Darwin Reference

Release 3.0.1

Thinking Machines Corporation

First printing, May 1998

The information in this document is subject to change without notice and should not be construed as a commitment by Thinking Machines Corporation. Thinking Machines reserves the right to make changes to any product described herein.

Although the information in this document has been reviewed and is believed to be reliable, Thinking Machines Corporation assumes no liability for errors in this document. Thinking Machines does not assume any liability arising from the application or use of any information or product described herein.

Thinking Machines Corporation
16 New England Executive Park
Burlington, Massachusetts 01803
Tel: 781-238-3400 • Fax: 781-238-3420

Contents

About This Manual

Objectives

Darwin Reference describes and explains the Darwin data mining product. It introduces data mining concepts, describes the tools Darwin uses, and explains the process of data mining with Darwin, from formulating the initial business question through working with the data to developing models and analyzing results.

Its companion volume, *Using Darwin*, describes the Darwin user interface and explains how to use it.

Intended Audience

The two books are intended for all users of Darwin software. They assume some familiarity with data mining and basic statistical techniques.

Revision Information

Using Darwin and *Darwin Reference* are new manuals. They are derived from the *Darwin User's Guide,* which accompanied all earlier releases of Darwin. That volume contains both background and theoretical material about data mining and Darwin as well as detailed description of the Darwin user interface and explanations of how to use it. With this release, the two topics are treated in separate manuals: background and theoretical material is contained in *Darwin Reference*, and the mechanics of the user interface are contained in *Using Darwin*.

Organization

Darwin Reference is organized as follows:

Chapter 1 Introduction to Darwin
Overview of the Darwin data mining process.

Chapter 2 Defining the Business Problem
Discusses the first steps in a data mining process, i.e., asking the

right questions and locating and preparing the relevant data. Includes information on processes and terminology used to identify items of data to Darwin (and to data-handling software in general).

Chapter 3 Preparing the Data
Discusses how to identify, clean, and organize data for use in various situations.

Chapter 4 Creating and Handling Datasets
Discusses how to access and manipulate data within Darwin by creating datasets for Darwin's model-building tools.

Chapter 5 Transformation Datasets
Describes the various modifications of datasets you can perform.

Chapter 6 Introducing Darwin Models
Introduces the three types of models Darwin creates: trees, neural nets, and match models. Gives an overview of the process of building and using models.

Chapter 7 Darwin Tree
Introduces tree models and describes how to use Darwin to train, test, and use tree models for classification, prediction, and forecasting.

Chapter 8 Darwin Net
Introduces neural net models and describes how to use Darwin to create, train, test, and use neural net models for classification, prediction, and forecasting.

Chapter 9 Darwin Match
Introduces the *k*-nearest-neighbors algorithm and memory-based reasoning, and describes how to use Darwin to create, train, test, and use match models for classification and prediction.

Chapter 10 Analyzing Results
Describes the Darwin tools for analyzing and visualizing data and for evaluating Darwin predictions.

Chapter 11 Generating Model Code
Describes how to export Darwin tree and net models.

Appendixes

Appendix A Data Design
Explains data design using examples of data choice and preparation for applications involving retail sales and marketing.

Appendix B Glossary
Lists terminology relevant to Darwin and to data mining.

Hardcopy Documentation

The complete Darwin documentation set includes

- *Darwin Reference* (this volume). Introduces data mining and Darwin, provides background and theoretical material on datasets, Darwin tools. and analyses.
- *Using Darwin* (companion volume to *Darwin Reference*). Describes the user interface and provides detailed instructions for using it.
- *Darwin Release Notes, Release 3.0.1.* Describes the release and documents any problems or bugs in the software. There are separate release notes for Solaris and for HP-UX.
- For Solaris system administrators: *Darwin Installation and Administration, Release 3.0.1 for Solaris.*
- For HP-UX system administrators: *Darwin Installation and Administration, Release 3.0.1 for HP-UX.*

Online Help

Darwin includes extensive online help that can be summoned from a list of contents and from Help buttons or the F1 key on dialog windows.

Online Documentation

All the Darwin documentation is available in HTML format at a password-protected site for customers only, accessible from our Web site at `http://www.think.com`. You can get the password from your system administrator.

To view these files, you will need

- Netscape 2.x or later, or
- Internet Explorer 3.01 or later

Notation Conventions

The table below displays the notation conventions observed in this manual.

Convention	Meaning
Boldface	Darwin commands, menu names, and menu items.
Project –> New File	Indicates the path for a command, e.g., on the **Project** menu, click the **New File** command.
`typewriter`	Data fields and values, special characters, etc., examples of files, data, and so on.
`italics`	Argument names and placeholders in command formats.
`% user input` `system output`	In interactive examples, user input is shown in bold typewriter; system output is shown in regular typewriter

Customer Support

Thinking Machines Customer Support encourages customers to report problems with Darwin and to suggest improvements in our products.

Customer support is available Monday through Friday from 9 a.m. to 5 p.m., Eastern Standard Time.

When reporting a problem, please provide as much information as possible to help us identify it. Please record and pass along as much of the following information as possible:

- The dataset that you used and its characteristics (size, number of fields and records, etc.)
- How the dataset was created (flat file, ORACLE, INFORMIX, versions, etc.)
- The type of model(s) that you used
- Where you were in Darwin (the folder you were in and the menu you were using)
- The three or four steps you performed just prior to the error
- Any error messages generated by either Darwin or the operating system

Because much of this information is collected automatically in a Darwin macro (command log), we recommend that you turn on command logging using the **Macro** selection of the **Options** menu and forward the log to Thinking Machines when you report a bug.

You should also record and communicate the following information:

- The hardware platform
- The release of UNIX that you are using
- The hardware platform that the client is running on
- The version of Windows NT or Windows 95 that you are running
- The mode you were operating in (serial or parallel)
- If parallel, the number of processors you were running on

Please contact Thinking Machines' home office customer support staff:

Internet Electronic Mail:	`customer-support@think.com`
U.S. Mail:	Thinking Machines Corporation Customer Support 16 New England Executive Park Burlington, Massachusetts 01803
Telephone:	
Within North America	1-800-677-1110 Monday – Friday, 9 a.m. – 5 p.m. EST
Outside North America	+1-781-238-3400 Monday – Friday, 9 a.m. – 5 p.m. EST
Fax:	1-781-238-3420

1 Introducing Darwin

Too much data and not enough information — this is a problem facing many businesses and industries.

A solution lies here, with data mining — classification, forecasting, predictive analysis, decision trees — different approaches to the same problem: how to use all that data, data that's now clogging your disks and tapes, to run your business more profitably.

"More profitably" can mean many things:

- increased sales
- decreased costs
- greater productivity
- more precise targeting
- increased opportunities

Most businesses today have a great deal of data, with a great deal of useful information hiding within it. Unfortunately, "hiding" is usually exactly what it's doing: So much data exists that it overwhelms traditional methods of data analysis. Data mining tools provide a means of getting at the information buried in the data.

1.1 Introducing Darwin

Darwin's data mining tools are designed to find hidden patterns in large, complex collections of data. Darwin offers proven methods for forecasting, classifying, and predicting, as well as tools for analyzing data and results. The overall process is this:

- plan a project to tackle a particular business problem
- locate and organize data that bears on the problem
- use Darwin tools to create *models* from your data
- use the model to gain information

- take action according to what you learn
- evaluate results
- iterate

1.2 Prediction, Classification, Regression, and Forecasting

Data mining applications like Darwin talk a lot about classification, prediction, and forecasting, and we'll be using those terms throughout this manual. What exactly do they mean, in the data mining context?

1.2.1 Prediction

Prediction is the most general of the terms. It refers to using information gained from some set of known values to predict other values.

Data mining makes predictions by using one of two methods: *classification* or *regression*. The task of predicting values via regression is frequently called *forecasting*.

As an introduction to classification and regression, let's look at some simple examples.

1.2.2 Classification

When you classify something, you divide a population into several discrete groups or categories, placing each member of the population in one of those categories.

Here is a small, simple problem to illustrate: Given the data below on height, weight, and age,

- Classify each individual as *male* or *female.*
- Characterize your level of confidence in your classification as being *high* (H), *moderate* (M), *low* (L), or *nonexistent* (N).

ID	Height	Weight	Age	Gender	Confidence
A	6′ 5″	325	27	?	?
B	5′ 3″	110	38	?	?
C	4′ 1″	65	10	?	?
D	1′ 3″	8	0.5	?	?

How did you make your decisions? You made them on the basis of what you know about human heights and weights. For example, you know that most people you have seen (or read about) who were 6′ 5″ tall and weighed 325 pounds have been males. Therefore, you assume that an unknown person of that height and weight probably falls into the same classification, and so is also male.

Note that past knowledge tells you not only which classifications you can make, but which ones you cannot make, on the basis of given data. Almost everyone who sees this problem recognizes that there is no way you can classify an infant's gender, based simply on height and weight, with any greater accuracy than that obtained by flipping a coin.

The classification table above might be filled in as shown below, reflecting the moderate degree of confidence that someone 5'3" and 110 pounds is female, and no confidence whatsoever in classifying the ten-year-old and the infant:

ID	Height	Weight	Age	Gender	Confidence
A	6′ 5″	325	27	male	H
B	5′ 3″	110	38	female	M
C	4′ 1″	65	10	female	N
D	1′ 3″	8	0.5	female	N

1.2.3 Prediction Using Classification

In this example, you used classification to attempt to discover particular features about individuals. Given records containing certain data values, you attempted to determine the most likely value for an empty field (gender) in the record. At the same time, you determined the *level of confidence* you could have in your prediction.

You can use the same data, and the same techniques, not just to fill in missing bits of knowledge, but, what is usually more interesting, to predict future behaviors for your four individuals.

For instance, you might ask, which of these four people is most likely to need a professional football uniform? Or, assuming each individual comes from a different household, which households are the best candidates for catalogs for children's toys and clothing?

For this second question, note that all four households could contain children; you know for certain only that two of them do. Whether a direct mailing should go to two or to four households is a business decision, based on the degree of certainty you think you need in order to take a particular action.

1.2.4 Prediction Using Forecasting

Forecasting is a type of prediction that works with numbers and numerical distances to provide particular, rather than categorical, answers. In technical terminology, forecasts predict values within a continuous series of values. Forecasts, therefore, can predict values that have not yet appeared in the data, while classification is limited to classes that have already appeared in earlier data.

You can think of classifying a record as choosing which bin, among several bins, to put the record into; of forecasting as figuring out where on a line of values the record should go.

The way you phrase your question reflects an assumption of classification or regression, as illustrated in these examples:

- **Forecasting:** What will be tomorrow's high temperature?
- **Classification:** Will the temperature be over 68° tomorrow?

- **Forecasting:** How many widgets will we sell in each of our stores next quarter?
- **Classification:** Which stores will show high sales of widgets next quarter?

- **Forecasting:** What will be tomorrow's high price for XYZ stock?
- **Classification:** Will XYZ stock raise, fall, or stay steady tomorrow?

1.3 Using Darwin to Solve Problems

There are several steps to the data mining process. Using Darwin, the process looks like this:

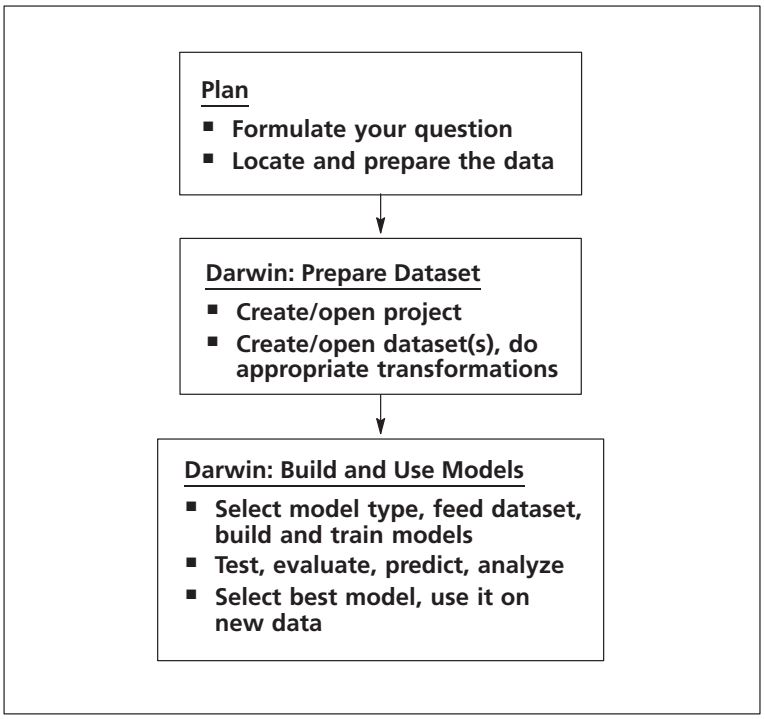


Figure 7. Darwin's data mining process.

Step 1. Decide on the Right Question

The very first step is to identify a business question you want to answer. Formulating the right question is extremely important, and can take a lot of time and effort. Here are some examples:

- How much of items *x*, *y*, and *z* should we order for next quarter's stock?
- How can we best identify which credit card candidates will be good credit risks?

- How can we reduce losses caused by equipment failure?
- How can we best target the next product introduction?

Step 2. Locate and Prepare the Data

Once you have decided on your business question, you locate data that is potentially relevant to that question. Some examples:

- a company's billing records for the last three fiscal years
- credit records for a group of people, from initiation of the records to the present
- data on equipment failures for the last year for a company, plant, or department

This *historical* data that you "mine" for information helps you make decisions about current and future actions. This is the data that Darwin uses to *train* its tools, providing them with information to build the models that are then used to answer your business questions. It's very important that this data contain the right variables for your question. You may have to manipulate or enrich the existing data.

These first two steps constitute the very important planning phase of the data mining process, a phase that occurs before you start Darwin. See chapter 2, "Defining the Business Problem," and chapter 3, "Data Preparation," for guidelines on this phase. See also appendix A, "Data Design."

Step 3. Create a Darwin Project

Once you have formulated your question and have prepared the relevant data, you are ready to start using Darwin. The Darwin user interface and detailed instructions for using it are described in *Using Darwin*.

Your first step within Darwin is to create a *project*, which will contain a given set of files and datasets. This is where you will keep all the bookkeeping and record-keeping related to that project. For example, the formulation of your business question can be stored there, a statement about the project's objective, as well as information about who is working on the project, a list of associated data files, plus any notes you may wish to make about the files or the tools you have used, etc.

Step 4. Create a Darwin Dataset

From the data that you prepared in step 2, you create a Darwin *dataset.* Basically, datasets are collections of data records that Darwin uses for learning and prediction.

Each dataset record contains a number of fields. For data mining purposes, the fields are of two types:

- *Independent variables*, the data fields that will be used for their predictive value. Most of the fields in your records are independent variables.
- *Target variables* (also known as *response* or *dependent variables*, and, in data mining, as *target fields),* the field(s) that hold the item(s) to be predicted. Typically there are a few target variables, often only one or two. (In the classification example in section 1.2, the target variable was *gender*.)

