 = Total number of shopping baskets in the database.

E = 30 = Number of baskets with eggs in them.

M = 40 = Number of baskets with milk in them.

B = 20 = Number of baskets with both eggs and milk in them.

Accuracy is then just the number of baskets with eggs and milk in them divided by the number of baskets with milk in them. In this case that would be 20/40 = 50%. The coverage would be the number of baskets with milk in them divided by the total number of baskets. This would be 40/100 = 40%. This can be seen graphically in Figure 2.5.

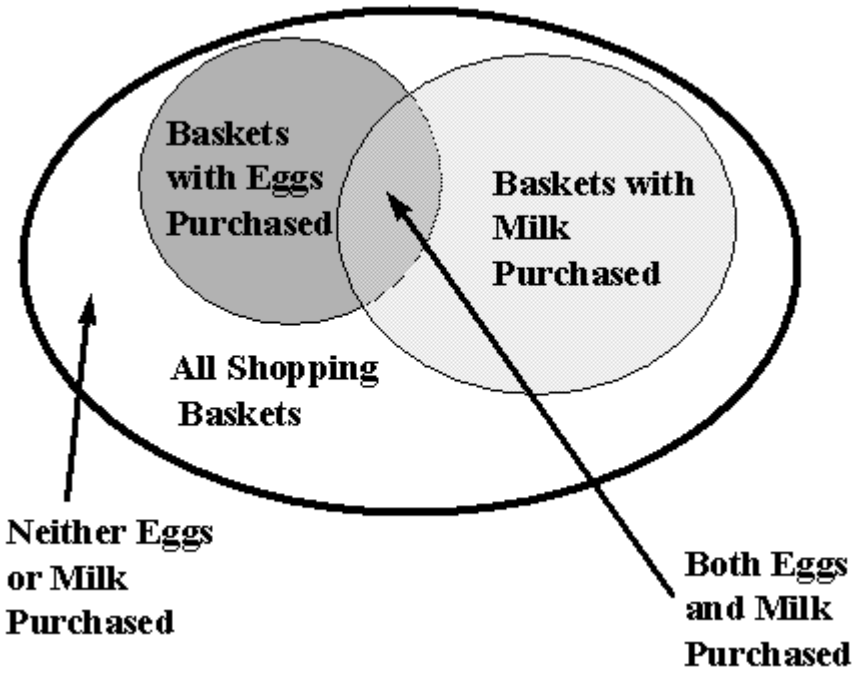


Figure 2.5 Graphically the total number of shopping baskets can be represented in a space and the number of baskets containing eggs or milk can be represented by the area of a circle. The coverage of the rule "If Milk then Eggs" is just the relative size of the circle corresponding to milk. The accuracy is the relative size of the overlap between the two to the circle representing milk purchased.

Notice that we haven't used E the number of baskets with eggs in these calculations. One way that eggs could be used would be to calculate the expected number of baskets with eggs and milk in them based on the independence of the events. This would give us some sense of how unlikely and how special the event is that 20% of the baskets have both eggs and milk in them. Remember from the statistics section that if two events are independent (have no effect on one another) that the product of their individual probabilities of occurrence should equal the probability of the occurrence of them both together.

If the purchase of eggs and milk were independent of each other one would expect that 0.3 x 0.4 = 0.12 or 12% of the time we would see shopping baskets with both eggs and milk in them. The fact that this combination of products occurs 20% of the time is out of

the ordinary if these events were independent. That is to say there is a good chance that the purchase of one effects the other and the degree to which this is the case could be calculated through statistical tests and hypothesis testing.

Defining "interestingness"

One of the biggest problems with rule induction systems is the sometimes overwhelming number of rules that are produced. Most of which have no practical value or interest. Some of the rules are so inaccurate that they cannot be used, some have so little coverage that though they are interesting they have little applicability, and finally many of the rules capture patterns and information that the user is already familiar with. To combat this problem researchers have sought to measure the usefulness or interestingness of rules.