Note that the dataset itself does not define which field contains the target variable. You decide what the target variable is, based on the business question you are asking. In building your *model,* you indicate which field or fields is/are the target field(s).

The planning information provided in chapter 2, "Defining the Business Problem," and in appendix A, "Data Design," can help you decide what fields your data records should contain.

You can create Darwin datasets from data residing in flat files (text or ASCII files) or in databases. These topics are discussed in chapter 4, "Creating and Handling Datasets," and chapter 5, "Transformation Datasets."

Step 5. Choose and Build Model(s)

Once your dataset(s) are ready, you decide which type of model you want to build — tree, net, or match model — to provide answers to your question. You will probably build more than one model, and more than one type of model. See section 6.2 for help selecting a model type.

Darwin can build three different types of models:

- The Darwin **Tree** model is based on the classification and regression tree (C&RT) algorithm.
- The Darwin **Net** model is a feed-forward multilayer neural network.
- The Darwin **Match** model**,** a memory-based reasoning model, uses a *k*-nearest neighbor algorithm.

All three models can use categorical data, either binary or multivalued, and continuous (ordered) data, and all and support parallelism.

The Darwin models are described in chapters 6–9.

Step 6. Test, Evaluate, Predict, Analyze Models

We said earlier that the historical dataset is used to build and train a model. What does this mean? Basically the process is this: As a preliminary step, you typically divide the historical dataset into subsets to be used for training, testing, and prediction.

You then create and train the model using the information in two or more of the subset datasets. You then use the model to predict the target variable in the prediction subset dataset. The prediction subset of course already has actual values in the target field; you compare predicted values with the actual target values and thus get an idea of how accurate your model's predictions are.

This process is likely to be iterative; you and Darwin keep adjusting various parameters in order to improve the model, i.e., increase the accuracy of its predictions.

Chapter 6 introduces the model types and gives an overview of the process of building and using models. Basic information about the models is given in chapter 7 for trees, chapter 8 for nets, and chapter 9 for match models.

Step 7. Use the Model on New Data

Depending on your business question, you may either make decisions directly from the information your model gives you, or you may apply your model to new data and make decisions concerning that new data. For instance, studying a decision tree created from historical data might give you guidelines on how to perform future actions.

New data of course has no values for its target variable. Generally, you apply Darwin's predictive models to new data to determine how to act on it. For example,

- Given a new mailing list, the model can predict what catalogs you should send to which list members to maximize your return on the mailing.
- Given a new applicant for credit, the model can forecast how much credit, if any, should you grant.

Step 8. Verify that the Solution Worked

You began your data mining process with a question or a problem. Your model presumably offers an answer to the question, a solution to the problem. Now you want to find out whether the proposed solution actually works.

Depending on exactly what the solution is, you may want to try it out on a subset of customers for a trial period before you implement it completely.

Step 9. For Ongoing Applications, Periodically Update Your Model

If your application deals with an ongoing business question, you may want to periodically update (and perhaps refine) your model. For instance, you might use a predictive model, based on data through 1997, to decide who among the 1998 credit applicants were likely to be good risks. At the end of 1998, you would add the credit histories of the accepted applicants to the model and use the updated model to make decisions about 1999's applicants.

1.4 Who Does What?

The steps listed above can be divided among various people (or even various groups working at different locations) according to expertise. Thus, one person or group could define a business problem, another could prepare the data, a third could build the models, and others could use the completed model to make predictions and decisions on newly arriving data.

1.5 Measuring Success

The success of predictive models in the business world is measured not only by how accurate the models are, but by the amount of profit they create. For example:

- Company X reduces bad credit losses by *n*%. Depending on the amount of credit Company X offers, the reduction could represent hundreds, thousands, or millions of dollars annually, even if *n* is a small number.

- Company Y uses a predictive model to target a direct mailing more precisely. By so doing, it spends only 40% of what it would ordinarily spend on the mailing, but generates 90% of its usual sales. The company has thus increased its return on investment for this particular mailing substantially. Again, this could represent substantial savings, depending on the company's costs and sales volumes.

2 Defining the Business Problem

The first step is to set out the problem and its context. You must

- identify the problem
- quantify your potential costs and rewards
- identify your data
- make sure your data is properly organized

This chapter and the next provide some suggestions for doing so. See also appendix A, "Data Design," for additional suggestions.

2.1 Defining the Problem

Begin by defining the business problem on which you want to work, together with the conditions under which you need it solved. Ensuring clarity at this stage helps you set the direction for selecting and preparing your data, for designing the analysis, and for judging the quality and utility of your results. As a starting point, consider the questions below. These questions are intended to help you clarify your circumstances and to answer the key question, "What will be the pay-off for solving this problem?"

- What business problem do I want to solve?
- What data do I have that might help find the solution?
- Will the data need enrichment or other modification? (See section 3.3, "Cleaning Data.")
- When do I need the solution?
- What resources can I use?
- What costs (actual costs or lost profits) do I incur by not solving it?
- Do I have time to solve it? How long will it take to make the savings?

- What is the minimum level of improvement I need from the analysis, and how will I measure the improvement?
- What costs might I incur as a result of incorrect analysis? In particular,
  - How much cost will each false positive incur?
  - How much cost will each false negative incur?

2.1.1 Costs and Benefits

Potential costs and benefits are important to business decisions. The key questions in deciding whether to perform a data mining project are these:

- What will be the return on investment (ROI) for the minimum level of accuracy you can accept?
- How will the ROI change with greater accuracy? (For every percentage increase in accuracy, what is the return?)

More precisely phrased considerations may be factored into prediction models as well. For this purpose, the basic cost considerations are set out as follows:

- How much benefit will be gained from a correct positive prediction? For example, how much revenue are you likely to gain if you send a direct mail piece to someone who responds to it?
- How much loss will an incorrect positive prediction (a *false positive*) incur? For example, how much will it cost you to send a mailing to someone who does not respond?
- How much benefit will be gained from a correct negative prediction? For example, how much will you save by not sending mail to someone who will not respond to it?
- How much cost will an incorrect negative prediction (a *false negative*) incur? For example, what potential revenue do you lose by not sending the mailing to someone who would respond?

Each data mining tool has its own method of allowing you to specify how (and whether) costs should be factored into the classification process. These methods are described in the discussions of the tools themselves.

2.1.2 The Time/Perfection Tradeoff

Data mining is an iterative process: You try something, look at your results, see something that might make an improvement, modify the model or try a different model, and so on. Or, you initially make several models and test them against each other, learning how to improve them in the process.

Identifying your time and resource constraints (and your expectations) before you start the modeling process helps you decide how to make any necessary tradeoffs between getting results "today" and getting the "best possible results."

2.2 Defining Your Question

Defining your question starts with defining the problem in business terms, as shown above. The second step is to pose the question in the *datacentric* terms demanded by data mining (or, indeed, by any data analysis.) To do this, you cast the problem in terms of *data fields* and *data records*.

2.2.1 Terminology: Fields and Records

Let's return for a moment to the issue of classification, first mentioned in section 1.2. Most people deal with at least one classification task every day, when they sort the day's mail and decide how to handle it: Read it now? Read it later? Answer it? Throw it away?

You could, if you wished, formalize this process as shown in the following "business form":

```
_____________________________________________________

ITEM #_____  ITEM TYPE _______   DATE_______
SENT BY ____________________

READ                RESPOND         DISPOSITION
[ ] Read Now           [ ] Yes      [ ] File
[ ] Read Later         [ ] No       [ ] Throw Away
[ ] Skim Quickly
[ ] Not worth reading

_____________________________________________________
```

Completed forms could, in turn, become data records:

```
ITEM      TYPE    SENDER    DATE      READ  RESPOND  SAVE
1016      03      J.Jones   970302    2     0        0
1017      11      1stNatl   970302    1     0        1
1018      05      Mother    970302    1     1        0
```

Each record represents one piece of mail. READ, RESPOND, etc., are now *fields* within the records, each holding one item of information about the piece of mail or about how you handled it.

Within each record, each field has a given *value*. The fields are constant from record to record. Their values vary. In fact, fields are frequently called *variables* in discussions of data analysis.

The RESPOND and SAVE fields have only two values each; such fields are called *binary* (or *yes/no*) fields. The READ, SENDER, and TYPE fields offer multiple choices, and are called *multiclass* fields. All five are called *categorical* fields, because they define the items they describe as belonging to some category.

The ITEM and DATE fields (as formatted above) are *ordered* or *continuous* fields; they can contain any number of values, but each value has a measurable distance from each other value.

Categorical and Ordered Fields

Not all fields are obviously either categorical or ordered. Some can be treated as either. Consider three fields you would probably find within records for T-shirt sales: NUMBER_SOLD, COLOR, and SIZE:

- NUMBER_SOLD is almost always treated as an ordered field. Numbers such as 1, 2, and 3 exhibit a fixed, measurable, ordering, whether we are talking about one or two shirts, one or two dozen shirts, or one or two truckloads of shirts.
- COLOR is almost always treated as a categorical field. There are usually a manageable number of categories (that is, colors), and there is no real sense of order or distance among values such as *blue*, *red*, and *yellow*.
- SIZE can be either ordered or categorical. Values such as *small, medium, large,* and *extra large* can be considered as four separate categories, or as four points on an ordered scale from smallest to largest.

Note: Within Darwin, you define a field as either categorical or ordered when you create the descriptor file for the dataset (as explained in section 4.5). Darwin also provides a command (**Set Form**, one of the **Dataset** menu's **Transformation** commands) that allows you to change the form of a field in an existing dataset from categorical to ordered or vice versa. Thus, you can change the way you treat a field according to the needs of your problem. Discussions of particular operations throughout this manual will point out instances in which you might want to do so.

2.2.2 Predictive Fields

Within the records described above, some items (`READ`, `RESPOND`, `SAVE`) explain what you did, and some can be used to explain why you did it. In particular:

- The `TYPE` and `SENDER` fields are most likely to hold information explaining why you read, responded to, or saved a piece of mail. Thus, they are most likely to hold information predicting how you will respond to future mail.
- The `DATE` field could come into play if Darwin discovered that you are most likely to read a certain type of mail if it arrives on a day when you received fewer than six pieces of mail, least likely to read it if it arrives on a day when you received 12 or more pieces of mail.
- The `ITEM NUMBER` field has no predictive value whatsoever.

Now, if you collect enough of this sort of data, and if you also collect data on how other people handle their mail (together with demographic data on those other people), you begin to have the data needed for data mining on a question about direct mail.

2.2.3 Defining the Target Field

Data mining tools, such as Darwin, deal with data exclusively in terms of records and fields. When a Darwin model makes a prediction, it puts a value in a data field that was previously empty. (Recall that the classification example in section 1.2 predicted the value of a field.)

Each question posed to a data mining tool must have an answer that can be contained in a single field. This field is known as the *target field* (or the *response field*) and the answer(s) we seek are known as *target values* (or *response values*). The target field is also sometimes referred to as the *dependent variable*, because its value *depends* on the values of the other fields, the *independent variables.*

Setting up a data mining problem, therefore, consists in part of asking the right question, and asking it in the right way.

In the example above, if you intend to send a catalog to a list of customers, you might ask

- Will they read it?
- Will they respond to it?
- Will they throw it away?

Of these, the most useful business question is likely to be

- Will they respond to it?

`RESPOND`, therefore, should be the target field.

2.2.4 How Much "Data Trimming" Should You Do?

One consideration when using Darwin is how tightly you should define your dataset.

Traditional methods of data analysis mandated that you remove all data fields except those you were sure would be useful. This was done partly to reduce the number of variables (thus reducing the complexity of the calculations required) and partly to reduce the volume of data you or your computer would need to handle.

With Darwin, this "data trimming" may be inappropriate. One major benefit of intelligent data mining is the ability to find unexpected relationships. In addition, Darwin is designed to handle both extremely complex records and very large datasets. Therefore, you may find it beneficial to give Darwin as much data as possible, and let the results of Darwin's preliminary analysis guide you in making further decisions.

In these circumstances, considering the dimensions of a problem becomes a guide to making sure you do not forget to include some useful data fields, not a restriction against possibly including other available data. Indeed, you may wish to enrich the data, adding information that can help you find relationships. If, for example, average income is of interest but zip code information is not part of the dataset, you may wish to add it so that you could find relationships between income and neighborhood. Also, there may be circumstances in which you would find it helpful to combine data from more than one database.

See also sections 3.3 and 3.4.

3 Preparing the Data

Having established a basic sense of your business problem and determined the sort of data you will need, you will probably find yourself dealing with these types of questions:

- General questions:
  - How much data do I have?
  - Do I need data from more than one source or database?
  - What period does the data cover?
  - How clean is the data? What do I do about dirty data?
    - Do I trust its accuracy? Are some values particularly likely to be incorrect?
    - Are there likely to be missing values?
- Setting out the problem:
  - How much of the data do I want to use?
  - Do I want to create a dataset for this problem only, or do I want a more general dataset that I can used for other problems as well?
  - What data dimensions do I need to analyze my problem? What is the correct granularity for each?
- Specific questions:
  - In what form is the data currently held?
  - What is the current record format?
  - Is the data consistent? (Do the records have the same field names or are different names used in different databases?)
  - Can I use the data as is, or should I reorganize it in some manner for this particular analysis?
  - What fields should the records contain?
  - What will be the target field?

3.1 Preparing to Create a Darwin Dataset

From the data you choose, you (or someone else) must create a Darwin dataset. A dataset contains all the records for a given problem or question, each containing all the fields and values in each field. You need to determine the format of each data record, which means deciding

- the number of fields in the record
- the order in which the fields occur
- what name you want to give each field
- for each field, whether the data is *categorical* or *ordered*
- for each field, the data type, as it exists in the source data

The instructions that tell the computer program how to interpret the characters in the file is called a *dataset descriptor*. The dataset descriptor must contain all the information needed to interpret the data file, including the items in the bulleted list above.

A dataset descriptor file can be created "by hand" or automatically, on your UNIX server. See *Using Darwin,* chapters 7 and 13, for details.

Both the data and the descriptor must reside on UNIX.

3.2 Field Names

Darwin allows you to name data fields anything you want. It uses, and displays, whatever names you supply in the dataset descriptor file. You can use either of these naming strategies:

- Use meaningful names, such as `date`, `zipcode`, etc. This makes working with the data somewhat easier, as the user sees meaningful names when using Darwin to display, analyze, and/or modify the data.

- Use meaningless names such as `F1`, `F2`, etc. This strategy is appropriate when data security is an issue. If you select this approach, you will probably need to supply printed information regarding field contents to the analysts who will work on the data.

3.2.1 Naming Objects

Darwin checks the validity of names of objects, i.e., projects, models, datasets, or input and output files. A name is valid only if it consists of characters from the following sets:

- numerals zero through nine (`0 – 9`)
- lowercase alphabet (`a – z`)
- uppercase alphabet (`A – Z`)

The following characters are *not* permitted in the names of Darwin objects:

```
. , : ; < > # @ ! ? | [ ] { } ( ) ~ % ^ & * “ ’ \ /
```

Control key characters are not allowed as input to Darwin dialogs. If you accidentally enter a control key character, Darwin beeps and removes whatever input you entered. Re-enter the input without the control characters.

3.3 Cleaning Data

Three major issues in cleaning data are *noise* (incorrect values), *missing values*, and improper use of *separators* (also called *delimiters*).

3.3.1 Noise

Noise in the data is frequently the result of typographic errors. In numeric data, it may show up as out-of-range values — numbers that are either much too large or much too small. In categorical data, it may show up as a unique or meaningless value. Values that are obviously incorrect should be corrected if possible, or else replaced with some agreed-upon value –– either something like a mean or average value or an artificial value that signifies “erroneous value replaced.”

It is of course practically impossible to get rid of noise entirely — any error that is close to a reasonable value is likely to be missed.

There is also intrinsic noise in any data, as the target will rarely follow a perfect deterministic function or statistical distribution.

3.3.2 Missing Values

Missing values can be handled in several ways:

- If most records have no value in a particular field, consider removing that field from your dataset. (Such attributes usually prevent the discovery process from developing strong patterns.)
- If only a few records have no value in the field, consider removing those records from the dataset.
- In some cases, you can replace missing values with an average (mean, median) value (for an ordered field, average value in the dataset; for a categorical field, a typical value in the dataset, i.e., a value that most of the records have for that field). In some instances, max or min may make sense.
- In other cases, you might replace missing values by some constant value that does not appear elsewhere in that field. For instance, given a field that contains values from 0 to 1000, you could use 2000 as your substitute value.

  If you use this strategy, make sure you choose a value that does not appear in your data. For instance, if a field contains values from 0 to 1000, you probably should not use 0 to indicate a missing value. (This may sound obvious, but it is a surprisingly common error and can create puzzlingly inaccurate predictions.)

Using Darwin describes how to accomplish these possibilities.

3.3.3 Separators

A separator (delimiter) is a symbol used to separate fields within a record, and problems arise if the same symbol appears *within* a field. For example, if a text file contains commas within any of its fields, it must not also use commas for separators (and vice versa). If commas are the separator in a text file, then, for example, a comma in an address field is interpreted as the end of that field, and everything thereafter is read into the wrong field, inviting the conclusion that this record contains more fields than it should. If you are choosing separator characters for a dataset, do your best to choose something that will never appear in your data. Try exclamation points (`!`), vertical lines (`|`), curly braces (`{` or `}`) — whatever you think will be safe.

Remember, too, that Darwin interprets two separators appearing side by side (such as `,,`) as representing an empty field. Thus, given the string `A,,B,` in a file that uses commas as separators, Darwin assumes that the first field contains the value `A`, the second field is empty, and the third field (if there is one) contains the value `B`.

3.4 Organizing Data

There are many occasions on which you will want to create datasets for Darwin with record structures that differ from those of the original data. This requires thinking through the problem and doing some manipulation of the data. Sometimes the preparation is best done outside of Darwin; sometimes it can be done within Darwin, using the **Dataset** menu's **Transform** commands on the dataset. (Chapter 5 describes these commands.)

When working with relational databases, you can reorganize your data by using SQL scripts either in the database or in Darwin (if the script is complex, it may be best to use it in the database). See section 4.6.2 for more information.

Here are some examples of situations that might require changing record formats:

- Security or privacy issues may require that you either remove meaningful identifiers or replace them with encoded or encrypted identifiers.
- Some categories of discrimination are illegal in certain situations. Fields that contain information that might lead to illegal discrimination should either be removed from the data or should be labeled in some legible way so that the fields can be omitted, if necessary, at some later time.
- A record of sales might include the bin number of each item of merchandise; that information would be useless for marketing research, and might be removed to save space.
- Sales by quarter for various items may have been recorded as

  ```
  item_code, Q1_sales_target, Q1_actual_sales, ...
  ```

  For some analyses, you might prefer to look at what percentage of target was met during each quarter:

  ```
  item_code, Q1_percent_target_achieved, ...
  ```

In this case, you would perform calculations on fields in the original data to create new fields for the Darwin dataset.

- A data field may carry too much information. For instance, if you are building a model to determine which users are likely to cancel a service, the termination dates of those who have canceled cannot be part of the record used to build the model. (If the termination date is included, Darwin will see it as the most important field in the record — which will be true, but not helpful for predictive purposes.)

  In a case such as this, you might leave the termination date in the first dataset you create, but then zero out the field in a transformation dataset. That way, you will have the full record available for other purposes, but it will not interfere with the current task.

- Two or more groups who keep records in different formats might choose to do some joint research. They would have to decide on a common record format for their joint dataset.

- An analyst might wish to study patterns of purchases, in order to discover which items are commonly purchased together. In the original data, each item purchased might have its own record. In the dataset, all items that were purchased at the same time would share a record. The original records might look something like this:

  ```
  store_code, bill_number, date, line_number, dept_code,
  item_code, quantity, price, discount, value
  ```

  The records wanted for the dataset might look like this:

  ```
  store_code, bill_number, date, item_code_1, quantity_1,
  item_code_2, quantity_2, ... item_code_n, quantity_n
  ```

  In this case, the analyst would select and group fields from several records to create each new record for the dataset.

4 Creating and Handling Datasets

> **Note:** Beginning with this chapter, there are many references to menus and commands on the Darwin graphical user interface. All the Darwin menus and commands are described in detail in *Using Darwin*.

For Darwin, all data is in the form of *datasets*. Datasets are 2-dimensional arrays; the dimensions are *records* and *fields*.

You use Darwin to create a dataset from a collection of data records plus a dataset descriptor file. The dataset descriptor tells Darwin how to read the data records. If you create a dataset from a database, Darwin creates the data descriptor from the database table.

4.1 Before Creating a Darwin Dataset

Before you are ready to create a dataset, you will have completed the planning phase of the data mining process, which means you will have

- formulated your business question (described in chapter 2)
- located, organized, and prepared the relevant data (described in chapter 3)
- transferred or arranged for the transfer (see your system administrator) of your data to the UNIX system where Darwin is installed so that Darwin will have access to it

and you will have

- started a Darwin session and created a project (how to do this is explained in *Using Darwin*.)

Then you (or your system administrator) must transfer the data file and descriptor file to the appropriate project directory.

Note: Because there are some initial steps that are performed on UNIX, you may be tempted to simply create, while you're still on UNIX, the project subdirectory within your Darwin directory on UNIX. Do not do this. There is more involved in creating a Darwin project than simply creating its UNIX subdirectory. Creating the project within Darwin, using the **New Project** command, accomplishes all the necessary related operations.

4.2 Creating Datasets

All the commands related to creating and working with datasets are accomplished by clicking pull-down menus and commands or icons on the Darwin graphical user interface. See *Using Darwin* for details.

You can create a dataset from a text file and a descriptor file or from a database:

- **Import Text File:**

  Text files are *flat files*. On UNIX, they reside in a UNIX file system. Therefore, they can be managed with standard UNIX file-handling commands outside of Darwin as well as by Darwin dataset commands.

  To create a dataset from a file, you need the data itself plus a dataset descriptor file. The dataset descriptor file identifies fields and data types, which it converts if necessary into data types that Darwin can use. Section 4.5 describes dataset descriptor files in detail.

  The mechanics of creating a Darwin dataset from a text file are described in *Using Darwin*, section 7.1.1.

- **Database:**

  If you are creating a dataset from a database table, you do not have to create a descriptor file; the database already contains the descriptor information. Again, Darwin automatically converts data types as necessary (see section 4.5.3 for a list of Darwin and ODBC data type counterparts).

  Note that SQL has no notion of ordered versus categorical fields; a dataset created from a database has all fields designated as ordered. Once the dataset has been created, therefore, you may need to identify for Darwin which fields, if any, are categorical. The Darwin **Dataset** menu's **Transform** command **Set Form** lets you change ordered fields to categorical and

categorical fields to ordered. (**Note**: Fields such as zip codes, which contain very large numbers of values, may be better left as ordered fields.)

See section 4.6 for general information about working with databases.

The mechanics of creating a Darwin dataset from a database are described in *Using Darwin,* section 7.1.2.

Darwin can also create datasets from other datasets open in memory; these are transformation datasets.

Transformation datasets are usually implemented as instructions that operate on the original dataset. Being merely instructions, not data, transformation datasets take up little room in memory. They do, however, impose some performance cost. For best performance, therefore, you may wish to copy the transformation dataset, especially if it is small enough to fit into physical memory (use the **Edit** menu's **Copy** command).

Dataset transformations are described in chapter 5.

Note: Darwin 3.0 and 3.0.1 dataset format is not compatible with dataset format for earlier releases of Darwin. Text datasets are recognized by all versions of Darwin, so convert all old datasets you want to use in Darwin 3.0/3.0.1 to text datasets. Do this before installing Release 3.0/3.0.1. See the Darwin 3.0.1 release notes.

4.3 Serial and Distributed Datasets

Serial dataset files are stored on a single node: a workstation, a file server, or the host file system for a multiprocessor. They can be used on any type of system, and can easily be transferred from system to system.

Some platforms and some servers support distributed datasets. Distributed datasets are stored across all the processors on which a parallel server is running. All nodes work together *in parallel*, each using its own data. Distributed datasets may improve I/O performance. However, they should be run on the server on which they were created.

In Darwin 3.0.1, the distinction between serial and distributed datasets and models is typically transparent to the user, since any saved object is, by default, serial. It is however best not to try to use a model created in a serial server with a dataset created in a multi-node server. Also, if you move your dataset between servers with different numbers of nodes, serialize the dataset first.

You can change a dataset from serial to distributed or from distributed to serial by clicking a toggle button. See *Using Darwin,* section 10.1.2, for details.

Note: Before changing a dataset from serial to distributed, you should randomize the order of records, so that each processor gets a random selection of records (use the **Dataset Transform** menu's **Randomize** command, section 5.2.7).

4.4 File and Array Datasets

Most datasets are created from data held in files or in datasets. Darwin refers to these as *file-based* datasets. Datasets that are created in memory, working from data found in other open datasets, are called *array datasets*.

Like file datasets, array datasets can be either serial or distributed. The **Randomize** command creates serial datasets on a serial server and creates distributed datasets on a parallel server. Datasets created by **Copy** mimic the source datasets: a copy of a serial dataset is serial, a copy of a distributed dataset is distributed.

Array datasets are memory objects, which means they disappear upon exiting Darwin unless you have saved them to file.

4.5 Dataset Descriptors

This section describes dataset descriptor files and explains how to create them "by hand" from within Darwin. You can also use the UNIX command `darwinDG` to create dataset descriptors automatically.

For the mechanics of creating a descriptor file by hand, see *Using Darwin,* section 4.6. For the mechanics of creating a descriptor file automatically, see *Using Darwin,* section 13.1.

Dataset descriptor files contain information that allows Darwin to read and manipulate the data records:

- the number of records in the dataset
- the number of fields in each record
- the name of each field (that is, the identifier you give the field for use within Darwin)
- the form of each field, i.e., categorical or ordered

- the data type of each field, as it appears in the source data (see section 4.5.2 for permissible data types)

Dataset descriptors can be stored inside a Darwin dataset file or as separate files. One descriptor file can be used for several datasets.

Dataset descriptor files must follow a specific format (see figure 8):

- The first line identifies the file as a Darwin descriptor file. This line must be exactly as shown, with no leading or trailing spaces.
- The second line supplies the number of records in the file, in the form *n* `records`, where *n* is the number of records in this dataset.
- The third line supplies the number of fields in each record, in the form *n* `fields`, where *n* is the number of fields in each record. Darwin checks for the presence of this field, but counts fields itself anyway, and works from its own count. (If you specify, say, 25 fields, and Darwin finds 30, Darwin uses all 30.)

  Note: The **Explode** and **Project** transformation commands change the number of fields in a record. (See section 4.5.4; see also chapter 5 for information about transformation commands, and see section 3.2 for information about field names).
- After the first three lines, you may use blank lines (to make the file more legible) and/or comment lines. Comment lines begin with a number sign (#). Darwin ignores both blank lines and comment lines.
- The fourth line (not including comment lines and blank lines) describes the first field in the record. The format is

  `fieldname form datatype`

  where
  - *`fieldname`* is the name you want given to the field
  - *`form`* is either **`c`** (categorical) or **`o`** (ordered):

    A *categorical* field is one that divides its data into discrete categories, for example, `"Color (Red, Yellow, Blue, Green)"`.

    An *ordered* field contains a range of values that are in some order, for example, `"Temperature (0 to 120 degrees)"`. (**Note:** *Ordered* data is sometimes called *continuous* data.)

    Note: String fields must always be categorical. Even if a descriptor file specifies that a given string field is ordered, Darwin ignores that specification and treats it as categorical.

Databases have no concept of ordered or categorical fields, so datasets created from databases originally have all fields specified as ordered. Use the **Set Form** transformation command to change fields from ordered to categorical where appropriate.

- *datatype* is one of the data types shown in figure 8 and listed in section 4.5.2. It represents the data type used in the source file, not necessarily the one Darwin will use.

- The remaining lines describe the remaining fields, in order, in the same way the first field is described.

- The last line must be exactly as shown, with no leading or trailing spaces.

See figure 8 for a dataset descriptor template, and figure 9 for an example of a correctly formatted descriptor file

4.5.1 File Extensions

A data text file and its descriptor file must have specific extensions in order to be recognized by Darwin as a text file and descriptor file:

- A data text file must have a `.txt` extension.
- Its descriptor file must have a `.des` extension.

If you create the descriptor automatically, the `.des` extension is automatically taken care of.

4.5.2 Darwin Data Types

For text files, the following data types are allowed:

`byte`	`float`
`short`	`double`
`int`	`unsigned`
`lint`	`string` (also `astring`) *`max-num-chars`*

These data types are also specified in the dataset descriptor template (figure 8); their definitions are included in Table 2, on page 31.

Note: There are working `*.txt` and `*.des` files available online as samples for you to experiment with. See the release notes for their location.

```
DARWIN(tm) DATASET DESCRIPTOR (Thinking Machines Corporation)
<number> records
<number> fields

# First line must be exactly as shown above.
# Second line must supply number of records in file.
# Example: 259000000 records
# Third line must supply number of fields per record.
# Example: 102 fields

# # as first character of line = comment.
# Blank lines and comment lines may be used only after the first 3 lines.

# Non–commented or blank lines after the first three lines describe
# each field in the order in which the fields occur within the record.

# Format
# <fieldname> c|o <type>|<type plus parameters>

# c = categorical field
# o = ordered field

# For each ASCII text file, <type> may be:
# byte, short, int, lint, float, double
# unsigned
# string (or astring) <max–num–chars>

# Single characters are defined as ”string 1”.

# Some sample lines (Actual lines would OMIT comment character):
# field1 o float
# field2 o double
# field3 c string 1
# # parameters: 6 characters and *10^–2 (default*1)

# Last line must be exactly as shown below
END DATASET DESCRIPTOR
```

Figure 8. Template for a Darwin dataset descriptor file.

```
DARWIN(tm) DATASET DESCRIPTOR (Thinking Machines Corporation)
2922 records
23 fields
HHID c string 13
MKTSEG c string 2
OPENDATE c string 6
PRODCAT c string 3
PRODUCT c string 6
ACCTDOB c string 6
CURRBAL o int
NRMPROFT o float
TRXTYPE c string 2
TRXDATE c string 8
TRXAMT o float
DCTYPE c string 2
DCHANNEL c string 2
NOADULTS c string 1
MARITAL c string 1
HASCHILD c string 1
ESTINC c string 1
HOMEVAL c string 1
NETWORTH c string 1
STATUS c string 1
HHBIZ c string 1
HHOLDEST c string 6
SALES o float
END DATASET DESCRIPTOR
```

Figure 9. A correctly formatted descriptor file.