Certainly any measure of interestingness would have something to do with accuracy and coverage. We might also expect it to have at least the following four basic behaviors:

- Interestingness = 0 if the accuracy of the rule is equal to the background accuracy (a priori probability of the consequent). The example in Table 2.5 shows an example of this. Where a rule for attrition is no better than just guessing the overall rate of attrition.
- Interestingness increases as accuracy increases (or decreases with decreasing accuracy) if the coverage is fixed.
- Interestingness increases or decreases with coverage if accuracy stays fixed
- Interestingness decreases with coverage for a fixed number of correct responses (remember accuracy equals the number of correct responses divided by the coverage).

Antecedent	Consequent	Accuracy	Coverage
<no constraints>	then customer will attrite	10%	100%
If customer balance > $3,000	then customer will attrite	10%	60%
If customer eyes = blue	then customer will attrite	10%	30%
If customer social security number = 144 30 8217	then customer will attrite	100%	0.000001%

Table 2.5 *Uninteresting rules*

There are a variety of measures of interestingness that are used that have these general characteristics. They are used for pruning back the total possible number of rules that might be generated and then presented to the user.

Other measures of usefulness

Another important measure is that of simplicity of the rule. This is an important solely for

the end user. As complex rules, as powerful and as interesting as they might be, may be difficult to understand or to confirm via intuition. Thus the user has a desire to see simpler rules and consequently this desire can be manifest directly in the rules that are chosen and supplied automatically to the user.

Finally a measure of novelty is also required both during the creation of the rules - so that rules that are redundant but strong are less favored to be searched than rules that may not be as strong but cover important examples that are not covered by other strong rules. For instance there may be few historical records to provide rules on a little sold grocery item (e.g. mint jelly) and they may have low accuracy but since there are so few possible rules even though they are not interesting they will be "novel" and should be retained and presented to the user for that reason alone.

Rules vs. Decision trees

Decision trees also produce rules but in a very different way than rule induction systems. The main difference between the rules that are produced by decision trees and rule induction systems is as follows:

Decision trees produce rules that are mutually exclusive and collectively exhaustive with respect to the training database while rule induction systems produce rules that are not mutually exclusive and might be collectively exhaustive.

In plain English this means that for an given record there will be a rule to cover it and there will only be one rule for rules that come from decision trees. There may be many rules that match a given record from a rule induction system and for many systems it is not guaranteed that a rule will exist for each and every possible record that might be encountered (though most systems do create very general default rules to capture these records).

The reason for this difference is the way in which the two algorithms operate. Rule induction seeks to go from the bottom up and collect all possible patterns that are interesting and then later use those patterns for some prediction target. Decisions trees on the other hand work from a prediction target downward in what is known as a "greedy" search. Looking for the best possible split on the next step (i.e. greedily picking the best one without looking any further than the next step). Though the greedy algorithm can make choices at the higher levels of the tree which are less than optimal at the lower levels of the tree it is very good at effectively squeezing out any correlations between predictors and the prediction. Rule induction systems on the other hand retain all possible patterns even if they are redundant or do not aid in predictive accuracy.

For instance, consider that in a rule induction system that if there were two columns of data that were highly correlated (or in fact just simple transformations of each other) they would result in two rules whereas in a decision tree one predictor would be chosen and then since the second one was redundant it would not be chosen again. An example might be the two predictors annual charges and average monthly charges (average monthly charges being the annual charges divided by 12). If the amount charged was predictive then the decision tree would choose one of the predictors and use it for a split point somewhere in the tree. The decision tree effectively "squeezed" the predictive

value out of the predictor and then moved onto the next. A rule induction system would on the other hand create two rules. Perhaps something like:

If annual charges > 12,000 then default = true 90% accuracy

If average monthly charges > 1,000 the default = true 90% accuracy.

In this case we've shown an extreme case where two predictors were exactly the same, but there can also be less extreme cases. For instance height might be used rather than shoe size in the decision tree whereas in a rule induction system both would be presented as rules.

Neither one technique or the other is necessarily better though having a variety of rules and predictors helps with the prediction when there are missing values. For instance if the decision tree did choose height as a split point but that predictor was not captured in the record (a null value) but shoe size was the rule induction system would still have a matching rule to capture this record. Decision trees do have ways of overcoming this difficulty by keeping "surrogates" at each split point that work almost as well at splitting the data as does the chosen predictor. In this case shoe size might have been kept as a surrogate for height at this particular branch of the tree.