4.5.3 ODBC Data Types

Table 2 shows the data types for databases and their Darwin counterparts.

Table 2. Darwin and ODBC Data Type Counterparts

ODBC Type	Darwin Type	Length
char	string	variable
varchar	string	variable
longvarchar	string	variable
decimal	double	8 bytes
numeric	double	8 bytes
bit	string 1	1 bytes
tinyint(S)	short	2 bytes
tinyint(U)	int	4 bytes
smallint(S)	unsigned	4 bytes
smallnt(U)	unsigned	4 bytes
integer(S)	int	4 bytes
integer(U)	unsigned	4 bytes
bigint(S)	lint	8 bytes
bigint(U)	lint	8 bytes
real	float	4 bytes
float	double	8 bytes
double	double	8 bytes
date	date	4 bytes
time	time	4 bytes
timestamp	timestamp	8 bytes

The last three types listed above have the following internal format in Darwin:

```
date:      int= year*12*31 + (month-1)*31 + (day-1)
time:      double= hour*60*60 + minute*60 + second
timestamp: double= date*24*60*60 + time (date and time computed
                                          as above)
```

where year=0-9999, month=1-12, day=1-31, hour=0-23, minute=0-59, second≧0.0, <60.0.

4.5.4 How Darwin Reads Descriptor Files

Records: If the dataset contains more records than the descriptor file specifies, Darwin ignores the extra records. For instance, if a dataset is 10,000 records long, and the descriptor file says that it is 8,000 records long, Darwin ignores the last 2,000 records in the file.

If a descriptor file specifies more records than actually exist, Darwin sends an error message when it reads the file and cannot find all the records it expects to find.

Fields: Darwin takes the number of fields in a record not from the third line in the descriptor file (which is primarily a memory aid for the person writing the file), but by counting the number of fields that are actually named and described in the file itself.

For databases, Darwin determines field length from the data type. For text files, which have variable length fields, it uses *field separators* (also called *delimiters*) to find the end of each field (see section 3.3.3). You can use blank spaces, commas, or any other character as a separator character. Be sure your separator character does not also appear in any field value, as this will cause Darwin to misread the record. For this reason, unusual characters such as exclamation points (!) and vertical bars (|) are often used as separators.

4.6 Working with Databases

Darwin accepts data from databases. Commands for connecting to and disconnecting from databases are available from the **Project** menu, as described in *Using Darwin*.

You may have only one database connection open at a time. Therefore, you must close one database connection before you open another.

If you do not disconnect from a database before you exit Darwin, Darwin closes the connection automatically.

4.6.1 Creating Darwin Datasets from Databases

There are two approaches to importing data from a database:

- If you can use SQL operators that Darwin supports, you can connect to the database and extract the data with a script. The following SQL operators

are not supported: JOIN, UNION, INTERSECT, and EXCEPT. Using them in a script from Darwin results in an error.

- If you need to use the SQL operators JOIN, UNION, INTERSECT, or EXCEPT to extract the information, create a view (virtual table) outside of Darwin and then import the information into Darwin using a SELECT statement in a Darwin Database Query.

Only one SQL query is allowed when creating a Darwin dataset. At times, however, you may want to take several actions to create the data you want in a dataset. For instance, you may want to sum the values of several fields to get a total value, combine fields from several tables into a single table, and so on.

To provide this functionality, and also to let you avoid repetitive typing, Darwin lets you create SQL scripts and to run these scripts when creating a dataset from a database.

See also section 4.2.

4.6.2 SQL Scripts and Queries

The simplest SQL script consists of a single SQL query. Scripts of this type are used when the same query will be used repeatedly (perhaps by several people) or when the query itself is a long one.

A more complex SQL script consists of three parts, each with a label. Darwin looks for these labels when it executes the script. SQL ignores them. Therefore, the labels each begin with two dashes, which is the standard method of identifying comments in SQL script files. The labels are:

- `--DARWIN_setup`
- `--DARWIN_retrieve`
- `--DARWIN_cleanup`

The *setup* part of the file contains data preparation steps. Generally, this involves creating temporary tables and/or views. The *retrieve* part contains the actual query that provides the data to Darwin. Frequently, this query will be as simple as `select * from new-table`. The *cleanup* part should contain commands that discard any temporary items you created during the setup part. Cleaning up is usually necessary to allow the script to be re-run, and is always good practice.

Notes:

- When you write SQL scripts, remember that all SQL statements must end with a semicolon (**;**).
- When you write an SQL query to ORACLE, it should *not* end with a semicolon. If your SQL query ends with a semicolon, you get a message like this:

```
Darwin User Error –3700: [VISIGENIC] [ODBC Oracle Driver]
syntax error or access violation
```

- A current SQL limitation is that when you use a script or a query to create a new Darwin dataset, do not use the SQL operators JOIN, UNION, INTERSECT, and EXCEPT. Using these operators in either a script or in a query results in an error.

A One-Part Example

This first example shows a simple (one-statement) SQL script file. It reads an existing table, chooses only those records in which the value of a particular field (`col4`) is greater than zero, and creates a three-field dataset from those records, with the first field being identical to `col1` in the original dataset, the third field being identical to `col4` but renamed, and the second field being the sum of values from `col2` and `col3` in the original.

```
select col1, col2 + col3 as col23,
col4 as result
from table1
where col4 > 0;
```

A Three-Part Example

The second example uses three tables from a database: `Orders`, `Items`, and `Stock`. The `Orders` table contains information about individual orders. The `Items` table contains details of the items in each order. The `Stock` table contains descriptive information about items. The attributes in each table are as follows:

Orders	Items	Stock
order_code	item_code	stock_code
order_date	order_code	manu_code
customer_code	stock_code	description
ship_instruct	manu_code	unit_price
back_log	quantity	unit
po_num	total_price	unit_descr
ship_date		
ship_weight		
ship_charge		
paid_date		

The sample SQL script that uses these tables contains three statements in the setup section.

The first statement uses information in the `Items` table to calculate summary information about each order. Specifically, it calculates the number of items in each order, and the total amount for the items in each order (shipping charges, etc., are extra).

The second statement pulls order details and item details into a single record by combining information from the `Orders` table, the `Items` table, and the summary view created by the first statement. It also computes the number of days between `order_date` and `ship_date`.

The third statement incorporates information about units and unit prices for each item into each record of the data.

Data being retrieved for the Darwin dataset is ordered by `order_code` for convenience.

The last portion of the script deletes the views created in this session.

Blank lines in the script are provided for readability only. Blank lines can be used in SQL scripts, but they are not required.

```
--DARWIN_setup

create view ordtot(order_code, num_items, order_amt) as
select order_code, count(*) num_items, sum(total_price) order_amt from
eda.items group by order_code;

create view orditems(order_code, ship_charge, days, stock_code,
manu_code, quantity, total_price, order_amt) as
select o.order_code, o.ship_charge, (o.ship_date-o.order_date) days,
```

```
i.stock_code, i.manu_code, i.quantity, i.total_price, a.order_amt
from eda.orders o, eda.items i, ordtot a
where o.order_code = i.order_code and a.order_code=o.order_code;

create view datatbl(order_code, ship_charge, days, stock_code,
manu_code, quantity, unit_price, total_price, order_amt) as
select o.order_code, o.ship_charge, o.days, o.stock_code, o.manu_code,
o.quantity, s.unit_price, o.total_price,
o.order_amt from orditems o, eda.stock s
where o.stock_code = s.stock_code
and o.manu_code = s.manu_code;

--DARWIN_retrieve

select * from datatbl order by order_code, stock_code;

--DARWIN_cleanup

drop view ordtot;

drop view orditems;

drop view datatbl;
```

5 Transformation Datasets

Datasets frequently need to be modified in one way or another. The Darwin **Dataset** menu's **Transform** commands allow you to make many common modifications.

Transform commands create modifications (or *transformations*) of a dataset. Thus, they involve two datasets: the source dataset, or input dataset, which provides the data to be transformed, and the new dataset, or output dataset, which holds the resulting transformed data.

Darwin assumes that you want to use the dataset in your current project (i.e., the active project, the one you are working in at any given time) as the source dataset and creates a name for the output dataset based on the source dataset's name. You can accept the names that are offered or specify a different dataset and/or name.

5.1 Implementation and Performance

Most transformation datasets are created as sets of instructions to be carried out on the input dataset. As such, they take up little space. However, you will get better performance by copying them to memory if, of course, you have room for them in memory. If the source (input) dataset is considered file-based, the new (output) dataset is also considered file-based (see section 4.4, "File and Array Datasets").

The **Randomize** transformation command always makes a copy of the source dataset. This has implications about memory required to do transforms, especially for very large datasets. The copies are kept in memory only, unless you specify that they be written to a file.

5.2 Transformations

Each dataset transformation command is described below. The mechanics of using the Darwin user interface to execute these commands are described in *Using Darwin,* section 7.3.

5.2.1 Append

Append combines two existing datasets into a new dataset by appending one to the end of the other (a top-to-bottom attachment). The resulting new dataset contains all records from both. Given datasets A and B, it creates a dataset C containing records from A followed by records from B.

Both datasets must have the same fields in the same order. Both must be either serial or distributed.

Assume dataset A contains three 3-field records, the fields are named `f1`, `f2`, and `f3`; all are of type int:

```
10 20 30
11 21 31
12 22 32
```

and dataset B contains four 3-field records, with the same field names and the same data type:

```
40 50 60
41 51 61
42 52 62
43 53 63
```

Append creates dataset C, which contains seven 3-field records:

```
10 20 30
11 21 31
12 22 32
40 50 60
41 51 61
42 52 62
43 53 63
```

Note: Append requires that field names and data types match. Field forms do not have to match, but if they differ, **Append** uses the form from the first dataset. For example, if dataset A declared field 1 as ordered, and dataset B declared it as categorical, dataset C would consider it ordered.

5.2.2 Explode

Explode transforms a multivalued categorical field into several binary categorical fields. You can create one binary field for each value of the multivalued categorical field, or you can create binary fields for only those values that are of interest to you in a given model.

For example, given a dataset containing the field `Color`, which takes the values `red`, `blue`, `yellow`, and `green`, you could use **Explode** to create four binary categorical fields, which you might name `Red`, `Blue`, `Yellow`, and `Green`. The field `Red` would contain a 1 for all records in which `Color` had the value `red`, and a `0` for all other records. The `Blue` field would contain the value `1` only where `Color` had the value `blue`, and so on.

5.2.3 Merge

Merge creates a new dataset by appending all fields from the records of one dataset to the records of another dataset (a side-to-side attachment). Thus, given datasets A and B, **Merge** creates dataset C in which record 1 contains all the fields from record 1 in A followed by all the fields of record 1 in B, and so on.

Thus, if dataset A contains three 3-field records, `f1`, `f2`, `f3`, all of type int:

```
10 20 30
11 21 31
12 22 32
```

and dataset B contains three 3-field records, `f4`, `f5`, `f6`, all of type unsigned:

```
40 50 60
41 51 61
42 52 62
```

Merge creates dataset C, which contains three 6-field records, `f1`, `f2`, `f3`, `f4`, `f5`, `f6`:

```
10 20 30 40 50 60
11 21 31 41 51 61
12 22 32 42 52 62
```

The two datasets must have the same number of records for the merge to take place.

Duplicate Fields

Each field within a dataset must have a unique name. If a dataset has two fields with the same name, Darwin can access only the first one. To guard against this error, the **Merge** command prints out an error message if the output dataset would contain two or more fields with the same name.

Here's how to remove duplicate fields:

- First, examine the input datasets for duplicates (use the **Project** menu's **Display** command)
- If the fields themselves are identical, use the **Select** transformation command to remove one of them.
- If only the names are identical, rename one of the fields. You will need to change the descriptor and recreate the dataset in order to change field names.

5.2.4 Missing

Missing creates a new dataset with a field whose value is set to 0 if value is missing, or 1 if value is present. You can reverse the assignment of values.

You can replace missing values with a user-specified value or with a value typical for that field — the field's average if field is ordered, or a plurality value (a value that most of the records show for that field) if field is categorical.

See also **Select** (section 5.2.11) and **Replace** (section 5.2.9).

5.2.5 Normalize

Normalize creates a new dataset in which all values are converted into into double-precision floating-point values in the range 0.0 to 1.0.

Numbers are normalized as follows: The lowest value (*min*) is normalized to 0.0. The highest value (*max*) is normalized to 1.0. For each intermediate value (*v*), the formula is $(v - min) / (max - min)$.

Character strings normalize to their numerical ASCII equivalents, which roughly amounts to alphabetizing them. This distinguishes among the categories, but is not likely to create any useful ordering. You may therefore want to replace character strings with numerical codes, especially if the values do have some logical ordering that might be useful for your analysis.

Note: Neural nets require normalized data, and Darwin **Net** automatically normalizes the data as part of its processing.

5.2.6 Project

Project creates a new dataset containing only specified fields from the source dataset.

For example, given a dataset with fields `f1` through `f10`, you could create a projected dataset containing only fields `f3` and `f8`. You can do this by specifying either the fields to *include* or the fields to *exclude* in the output dataset.

5.2.7 Randomize

Randomize creates a new dataset containing all the records of the original dataset, reordered in a random order.

Randomization ensures a random distribution of records throughout a dataset. Some cases in which randomizing records is recommended are:

- when you are creating a training dataset for a net (nets need to have records presented in random order)
- before you divide a dataset into subsets for training and testing
- to ensure load balancing for distributed datasets after using the **Select** command

The purpose of randomizing is to minimize the distortion in results that can occur because of some incidental and unknown clustering or ordering of records. Typically you will copy the base dataset before randomizing so that you can preserve the original.

Note: Because random access to ASCII text files is very slow, randomization of datasets based on such files is also slow. To improve performance, **Randomize** first makes a temporary copy of this type of dataset in virtual memory (space permitting), or creates a temporary distributed dataset file that it deletes when the operation is finished), then randomizes it. If there is no space for the temporary copy, Darwin generates an error message and does not perform the transformation. (An alternative approach is to first **Save** the dataset, which creates a dataset file; then open that file and randomize it.)

5.2.8 Range

Range creates a new dataset from a range of records (that is, a contiguous subset) of the source dataset. For example, you might divide a 1000-record dataset, `Q1.sales`, into four smaller datasets, by running **Range** four times:

- `Q1sales.1`, containing the first 25 percent of records
- `Q1sales.2`, containing the second 25 percent of records
- `Q1sales.3`, containing the third 25 percent of records
- `Q1sales.4`, containing the final 25 percent of records

For serial datasets, you supply the starting and ending percentages for the range. For distributed datasets, you select portions of the records available on each node. You specify the starting point and ending point as percentages, and the specified portion is selected on each node. Thus, a **Range** dataset created from a distributed dataset maintains the same distribution of records per node as the original dataset, and does not transfer records from one node to another.

See also **Split**, section 5.2.13, which automates this process for three subsets of a dataset.

5.2.9 Replace

Use the **Replace** command when you want to alter values within records. Replace works on a single field. It creates a new dataset in which the values in a specified field are replaced by a new value, which you specify. You specify

- the field you want to modify
- the values you want replaced
- the new value should replace the original values

You specify the values to be replaced by specifying a value plus a logical test:

- equal to
- not equal to
- greater than
- greater than or equal to
- less than (lower than)
- less than or equal to (lower than or equal to)

For example, you could specify any of the following actions for a particular field:

- replace values greater than or equal to `20` with the value `20`
- replace values not equal to `Red` with `Not-Red`
- replace values less than `3000` with `0`

As **Replace** creates its new dataset, it checks the value in the selected field in each record; it replaces values that meet the required test with the specified new value and retains all other values as they are.

As an example, take the following dataset:

```
f1    f2    f3    f4
 10   300   300   550
400   600   710   890
226   550   220  1600
```

Suppose you want to replace values less than `100` in field `f1` with the value `100`. After using **Replace**, the new dataset will look like this:

```
f1    f2    f3    f4
100   300   300   550
400   600   710   890
226   550   220  1600
```

5.2.10 Sample

Sample takes a random sample of records from a dataset. You specify a *sample rate* to specify the fraction of the dataset that you want used in the new dataset. For example, you could choose to sample roughly 10%, 20%, or 50% of the records. (The numbers may not be exact: sampling rates are not identical to percentages.) The default is 100%.

Note: You can take multiple samples, but you have no guarantee that records chosen in one sample will not reappear in a second sample. If you want to take several distinct random samples, you should first randomize the entire dataset and then use the **Range** command to take separate subsets of that dataset.

If the entire dataset is so large that you don't want to randomize all of it, you can break a large sample into ranged samples. For instance, you could sample 30,000 records out of a 100,000-record database, and then break the 30,000 records into three 10,000-record datasets.

5.2.11 Select

Select creates a new dataset containing only those records in which the value of a given field meets a specified logical test:

- equal to
- not equal to
- greater than
- greater than or equal to
- less than
- less than or equal to

For example, if Customer Income is a field in your original dataset, you can use **Select** to create a new dataset that contains only customers whose income is greater than, for example, $50,000.

You can also use **Select** to specify those records in which a given field's value is (or is not) missing.

Note: When you use **Select** on a distributed dataset, records remain on their original nodes. If, in this case, you use **Select** on a highly ordered field (for example, a field that has been sorted in increasing value), the resulting dataset may have few or no records on some nodes, and many records on other nodes. To avoid this imbalance, **Randomize** a distributed dataset first, then **Select** the records you want from the randomized dataset.

5.2.12 Set Form

Set Form creates a new dataset in which specified fields are changed from ordered to categorical and vice versa.

Note: Datasets created from databases have all fields specified as ordered (databases do not distinguish between ordered and categorical). Use **Set Form** to change the form of those fields that you want Darwin to treat as categorical.

5.2.13 Split

Split creates three new subset datasets from the source dataset. You specify the percentage of records for each subset.

See also **Range**, section 5.2.8, which allows you to divide a dataset into any number of subsets.

5.3 Using Transformation Datasets

Frequently, you want to perform more than one transformation on a dataset. Darwin allows you to transform a transformation dataset. You can chain several such transformations, providing a new dataset name each time. Because the datasets (except those created by **Randomize**) are implemented as pointers plus instructions, the multiple transformations do not require additional memory.

Multiple transformations will, however, create poor performance. If possible, therefore, you should either make an in-memory copy of the final transformation dataset, or save the final dataset (the one that contains the result of all the transformations) into Darwin dataset format. Then you can unload the intermediate datasets you created, open the saved dataset file, and get good performance from that dataset.

5.3.1 Example of Multiple Transformations

Assume that you want to create a neural net to model some aspect of customer behavior relating to customer income. When you look at your base dataset (using **Frequency Count** from the **Analysis** menu), you notice that income distribution among your customers is heavily skewed toward lower incomes.

You know that a net model works best when it has roughly equal numbers of target values in its dataset, so you decide to create a training dataset that will be relatively rich in high-income customers.

This example assumes a serial dataset on a single-processor machine.

Note: For net models, Darwin automatically normalizes the data during operations, so there is no need to run **Normalize** on the dataset.

1. **Select:** First, use **Select** to create a dataset containing customers whose incomes equal or exceed some lower bound. You might name it `income.ds.1`.

2. Run **Select** again on your original dataset, this time selecting only those who do not meet your chosen income level. Call this dataset `income.ds.2`.

3. **Sample:** Run **Sample** on `income.ds.2` to get a sample roughly equal in size to `income.ds.1`.

4. **Append:** Append `income.ds.2` to `income.ds.1`, calling the new dataset `income.ds.3`. You now have a dataset that contains roughly equal numbers of high- and low-income customers.

5. **Randomize:** Run **Randomize** on `income.ds.3` and name the result `income.ds.4`.

6. **Display:** Use the **Project** menu's **Display** command (or double-click on its name in the **Workspace** listing) to view a few records from `income.ds.4`, just to make sure you see what you expect.

7. **Split:** Run **Split** on `income.ds.4` to create your training, testing, and prediction subsets of `income.ds.4`. Edit the suggested names for the output datasets to rename them as follows: `income.train`, `income.test`, and `income.pred`. The three subset datasets are automatically saved.

You're now ready to use the three subset datasets to build a model.

6 Introducing Darwin Models

At this point, you have created a Darwin dataset and you are ready to choose the type of model you want to build.

This chapter introduces the different types of models and provides an overview of the model-building process. Information specific to each type of model is provided in the next three chapters: chapter 7 for trees, chapter 8 for neural nets, and chapter 9 for match models.

Darwin offers three methods for classifying, predicting, and forecasting data:

Model	Method
Tree	Classification and regression trees (C&RT)
Net	Feed-forward neural networks
Match	Memory-based reasoning (MBR), based on a trainable *k*-nearest-neighbor algorithm

Novices and nontechnical users will be relieved to know that Darwin demands no particular knowledge of the theory or implementation of these methods — only a practical knowledge that can be gained through experience.

6.1 Why Multiple Methods?

Experience has shown that, for any given problem in classification and prediction, one of the methods listed above will probably outperform the others. By making all three methods available, Darwin lets you choose the best method for solving your current problem. There are also situations in which you will want one kind of model in the initial stages but a different one for later stages.

You can use the Modeling Wizard to create models rapidly to get an idea of which kind may be best for your particular problem and your particular data.

How will you know which model to choose?

- As a start, use the rules of thumb presented below.
- As you gain experience, you may develop a sense of which type of model you will want to use for a given kind of problem, although a model's performance is always dependent on both the data and the problem.
- In some cases, you'll want to try all the model types on a small dataset, and simply go with the one that performs best. However, some model types converge faster than others, so more than one run may be needed (i.e., several runs with different training dataset sizes.)

6.2 Choosing a Model Type

Each of the model types is based on a specific algorithm and is thus appropriate for certain kinds of questions and certain kinds of data. Each also has special strengths and limitations.

All three model types handle categorical data, either binary or multiclass, as well as ordered (continuous) data. Categorical data is associated with classification problems. Ordered data is associated with regression (forecasting) problems.

Although all three model types can handle the same kinds of data, the way they handle it and the output from each is different.

6.2.1 Darwin Tree

As you start out, the data is an undifferentiated mass, which Darwin **Tree** begins to differentiate by posing specific rules that create *branches* or *splits.* This process continues until the data in the branches is as homogeneous as possible. Darwin also *prunes* trees to define a family of subtrees with different size/accuracy tradeoffs. The idea is to choose the subtree with the best generalization capabilities, which is likely to be one of the pruned subtrees.

Trees are especially useful for problems that have, or appear to have, high dimensionality, i.e., a large number of fields or variables. The process of growing and pruning trees selects those fields that play an important role in predicting the response values.

Trees are also the method of choice if you want to see the rules the model creates. Sometimes, just looking at these rules and seeing the patterns they present can provide useful information. (Net and match models do not show their

internal processes in such a clear and easy-to-read manner.) Darwin **Tree** is described in chapter 7.

Darwin **Tree** models are also exportable, i.e., you can generate code to include the models in application programs or embed them in Web browsers. Therefore, if you will want the option of exporting your model, you will want to use either **Tree** or **Net.** See chapter 11 for details.

Note: The process of creating the type of model called a *tree* is sometimes referred to as *growing a tree*.

6.2.2 Darwin Net

Neural network algorithms grew out of mathematical models for human thinking that attempt to simulate the massive interconnectedness of pathways in the brain. Neural nets are often referred to as "black box" tools, i.e., your data goes in at one end, your predictions emerge at the other end, all without your needing to understand the many complex calculations that happen in between.

Neural networks are known to be extremely successful in developing classification and forecasting models on large datasets. In recent years, there has been further refinement in the mathematical principles that govern the way a neural network works. Darwin provides methods that assist you in building good neural network models as well as understanding the way they work (for example, **Sensitivity** analysis). Darwin **Net** is described in chapter 8.

Darwin **Net** models are also exportable, i.e., you can generate code to include the models in application programs or embed them in Web browsers. Therefore, if you will want the option of exporting your model, you will want to use either **Tree** or **Net.** See chapter 11 for details.

6.2.3 Darwin Match

Match models are excellent at handling idiosyncratic data, and at discovering cluster patterns in data.

Darwin **Match** is based on memory-based reasoning, using a *k*-nearest-neighbor algorithm. This is something like the notion that "birds of a feather flock together" — that is, if you are like your nearest neighbors with respect to many key variables, you are assumed to be like these same nearest neighbors with respect to the target variable.

Predictions are made by looking at similar records (nearest neighbors) in the existing dataset and calculating weighted distances, weighting the attributes according to their importance to prediction and in terms of distance from the target.

With binary classification problems, match models predict values as zero or one, taking the "majority" or "average" of the nearest neighbors to predict the target value. Darwin **Match** is described in chapter 9.

6.3 Building a Model

Whichever type of model you choose, building models involves the same basic process, although the details differ:

- Creating and/or training a model.
- Testing the model.
- Using the model for a practice prediction.
- Analyzing your results.
- Using the model to predict with new, unclassified, data.

This process is, of course, iterative. At any step, you may decide to loop back to a previous step, make modifications, and continue from there. You may even decide that you need different datasets or a different type of model.

For any given problem, you can create several models of one or more types, compare their performance, and use the model that gives you the best results. You may find that you create and test one group of models, make some changes to models and/or datasets, create and test a second group of models, and so on.

Table 3 lists the commands used in this process, showing the similarities and differences among the three model types.

Table 3. Comparison of steps in building models, showing commands and datasets.

	Tree	Net	Match
Create and Train Model	Create Tree (training dataset)	Create Net (training dataset)	Create Match Model (model dataset)
Train and Test Model	Test/Evaluate (testing dataset)	Train and Test (training and testing datasets)	Optimize Weights (testing dataset)
Evaluate Model	Test/Evaluate (evaluation dataset)		
Predict with Model (practice prediction)	Predict with Tree (prediction dataset)	Predict with Net (prediction dataset)	Predict with Match (prediction dataset)
Analyze Results	Evaluate Prediction (merged prediction dataset)	Evaluate Prediction (merged prediction dataset)	Evaluate Prediction (merged prediction dataset)
Loop Back	Reprune Tree	Re-Start	Try different values of *k*

6.3.1 Creating and Training Models

Darwin trains a model by examining the training subset of the base dataset, determining relationships between the value of the target field and those of the independent variable fields — those that enable the model to predict, with some probability, the value of the target field.

A trained model thus expresses the predicted value of a record's target field as a function of the other field values and reports a confidence with each prediction. This is most clearly seen in tree models, which spell out the rules by which they perform their predictions and the confidence for each prediction.

Frequently, you will want to create several models during the training phase of developing a model. The result of the training phase would then be a group of prediction models to be compared with each other. The models differ from each other according to type and the parameters you set for each. The choices available to you for creating different versions of each type of model are explained chapters 7, 8, and 9.

6.3.2 Training and Testing Models

All models go through training and testing phases, though the details differ:

- With **Tree**, you use the training dataset to grow a full tree; you use the testing and evaluation datasets to compare the effectiveness of various subtrees.
- With **Net**, you have several options: One is to use the training and testing datasets for a combination of training and testing. Another is cross-validation, which does the same and then switches the roles of the two datasets.
- With **Match**, the training dataset becomes part of the model; you use the testing dataset to optimize the weights in your model.

For **Tree**, the testing passes are explicit — you run one or two testing passes over separate subsets of your dataset. For **Net** and **Match**, the series of testing passes are built into the training phase and optimization.

Overtraining

It is always possible that the training dataset contains idiosyncrasies in its data, and, if it does, the model will reflect those idiosyncrasies. The model's predictions will of course also reflect these idiosyncrasies. This problem comes about because of *overtraining*, i.e., at a certain point in its training, the model has gone beyond learning everything that's useful and has begun to incorporate nonessential, idiosyncratic features of the training dataset. The model commits the dataset to memory.

Darwin's models have different ways of avoiding overtraining. Darwin **Tree** lets you guard against it explicitly by testing the training dataset over two separate subsets of your base dataset. The assumption is that idiosyncrasies in one subset are not likely reproduced in another. **Tree**'s other protection against overtraining is pruning and creating subtrees; Darwin removes branches that have led predictions astray. (This is why it is a pruned subtree that is likely to be your best tree model.)

With **Net** and **Match**, the protection against overtraining is implicit. **Net**, for example, stops training on its own when it detects signs that overtraining has started, e.g., when error rates have stopped decreasing and have begun to rise again. **Net** can be trained using a "train and test" method, which, in one operation, uses one dataset for training and another for testing. Another option is cross-validation, which does the same thing and then switches the roles of the testing and training datasets.

6.3.3 Predicting with Models

The prediction phase of building a model is a practice prediction on the prediction subset of your base dataset. This subset of course contains the actual values in the target field, which Darwin ignores when making its predictions. You then compare the predicted values with the actual values, which lets you see how accurate the predictions are.

The output from the prediction phase is called a *prediction dataset*; it contains the predicted value for the target field in each record, plus the confidence for that prediction (a number between 0 and 1, inclusive).