Another commonality between decision trees and rule induction systems

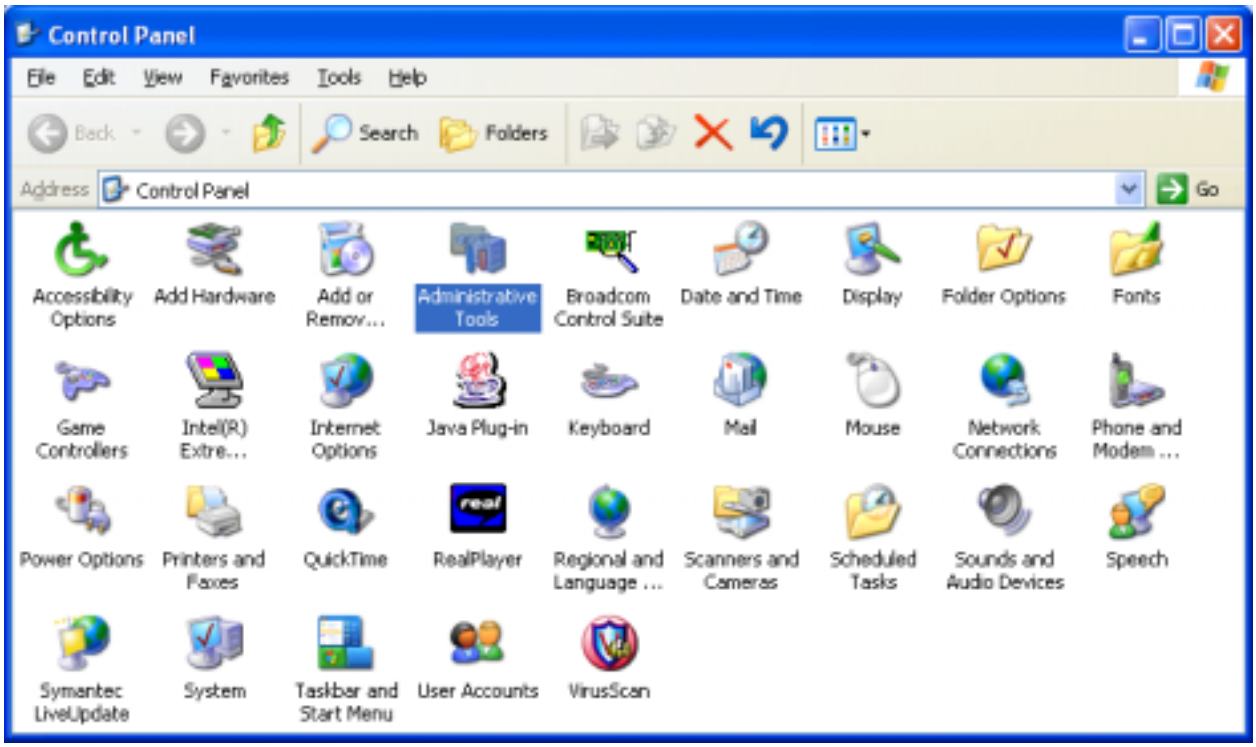
One other thing that decision trees and rule induction systems have in common is the fact that they both need to find ways to combine and simplify rules. In a decision tree this can be as simple as recognizing that if a lower split on a predictor is more constrained than a split on the same predictor further up in the tree that both don't need to be provided to the user but only the more restrictive one. For instance if the first split of the tree is age <= 50 years and the lowest split for the given leaf is age <= 30 years then only the latter constraint needs to be captured in the rule for that leaf.

Rules from rule induction systems are generally created by taking a simple high level rule and adding new constraints to it until the coverage gets so small as to not be meaningful. This means that the rules actually have families or what is called "cones of specialization" where one more general rule can be the parent of many more specialized rules. These cones then can be presented to the user as high level views of the families of rules and can be viewed in a hierarchical manner to aid in understanding.

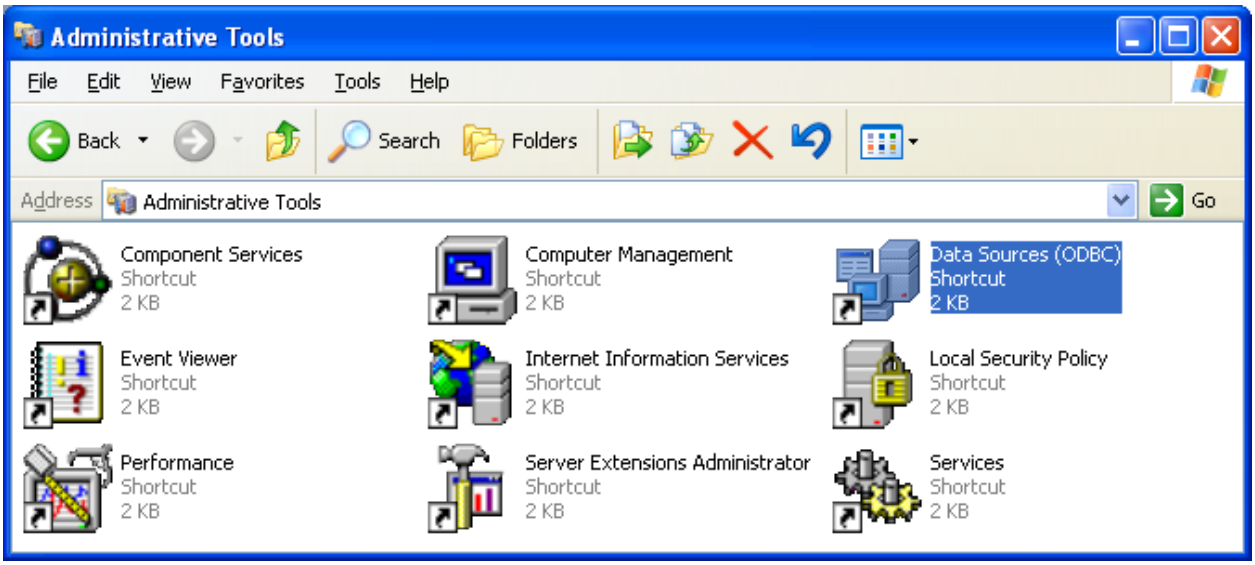
Appendix B: Setting Up an ODBC Data Source

In this appendix, we are going to go through the process of setting up an ODBC data source.

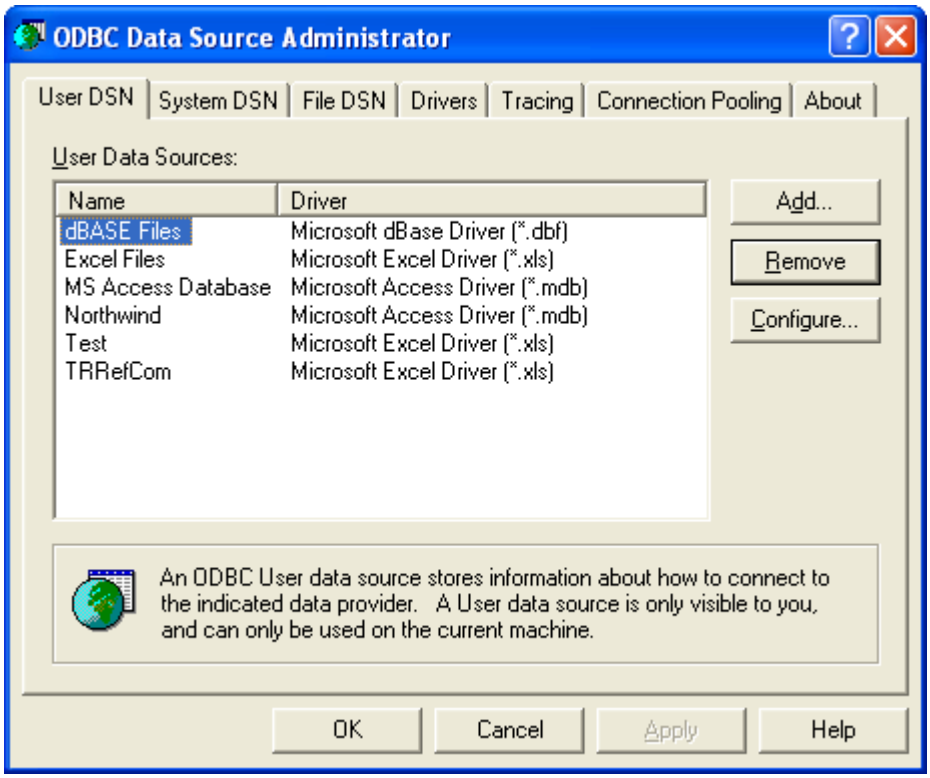
Click on the 'Control Panel' menu option on the Start Menu. The Control Panel dialog will appear.