Prediction datasets created by **Tree** also provide the potential cost associated with an inaccurate prediction, and the node of the tree at which the prediction was made. Prediction datasets created by **Net** specify multiple probabilities in cases in which the target field can take multiple values.

6.3.4 Analyze Results

The **Analysis** menu contains commands useful for evaluating the model's performance. The **Evaluate** command compares predicted target values with the actual values and reports error rates. Other commands on the **Analysis** menu produce statistical summaries of the values, frequency counts of particular values, performance analyses, and Lift computations.

Most of the results of these commands can be plotted graphically.

6.3.5 Predict with New Data

When you have selected a model (or models) whose accuracy is sufficient and whose performance is reliable, you use that model with new, unclassified data — data for which you do not know the value of the target field. You use this same **Predict** command to predict values in the target field.

You'll need to perform all the preparatory steps to select and prepare the data, and create a Darwin dataset from the data in order to use your model to predict values in the target field(s).

6.4 A Two-Stage Process for More Complex Problems

When working with relatively small numbers of variables, or on fairly predictable problems, all four steps of the process are typically performed in sequence. For more complex problems and for cases with large numbers of variables and/or data records, the process is frequently performed in two steps:

1. Do preliminary studies on a relatively small subset of records to test and refine the planned method of attack. (Trees, because they provide visible rules, are particularly helpful in this early phase.)

2. Possibly try several types of models.

3. Based on results of these preliminary studies, create a new dataset. The new dataset may contain fields that present data in new ways, such as by summing, averaging, merging, or splitting fields. The full process is now carried out on large numbers of records, and the true "production models" are created.

7 Darwin Tree

This chapter provides an introduction to trees in general and to Darwin **Tree** in particular. See *Using Darwin* for the specifics of using the Darwin graphical user interface to execute the various commands.

This chapter is organized as follows:

- Introduction to Trees
- Creating Trees
- Advanced Options: Tree Size
- Advanced Options: Tree Growth
- The Model-Building Process

7.1 Introduction to Trees

Most business people are acquainted with relatively simple trees called *decision trees*. The trees created (or *grown*) by Darwin are formally known as classification and regression trees (C&RT). By looking at these trees, you can see how Darwin constructed the model.

Darwin **Tree** can perform binary and multiclass classification and prediction problems — i.e., given two or more categories, it can predict the category into which a given record will fall. Darwin **Tree** can also predict values in ordered target fields.

Trees offer clear sets of rules to follow when making the decisions that solve a given problem. In many cases, users find that simply examining the displayed model gives them enough information to act upon.

Darwin **Tree** can solve classification problems with as many as several thousand possible classification values. (Theoretically, the number of possible values is unlimited; but the practical limitations of machine memory, etc., make "a few thousand" the practical limit.)

Darwin tree models are also exportable. See chapter 11.

7.1.1 A Simple Classification Problem

As an introductory exercise, imagine that we are being asked to distinguish among a man, a monkey, a bird, a cat, a whale, and a fish. (In data mining terms, we must classify each record as belonging to exactly one of the categories man, monkey, bird, cat, whale, or fish.)

We have only the data in figure 10 to work with. As you can see, the data is far from complete.

Men	have 2 hands, 2 feet do not have fur, feathers, or scales live on land
Monkeys	have 2 hands, 2 feet, and tails have fur live in trees
Birds	have 2 wings, 2 feet, and tails have feathers live in trees
Cats	have 4 feet and tails have fur live on land
Whales	have flukes and tails do not have fur, feathers, or scales live in water
Fish	have fins and tails have scales live in water

Figure 10. Classification problem.

We can build several trees to solve this classification problem. There are several questions we can ask that will create *splits* (or branches) in the tree, and we can ask the questions in almost any order. Furthermore, since the number of possible classifications is limited, we do not need to use all our information in order to make the classification.

Figure 11 shows one possible tree that we could create from our data.

Notice how few of the possible questions (or *rules*) appear in this tree. You can create other trees, with different rules, from this same set of data. Since the example is designed to create clear distinctions, all the trees you create will probably work.

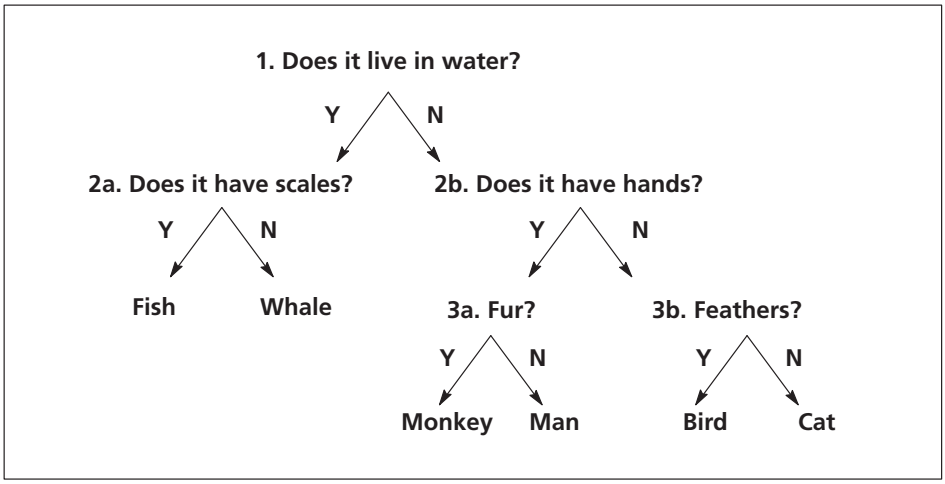


Figure 11. Example tree.

7.1.2 Looking for Specific Answers

In our example, we wanted to know which of six classes each record in our imaginary dataset belonged to. If we were only concerned with one class — for instance, if all we cared about was "Is it a bird?" we would build a tree with a single split, using the split at 3b. We would no longer need to classify every record precisely. We would need only a binary distinction between *bird* and *not-bird*.

Any of the categories can be targeted in this way:

- Is it a fish? = (1) Does it have scales?
- Is it a man? = (1) Does it have fur? (2) hands?
- Is it a monkey? = (1) Does it have fur? (2) hands?

- Is it a cat? = (1) Does it have fur? (2) hands? or, *better,* ask a new question, (1) Does it have four feet?
- Is it a whale? = (1) Lives in water? (2) Has scales? or, *better,* another new question: (1) Does it have flukes?

In other words, your tree reflects the particular problem you are trying to solve. Different problems create different trees from identical data.

7.1.3 Using Trees for Predictions

Prediction comes into play when you have a dataset that cannot provide perfect answers every time.

For instance, suppose we change the example so that the dataset contains only one known field: the field that tells us whether the animal described by each record has feathers, fur, scales, or none of the above. If I say "This animal has feathers," you can say with certainty "It's a bird." If I say "This animal has fur," you can say that it's either a monkey or a cat. The probability of it being one or the other depends on several factors. If I said, for example, "It has fur, and it is a pet," then you consider the relative popularity of cats and monkeys as pets, and you say "Then it's probably a cat." You're still not absolutely certain, but you've now got a pretty good working hypothesis.

Would you send the owner of this "probable cat" a catalog for cat products? Well, exactly how strong is the probability? How much does sending the catalog cost you? Does the catalog contain any items that someone might conceivably buy for a pet monkey?

Given some data, a Darwin tree can answer the second and fourth questions for you. The answer to the third question ("How much will it cost me to send a catalog to someone who doesn't want it?") is up to you. Darwin accepts (but does not demand) information concerning the cost of incorrect positive and/or negative predictions, and factors that information into its calculations.

7.2 Creating Trees

At the root of the tree, the data is in one heterogeneous mass. From the data, the tree generates a series of *rules* — logical tests (or questions) — each of which splits the records into successively smaller, mutually exclusive groups.

Each split creates *branches* in the tree. Branching has two goals: to create groupings that are as homogeneous as possible (at least with regard to the target field), and to create splits that allow for further useful splits as the process continues.

The process of splitting continues until either the set of records in any branch is as homogeneous as possible, i.e., until there are no good splits left, or until a (specified) density threshold is reached, or until a (specified) number of maximum nodes is found. Nodes at the end of branches (the products of the final splits) are called *leaf nodes,* or *leaves*. The splitting process continues until all branches end in leaves.

When the tree has been created, each leaf node contains records that belong predominantly to one class: we describe this by saying that each leaf node is *associated* with a specific class. Several leaf nodes may, of course, be associated with some one class.

The goal is to create a tree that produces the best (i.e., lowest) error rate with the fewest nodes. (The larger the tree, the more resources it consumes when being built and used. The higher the error rate, the less certain the results.)

The way you frame your original business question may well influence the size and performance of your tree. The way you prune the tree, removing leaves and branches to get the best combination of size and error rate, can also be important to size and performance.

7.2.1 Pruning

If a training dataset reflected the full population perfectly, the full tree would have the lowest error rate, i.e., lower than that of any pruned subtree. However, this is almost never the case. All data samples contain idiosyncrasies, training datasets among them. As the tree continues its splitting process, making finer and finer distinctions, it inevitably makes distinctions that reflect the idiosyncrasies of the training dataset and are thus likely to cause higher error rates when used against fresh data. This problem results from what is called *overtraining*.

To ensure the lowest error rate, Darwin *prunes* the tree. That is, it tests the results of removing some of the splits at the ends of branches. The result is a series of *subtrees*, each of which is a differently pruned version of the full tree. When Darwin tests or evaluates a tree, it reports the error rate for each of the subtrees and the number of nodes in each subtree. The user then decides which subtree to use for prediction or classification of new data. Typically you'll choose the subtree with the lowest error rate and the fewest nodes — the fewer the nodes, the more efficient (i.e., faster) the model.

7.2.2 Repruning

Repruning a tree allows you to test an alternative pruning function. This operation affects the organization of the subtrees; it does not affect the structure of the full tree (that is, it has no effect on the rules that result from creating the tree).

The two prune functions are **cost** and **gini**. The default for creating a tree initially is **cost**. The default for repruning a tree is **gini**. See section 7.4.2 for more information about prune functions.

7.2.3 Tree Rules

One of the advantages of trees is that they produce a human-readable set of rules by which the splits are made. Typically you'll decide on a particular subtree whose rules you wish to examine. If you do not specify a subtree, the full tree (0) is used.

The display lists each node that forms a decision point, using the following format:

- The first line identifies the node number, in the form

  `[ TREE NODE N`

- The second line shows the total number of records at this node, expressed both as the actual number and as the fraction of records in the training dataset that the number represents. That is, it shows

  `Total records: r1 (d1)`

  where

  r1 = the number of records in node *N*
  d1 = (number of records in *N*)/(number of records in training dataset)

- The third line shows the number of records with a particular target value (expressed first as the actual number and then as the fraction this represents of records with that value in the training dataset). Thus, it shows

  `Target records: r2 (d2) ]`

  where

  r2 = the number of records in node *N* with the correct classification
  d2 = (number of records in *N* with class = *j*)/
  (number of records in training dataset with class = *j*)

- The fourth and succeeding lines describe the rule that would put a record into the class of records having that target value, as follows:

  ```
  IF certain fields have certain values
  THEN target field = value
  WITH misclassification cost = C
  ```

  The final line shows the potential *misclassification cost* for the record if the prediction were made at this node.

  If you supplied costs for false negative and false positive predictions while creating the tree, those costs, together with the probability of error, make up the misclassification cost given here. If you did not supply costs, then the predicted misclassification cost is simply the probability of error.

Here is a sample rule. Field `a10` is the target field.

```
[ TREE NODE 41
  Total records: 10 (0.057)
  Target records: 9 (0.0628445) ]
IF a1 <= 11 AND
   a4 in { 40 41 42 44 49 } AND
   a13 > 200 AND
   a17 = 512
THEN a10 = 1
WITH misclassification cost = 0.1
```

The statement `a4 in { ... }` means that the value of the field `a4` can be any one of the bracketed values. The cost of 0 indicates that all records at this node fall into a single category. (In this example, they are all responders.) A cost of 0 thus corresponds to a confidence of 1. If target values are mixed, cost is higher, and confidence lower. For example, if `Total records = 10` and `Target Records = 9`, cost would be 0.1, and confidence would be 0.9, or 90%.

See *Using Darwin,* section 8.1.2, for the details of this command.

7.3 Advanced Options: Controlling Tree Size

There are three parameters that affect the size of a tree. The first two, *density threshold* and *max nodes*, have default values that are fine for most purposes; they are called **Advanced Options** on the Darwin graphical user interface (see *Using Darwin,* section 10.1.3). Adjusting their values is most likely to be useful during the early stages of working on a model, when you want to see rapid interim results. The third, *dataset size,* can be manipulated with a combination of adjusting the density value and judicious use of transformation datasets.

7.3.1 Density Threshold

Density specifies the number of records with each target value that halts further splitting. Density is a number between 0 and 1 that represents the fraction of records that halts splitting. It is calculated in roughly the following manner: First, find the target value that appears least often in the dataset, and set n to the number of records in which that value appears. Then multiply n by d, the density threshold supplied by the user (or the default), to get the minimum number of records a node can contain.

When you set a density threshold, you are demanding that every leaf node in the tree must have at least $n*d$ records. Splitting in any branch stops when further splits would create leaves with fewer records than this. When no split on any branch can create nodes with at least $n * d$ records, all growth stops.

The default density for creating a tree is 0.05; this value is in general appropriate for large datasets. For small datasets (such as some of the practice datasets included with Darwin) the density should be set to 0.

7.3.2 Maximum Nodes

Sets an approximate limit on the number of nodes the tree can contain. When the tree reaches this size, further growth stops. Trees whose growth is halted by this method are biased; this option should therefore be used only in the earliest stages of model building. There is no Darwin default value.

7.3.3 Dataset Size

Tree size tends to increase with dataset size. If your full dataset is very large, you may want to create your initial trees on a relatively small number of records or set the density to a larger value, e.g., 0.15, and use the larger number of records only when you are sure you have a good model. For example, if you had 1,000,000 records on which you could train a tree model, you might use only 10,000 records for your first attempts, and work your way up to larger numbers, and not the use the full 1,000,000 records until you were certain that you were creating a useful model.

Do not remove fields from your dataset when creating a tree unless you are certain they will not be needed or should not be there (see section 3.4). In general, it is best to let Darwin decide which fields are relevant to its model.

7.4 Advanced Options: Optimizing Tree Growth

There are several parameters for optimizing tree growth. All have default values, which are appropriate to start with. The sections below describe the circumstances under which you may wish to specify a value other than the default. These are on the **Advanced Options** dialog of the Darwin graphical user interface (see *Using Darwin,* section 10.1.3).

7.4.1 Decrease Function

Darwin trees have two *decrease functions*, which are internal functions for measuring the degree of impurity in a split. By default, Darwin uses the **gini** decrease function. When you grow or reprune the tree, you can select **entropy**, the alternative decrease function, instead.

To create a tree, Darwin begins with the root node, which represents the full dataset, and searches for the "best" split. This first split partitions the dataset into two new nodes.

At each new node, Darwin repeats the process of finding the best split. If there are no "good" splits at a node, the node is left unsplit and becomes a leaf. The splitting process continues until all nodes are leaves.

The decrease functions tell Darwin whether and how to split a given node by computing a quantity called the *decrease in diversity* for each potential split. The best split is the one with the largest positive decrease. If the decrease is less than equal to zero for all potential splits, the node becomes a leaf. You can change the threshold for stopping the splits.

The **entropy** and **gini** decrease functions compute the decrease of diversity of a potential split by comparing the index of diversity of the records at the unsplit node with that of the same set of records after the split. The index of diversity is a measure of the bias (or lack of bias) of target values within a set of records.

A higher index of diversity reflects relatively even proportions of target values; a lower index of diversity indicates that the set of records includes many more instances of one target value than the other. The goal in growing a tree is to reduce the index of diversity. A tree whose leaves have low indices of diversity provides classification rules that can predict the target values with high confidence.

For a potential split S, the **entropy** and **gini** functions compute the decrease as follows:

```
Decrease of diversity =  index_of_diversity (records at unsplit node)
                         – index_of_diversity (records after S)
                      =  index_of_diversity (records at unsplit node)
                         – (PL * index_of_diversity [records at left node after S]
                          + PR * index_of_diversity [records at right node after S])
```

where PL and PR are the proportions of records falling into the left and right nodes, respectively, after the split.

A positive decrease of diversity indicates that the split is desirable because it lowers the index of diversity; Darwin chooses the split with the largest positive decrease. A negative decrease indicates that the split is undesirable because it increases the mixedness of the target values.

The **entropy** and **gini** functions differ in how they define the index of diversity. Assume you have two target values, denoted by + and –, and P^+ and P^- are the proportions of records with target values + and –, respectively, in the set of records under consideration. Then:

for **entropy**: $-(P^+ \log P^+ + P^- \log P^-)$
for **gini**: $2P^- P^+$

7.4.2 Prune Functions

Prune functions dictate how a tree is pruned (split) into subtrees.

Each leaf of a tree is assigned a class and a probability of misclassification. In the discussion that follows, if + and – denote the two classes, and a leaf is assigned class +, then P^- denotes the probability that a record at the leaf has been misclassified (actually belongs to class –), and $P^+ = (1 - P^-)$ is the probability of a correct classification.

The error rate of a leaf is defined in terms of P^+ and P^-, depending on the pruning function you select when creating the tree. Two pruning functions are available: **cost** (the default) and **gini**. They define the error rate of a leaf as shown below. Here, C^{+-} is the cost of assigning a record to class + when it really belongs to class –.

for **cost**: $P^- C^{+-}$
for **gini**: $2P^- P^+$

With the **cost** pruning function, the pruning process takes into account both the probability of making incorrect classifications and the cost of these incorrect answers. Traditionally, the **gini** pruning function is used when the emphasis of the problem is less on predicting classes accurately than on predicting probabilities (proportions of classes at each leaf) accurately. In this case, classification theory says that the best strategy is to create a tree using the **gini** decrease function and prune it using the **gini** pruning function.

7.4.3 Priors

A Darwin tree model works best when it has a reasonable number of examples of each classification value in its training dataset. When only a few possible values exist, it works best with more or less equal numbers of each value.

Reality, however, does not always cooperate with this rule. For instance, a database may accurately reflect reality, yet have 90% "positives" in its target classification and only 10% "negatives."

In order to work around this problem, you can (1) create a training dataset in which positives and negatives are more or less evenly balanced, and then (2) supply priors information that tells Darwin what the true balance of value is.

Similarly, if your historical database population is biased toward one classification, but you have reason to believe that your unclassified population has a different bias, you can supply the expected bias as priors information.

Priors are specified in a file that contains a list of possible classifications. Each line contains one classification (that is, one target value) plus the probability that any given record belongs to that classification. All values appearing in the data must be included.

The format is

```
DARWIN(tm) TREE PRIORS (Thinking Machines Corporation)
target-value-1   prior-1
target-value-2   prior-2
...
target-value-n   prior-n
END TREE PRIORS
```

The rules for priors files are as follows:

- The first and last lines of the file must contain exactly the text shown.
- Each *prior-n* must be greater than or equal to 0 and less than or equal to 1.
- The sum of all *prior-n* values must equal 1.

For example:

```
DARWIN(tm) TREE PRIORS (Thinking Machines Corporation)
good-risk .88
poor-risk .12
END TREE PRIORS
```

This priors file tells Darwin to adjust its findings toward these proportions.

7.4.4 Costs

You can specify the costs involved in making an incorrect decision. Doing so can be useful when the cost to you of different misclassifications varies significantly.

Costs are specified as a matrix, in which rows (*i*) are predicted values, and columns (*j*) are actual values. A cost is the cost of predicting value *i* when the actual value is *j*.

If costs are not supplied, Darwin uses the value 0 for all correct predictions and 1 for all incorrect predictions. Thus, the default matrix for a simple yes/no target would create a costs file that looks like this:

```
DARWIN(tm) TREE COSTS (Thinking Machines Corporation)
    Yes   No
Yes  0    1
No   1    0
END TREE COSTS
```

The rules for costs files are as follows:

- The first and last lines of the file must be exactly like those shown in the example.
- Row and column heads must be listed in identical order.

```
Correct:    1 2 3        Incorrect:    1 2 3
          1                          3
          2                          2
          3                          1
```

- The cost of correct predictions must be 0. The cost for incorrect predictions must be greater than 0.

7.5 The Model-Building Process

Figure 12 illustrates the basic process of using Darwin to build a tree model, showing the input, commands used, and output at each step.

7.5.1 Before You Start

To build a tree model, you start with

- A question you want answered, phrased so that the answer can be given by classifying records according to the values of the *target field(s)* in each data record.
- A source of *historical* data (that is, data for which the values in the target field(s) already exist), from which you create a Darwin dataset.

  Divide the dataset into three subsets: one each for training, testing/evaluation, and prediction. The subset datasets could be, for example, `demo.train`, `demo.test`, and `demo.pred`.
- The name of the *target field* for this tree: that is, the name of the field whose value you want to predict. For example `magazine_subscriber__p`.
- Optionally, information on costs (see section 7.4.4). (Most users prefer to familiarize themselves with the basics before they start using cost information.)

Note: There are two types of fields that can greatly slow the training of tree models, and cannot provide useful data:

- The first of these are fields that have a unique value in each record, so that the number of values equals the numbers of records: for example, a social security number or a record ID.
- The second type are fields that have the same value in all records (these are sometimes called *constant fields).*

Remove both types of fields, if possible, before using the dataset to train a tree. To accomplish this,

- First, use the **Analysis** menu's **Summarize** command (section 10.2) to look at field values. Constant fields can be recognized easily, because they have a standard deviation (`STD dev`) of 0 (except for strings, which may have a standard deviation of 0 but not be constants).
- Then use the **Dataset Transform** command **Project** to drop all the constant fields from the dataset. See section 5.2.6 and *Using Darwin,* section 7.3, for details.

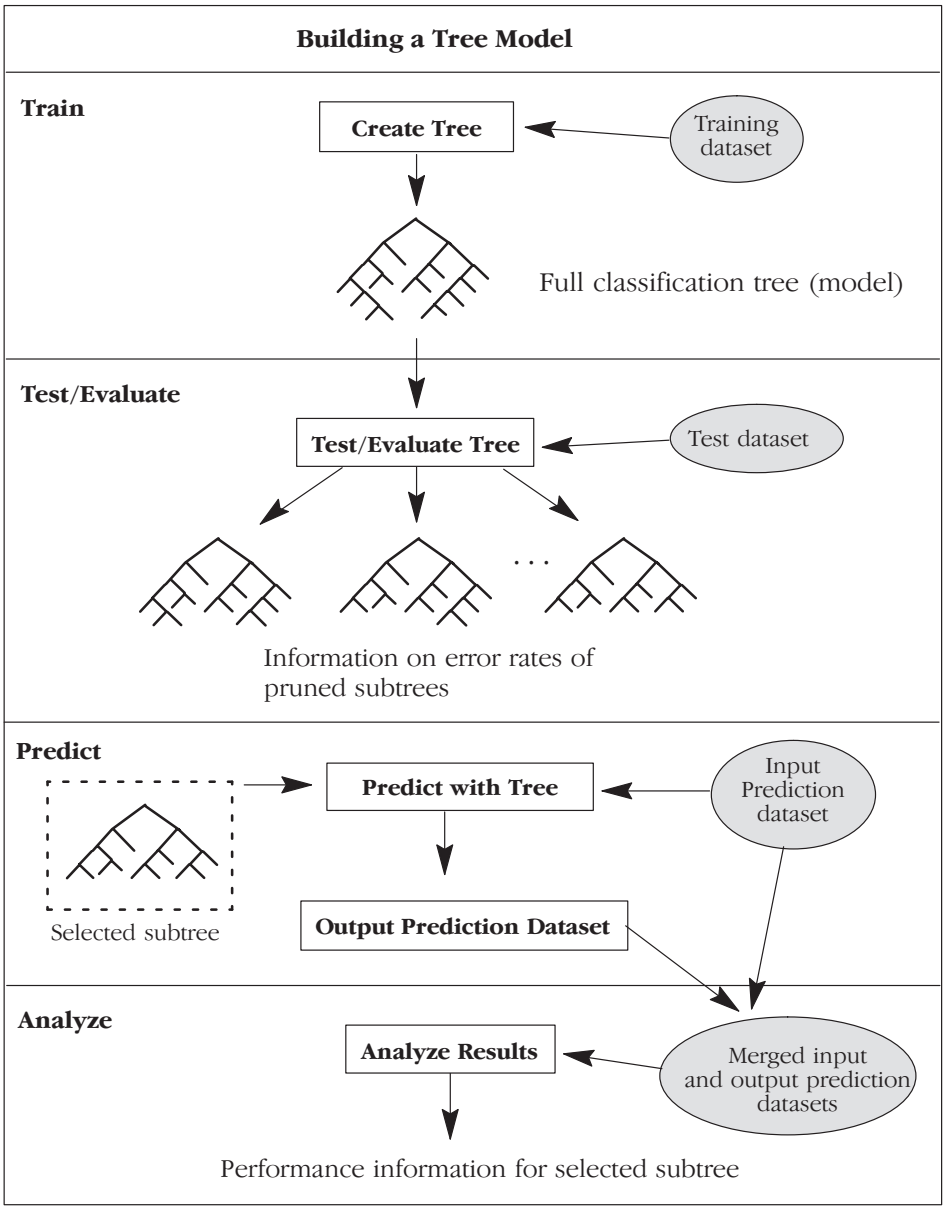


Figure 12. Building a tree model.

8 Darwin Net

This chapter provides an introduction to neural networks in general and Darwin **Net** in particular. See *Using Darwin* for the specifics of using the Darwin graphical user interface to execute the various commands.

This chapter is organized as follows:

- Introduction to Neural Networks
- Creating a Net
- Advanced Options: Building a Net
- Advanced Options: Training a Net
- The Model-Building Process

8.1 Introduction to Neural Networks

As an introduction to neural networks, let's consider how we recognize someone we know. Our mind takes in a great deal of data about shape, colors, movement, expressions, and so on, in order to make the identification. Some of the clues the mind uses are obvious, while some are so subtle that we are not consciously aware of them. In fact, we seldom think of the process at all, except when we try to describe a person.

This complex recognition capability involves massive interconnections of neurons in the brain. The notion of massive interconnections of what we might call *recognition factors* and data paths form the theoretical basis for neural networks. As with the brain, nobody knows precisely how neural networks achieve their results, but we do know that they have an excellent record of success in developing classification and forecasting models on large datasets.

Neural networks work with clusters and patterns of facts, often in highly complex ways. They are often called "black box" tools: your data goes in at one end, your predictions emerge at the other end, and it all happens without your needing to understand the many complex calculations that happen in between.

Neural networks consist of processing units (neurons), and connections that connect these units (synapses) together. The way each of the processing units responds to the information that it receives is determined by its activation function. The strength of the connection between two units is determined by weights. The number of processing units, the kind of connection, the type of activation functions, and the weights in a neural network completely characterize its information-processing characteristics.

Darwin neural nets can handle both binary and multiclass classification problems. They can also handle regression problems, or forecasting: these are problems in which the target value is "continuous" (for example, in which the value could be a number anywhere between 1 and 1,000,000).

Neural net models are exportable. See chapter 11.

8.1.1 The Basics of Neural Networks

Darwin implements the feed-forward network structure, in which the units are constructed as layers:

- The first layer is the *input layer*. It contains one *node,* or *unit,* for each independent field in the record.

- The last layer is the *output layer*. It contains as many nodes as there are outputs. For regression problems, this means a single output node, predicting the value to be found in a single *ordered* target field.

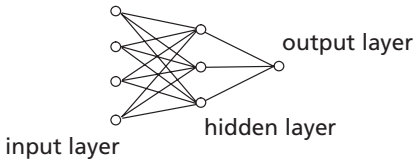
  Binary classification problems use a single output node to distinguish between two values (1 or 0, for *positive* or *negative*) of a categorical target field. Categorical fields with multiple values require one node per value to be predicted, and Darwin breaks the field that carries these multiple values into binary fields, one field for each of the original values to be predicted.

- Between the input and output layers are what are called the *hidden layers*. This is where the work of recognition, classification, and prediction occurs.

  An activation function is associated with each unit in the hidden and output layers. See section 8.2.2.

  The layers are joined by *all-to-all* connections. That is, every node in layer 1 connects to every node in layer 2; every node in layer 2 connects to every node in layer 3, and so on.

  The following diagram shows a very small neural network, with 4 input nodes, 1 output node, and 1 hidden layer with 3 nodes:



Building a neural network model involves creating an initial network structure, training the network on the training dataset, and testing/evaluating the performance on a separate dataset. (You can also use the train and test option or the cross-validation option; these do training and testing simultaneously, using the training and testing datasets; see section 8.3.1.)

Building a net is typically an iterative process that is accomplished by using a set of commands. Once it has constructed a model, Darwin feeds new records into the input layer and makes its predictions or forecasts for the new data through calculations based on the internal state of the network.

8.2 Advanced Options: Building a Net Model

Building a net means designing its topology. The parameters related to building the net are the number of hidden layers and the size of each (number of units per layer), activation functions, and weight. On the Darwin user interface, these parameters are on the **Advanced Options, Net Build** dialog window. See *Using Darwin*, section 10.1.4.

8.2.1 Layers

Net models have an input layer (typically one), a hidden layer (usually one, but could be more), and an output layer (typically one). For each layer, you specify the number of units:

- **Input:** The number of units in the input layer is the number of independent variables in the dataset.
- **Hidden:** The number of units in the hidden layer; by default, this is the same as the number of units in the input layer.

  Optimization: You can have Darwin determine the optimal size for the hidden layer. This is likely to take a while.
- **Output:** The output layer corresponds to the target field(s). The number of units in the output layer must match the number of target fields.