Double-click on the 'Administrative Tools' icon. The Administrative Tools dialog will appear.



Double-click on the 'Data Sources (ODBC)' icon. The ODBC Data Source Administrator dialog will appear.

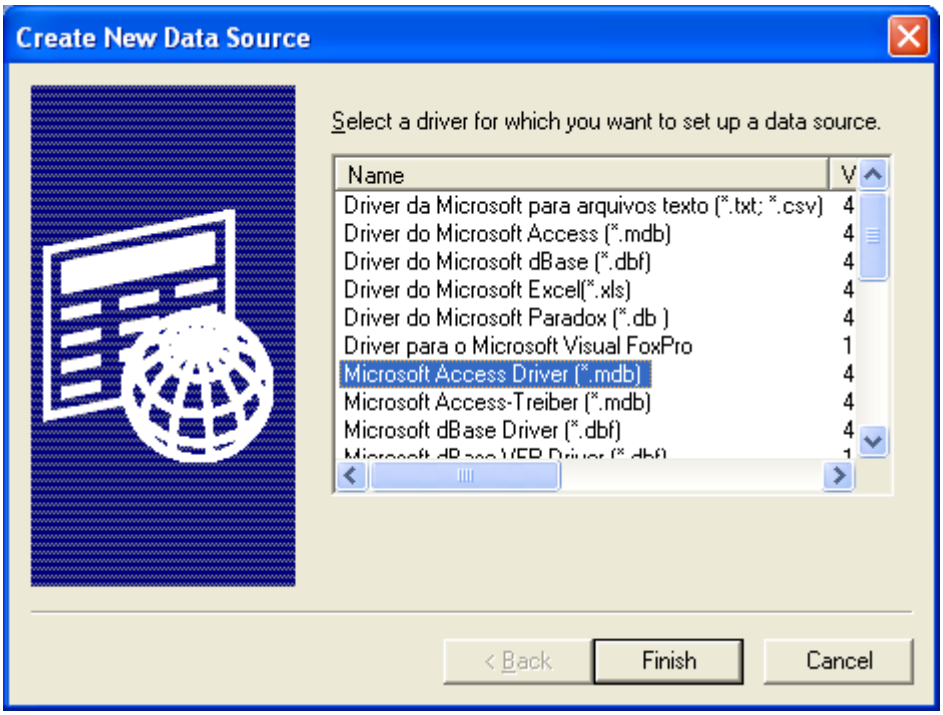


The 'User DSN' tab allows you to set up a database files as an ODBC data source for your own use.

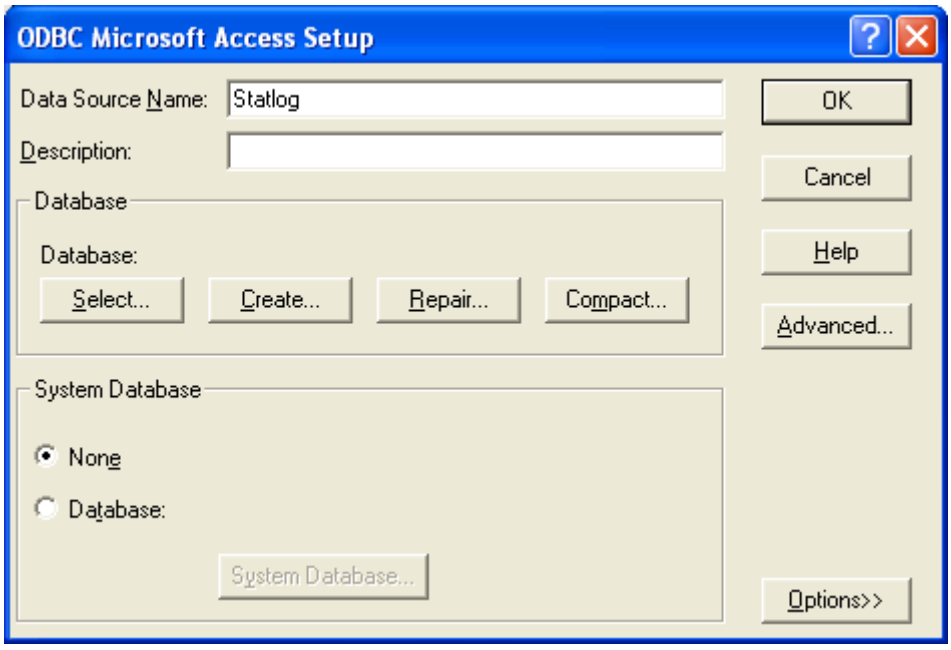
The 'System DSN' tab allows you to set up a database files as an ODBC data source for other users of your computer to use. This is the one to use if your ODBC data source is to be accessed via a ProWeb application.

Click on the 'Add' button to add a new ODBC data source. The Create New Data Source

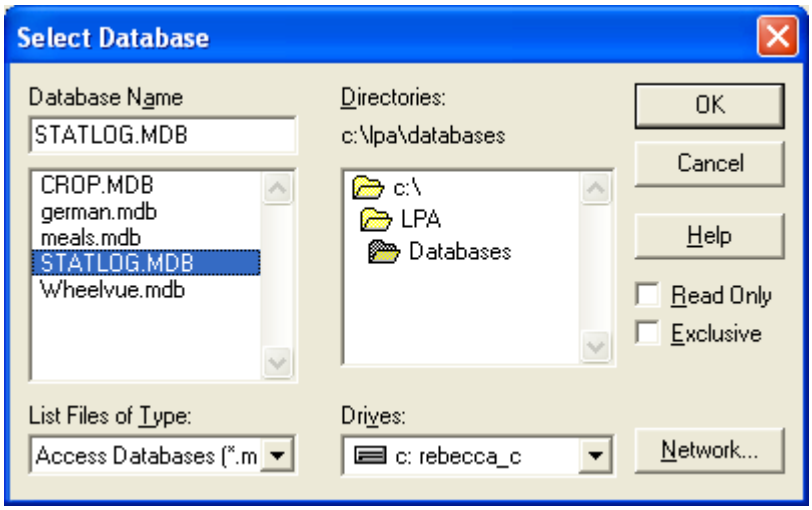
dialog will appear.



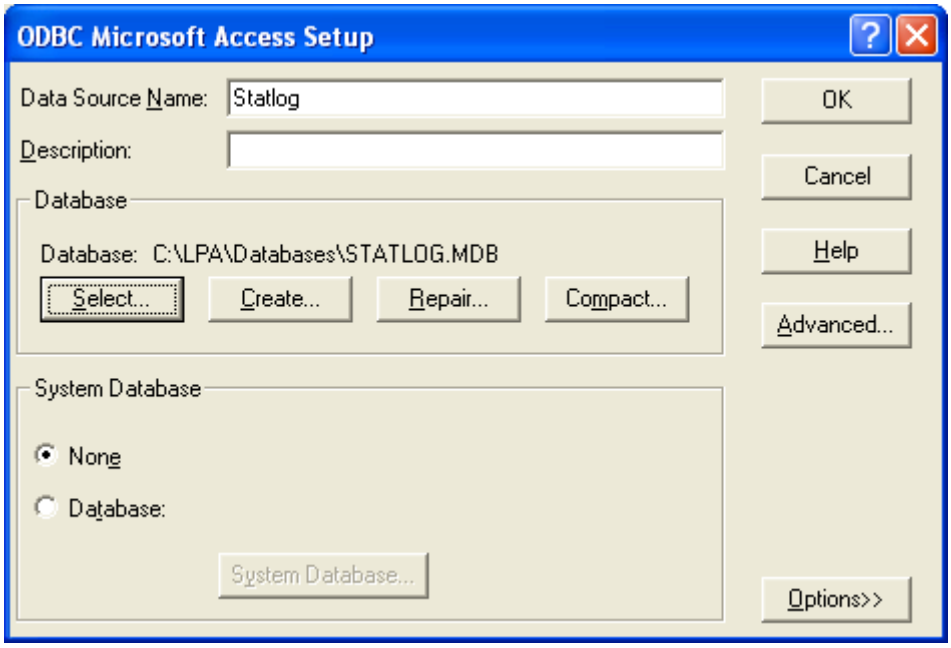
Select the driver for your data source; in the case of a Microsoft Access .MDB file, select 'Microsoft Access Driver (*.mdb)' and then click on Finish. The ODBC Microsoft Access Setup dialog will appear.



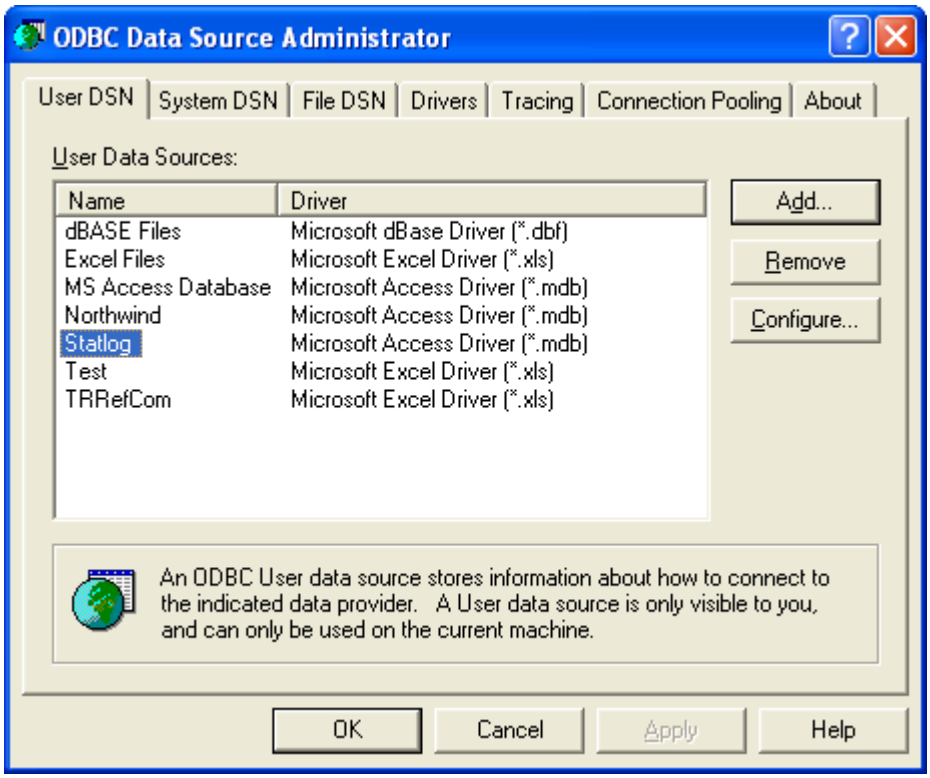
Type in the name for the ODBC data source in the 'Data Source Name:' field. We are going to add 'Statlog' as our new ODBC data source. Click on the 'Select…' button; the Select Database dialog will appear.



Navigate to where your chosen database file is and select it. STATLOG.MDB will be in a different location on your computer. Click on OK when done. The ODBC Microsoft Access Setup dialog will reappear.



Click on OK. The ODBC Data Source Administrator dialog will reappear; the ODBC data source you have just added (i.e. Statlog) will appear in the list.


ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
User Data Sources:
Name
Driver
dBASE Files
Microsoft dBase Driver (*.dbf)
Excel Files
Microsoft Excel Driver (*.xls)
MS Access Database
Microsoft Access Driver (*.mdb)
Northwind
Microsoft Access Driver (*.mdb)
Statlog
Microsoft Access Driver (*.mdb)
Test
Microsoft Excel Driver (*.xls)
TRRefCom
Microsoft Excel Driver (*.xls)
Add...
Remove
Configure...
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.
OK
Cancel
Apply
Help