The number of input units plus the number of output units must equal the number of fields in the dataset.

Note: The input layer requires one node for each field in the record, with the exception of the target field(s). For univariate forecasting (regression) and for binary prediction (yes/no) problems, the output layer requries one node. For multiclass classification problems, the output layer requires one node for each potential output value (each of which must be in its own binary field).

8.2.2 Activation Function

Three activation functions are available: sigmoid, hypertangent, and linear. Specify the activation function for the hidden layers and for the output layer.

A good rule of thumb for regression problems is to use linear for the output layer and sigmoid for the hidden layer. For classification problems, use sigmoid for both hidden and output layers.

- **Output Layer:** Sigmoid is the default. Use sigmoid or hypertangent for classification problems; use linear for linear regression problems. Use any of the functions for non-linear regression problems.
- **Hidden Layer:** Sigmoid is the default. For non-linear models, the choice of hidden activation function is dependent on the data. Try both sigmoid and hypertangent and see which works best.

  Note: A linear hidden activation function creates a purely linear network.

8.2.3 Weights

Weight refers to the relative importance of the net's interconnections. Each interconnection is assigned a weight, specified as a single number ranging from $-x$ to $+x$. As a starting point for training, Darwin provides a default set of random weights between −1 and 1.

Training a net means passing the data through the net, adjusting the weights after each pass. The result of training a net is a set of weights that are used in the calculations that determine a prediction.

8.3 Advanced Options: Training a Net Model

The parameters that influence training a net model are learning mode, training algorithm, cost function, and number of iterations. On the Darwin user interface, these parameters are on the **Advanced Options, Net Train** dialog window. See *Using Darwin*, section 10.1.5.

8.3.1 Learning Mode

You can train your neural net model using any of three learning modes: **Train and Test**, **Cross-Validation**, and **Simple Training**:

- **Train and Test** (the default) trains and tests simultaneously, using either two datasets or two parts of one dataset. This mode is helpful in guarding against overtraining of the model when the amount of historical data is relatively small.

  If you use two datasets, you specify the datasets to be used. If you use two parts of a single dataset, you specify the proportions to be used for training and for testing.

  Train and Test produces a table that displays the training error and the testing error. The values displayed are root mean square errors, which may be a bit confusing for binary classification problems because the root mean square error is the square root of the classification error. Because it is a number less than 1, the square root is bigger than the actual error.

- **Cross-Validation** uses an efficient implementation of twofold cross-validation to assist the training process. In cross-validation, Darwin uses its datasets (or portions of a dataset) alternately for training and testing. That is, it first trains with A and tests with B; then it trains with B and tests with A. It then averages the two results to determine the error rate.

  When working with amounts of data that are too small to allow you to split your dataset into three parts, it is often a good strategy to use cross-validation first, to test out various network structures, and then to use train and test to further train the most promising model.

- **Simple Training** simply trains the network for a specified number of iterations. While training proceeds, Darwin displays (in the status bar at the bottom of the window) the number of completed iterations and the current error rate. If you use **Simple Training**, you must test the net in a separate step.

8.3.2 Training Algorithm

There are five training algorithms:

- **modified newton**
- **conjugate gradient** (the default)
- **steepest descent**
- **backpropagation**
- **genetic algorithm**

Conjugate gradient and **modified newton** are superlinear algorithms. In general, they converge (that is, reach their finishing point) much faster than **backpropagation** and **steepest descent**, but they need good starting points.

With most datasets, the **conjugate gradient** and **modified newton** algorithms converge more quickly to useful answers than the other algorithms. At this time, we recommend that you use one of these two algorithms for training.

Backpropagation and **steepest descent** will always converge to some minimum, no matter what their starting point. **Backpropagation** is both the simplest and slowest of the gradient-based algorithms. Its convergence speed is linear.

Steepest descent is like **backpropagation**, except that it uses line search in determining step size.

The final training algorithm, **genetic algorithm**, is the only algorithm in the set that is always able to find a global minimum. But, like any algorithm for global minimization, it may take considerable time to do so.

The rules of thumb for selecting a training algorithm, therefore, are as follows:

- First, try **conjugate gradient** or **modified newton** for several iterations. If this method converges, well and good.
- If the algorithm does not converge, you can either
  - Use the **Transform** command **Randomize** to provide a new starting point for the current algorithm, or
  - Try either **backpropagation** or **steepest descent** to get started, and then continue with your original algorithm.
- If you are still not satisfied with your results, use the **genetic algorithm**.

If you choose either **backpropagation** or **genetic algorithm**, you must specify the **learning rate** for the training process. (For the **genetic algorithm**, this is actually the **mutation rate**.) Set the rate somewhere between 0.01 and 0.05 to start (default is 0.05).

8.3.3 Cost Function

Darwin offers three cost functions:

- **Square:** Square is the default function; it is required for regression problems and can also be used for classification problems.
- **Cross-entropy:** Usually the best choice for classification problems.
- **P Norm:** This is the standard p-norm function of the difference between the model and actual predictor values.
  - If you elect **pnorm**, specify the value of p. The default is 2, which makes the **pnorm** metric equivalent to **square**.

8.3.4 Iterations

Specify the number of iterations the net is to train. The default is 100. This is the number of times Darwin will pass the data through the net to train it.

- You can start with a small number of iterations (say 10 or 20), examine results, then either create a new model, continue with further iterations, or re-start training from a new starting point.
- You can request a large number of iterations (100 or more), watch the error rate as Darwin works, and click **Stop** on the **Project** menu if you want to interrupt the process. When you use **Stop**, Darwin does not halt immediately. Rather, it comes to a clean stop, then presents output based on its results through the last complete iteration.

Whichever strategy you choose, Darwin stops training the model when the specified number of iterations is reached or when it detects the beginnings of overtraining, i.e., when the error rate has reached its minimum and has begun increasing.

8.3.5 Continue Training Net Model

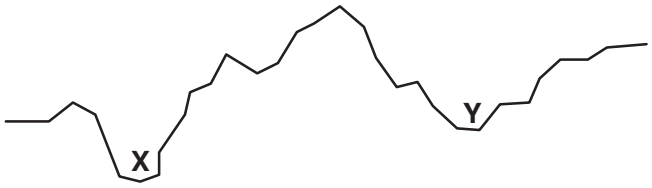
There are circumstances under which you may decide you want to continue training a net model that you have already trained. This can be the case if, for example, you specified a small number of iterations for training and at the end of training you see that the error and testing rates are still decreasing, which means the model had not yet reached its best state.

If you want to continue training in connection with the **Perturb** option, i.e., with slightly different weights, see **Perturbation** (section 8.4.2).

8.4 Retraining a Net Model

When Darwin creates a neural net model, it maps predictions onto *n*-dimensional space, trying to minimize the difference between its predictions and the actual values of the data. The surface of the map may be thought of as being composed of peaks and valleys, with the peaks being high levels of difference, and the valleys being the low levels.

What Darwin seeks is the lowest valley — the global minimum versus the local minimum. For a single variable, the surface might look like the following, with X marking the desired point of lowest error.



In some cases, the net can get "stuck" at point Y during training. This produces a net with poor performance. In these cases, you have three choices: you can create a net with a different structure, you can try a different training algorithm (as explained in section 8.3), or you can re-train the net, this time using a different set of starting weights.

8.4.1 Re-Train

The result of training a net is a set of weights that are used in the calculations that determine a prediction. The term for manually changing the trained weights is *perturb* (see section 8.4.2).

Retraining the net means building a new net with different starting weights.

You can choose to perturb the weights completely or not at all, or any degree in between. You determine the degree of perturbation by moving a slider bar between 0 and 1, where 1 means no perturbation, i.e., the original set of starting weights, and 1 means complete perturbation, i.e., a completely modified set of starting weights.

8.4.2 Perturbation

Perturbation refers to manually changing the weights that result from training a net model.

There are two circumstances under which you might want to perturb the weights:

- To retrain the net using different initial weights, in cases where you are happy with all the parameters and do not want to enter new ones to create a new topology, but you are not satisfied with the model's performance.
- To "dither" the weights a little, in cases where you suspect Darwin may have found a local minimum, but not the absolute minimum, for the error rate. A small change in the weights may allow Darwin to find a new and better error minimum. Getting "stuck" at a local minimum is a possibility with any of the training algorithms other than genetic. The genetic algorithm is always able to find the absolute minimum, but it may take considerable time to do so. (You can select different raining algorithms.)

8.5 The Model-Building Process

Figure 13 illustrates building a net model using the train and test option, which can use two datasets or two parts of a single dataset, one for training and one for testing.

You create nets in two steps. In the first step, you define the structure of the net. In the second, you train the net, using one of the learning modes (see section 8.3.1). Training a net involves passing the data through the net a specified (by you) number of times, with Darwin adjusting the weights after each pass.

8.5.1 Before You Start

To build a net model, you start with

- A question you want answered, phrased so that the answer can be given by the values of the *target field* within each data record.
- A source of *historical* data (that is, data for which the classifications already exist).
- The records in the training dataset should be in random order. (Use the **Randomize** Transform command.) The reason for this is that the order in which records are presented may affect the convergence of the training algorithm.
- Net models require normalized data. Darwin normalizes the data automatically.

- Usually, you would divide your source dataset into three parts, one each for training, testing, and prediction. Use the **Transform** command **Split** for this; but remember to use **Randomize** *before* you use **Split**.

  If you have only a small amount of data, however, you may want to divide your data into two parts, using most of it for training and testing in combination and the rest for prediction. The cross-validation option is particularly useful in this case (see Section 8.3).

- The name of the *target field* for this net: that is, the name of the field whose value you want to predict. For net models, the target field has several possibilities:

  For a regression (forecasting) problem, the values in the target field must be ordered. For a classification problem, you must have one or more binary categorical fields. Darwin accepts multiple fields as target fields, both ordered and categorical; Darwin converts multiclass categorical fields to binary.

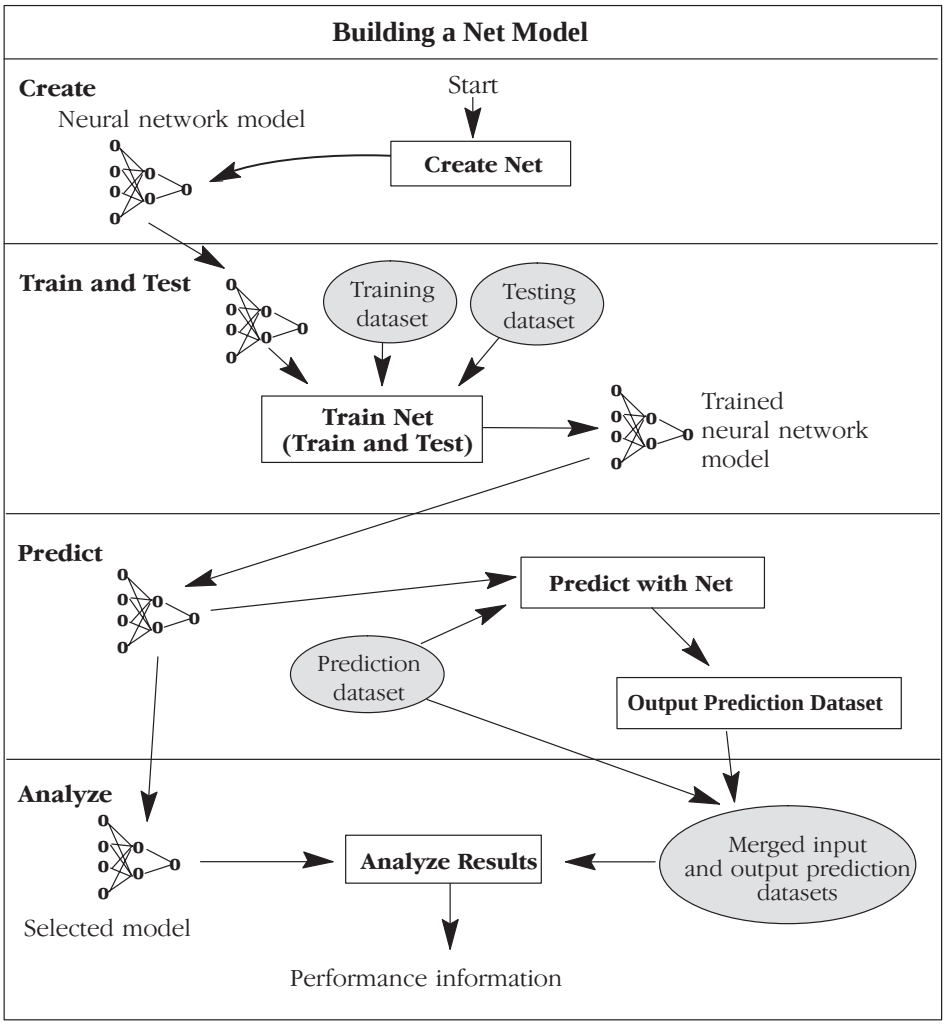


Figure 13. Building a net model (using Train and Test).

9 Darwin Match

This chapter provides an introduction to match models in general and to Darwin **Match** in particular. See *Using Darwin* for the specifics of using the Darwin graphical user interface to execute the various commands.

This chapter is organized as follows:

- Introduction to Match Models
- How Darwin Match Models Work
- Advanced Options: Optimizing Match Models
- The Model-Building Process

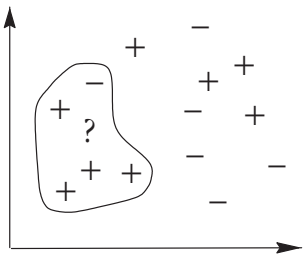
9.1 Introduction to Match Models

Darwin **Match** performs classification and prediction tasks using memory-based reasoning (MBR), with a *k*-nearest neighbors algorithm. Darwin **Match** can perform binary and multiclass classification and prediction problems — i.e., given two or more categories, it can predict the category into which a given record will fall. Darwin **Match** can also predict values in ordered target fields.

Of all the Darwin models, this one perhaps shows most clearly how Darwin makes effective use of all the data in a dataset. Darwin match models are useful in handling idiosyncratic data; they discovers clustering patterns in data, that is, they locates "islands" in the data, whereas other models identify trends.

Memory-based reasoning compares a new record directly to known records. It classifies the known records and predicts the behavior of a new record from the characteristics of those known records that are closest (calculating weighted distances) to the new record.

The two important factors are the number of neighbors and the shape of the neighborhood. The user can decide how many neighbors (the value of *k*) or accept the Darwin default value of 2. Darwin determines the weights attached to each variable, and thus determines the shape of the neighborhood.



- ? is target value.
- Each + is a neighbor with a value of 1.
- Each – is a neighbor with a value of 0.
- Neighbors closest to ? are mostly +; therefore the target value is probably 1.

9.2 How Darwin Match Models Work

Here is a simplified example that illustrates how a Darwin match model works. Each record has four fields: `record ID (#)`, `Age`, `Income`, and `Total Debt`.

The records in our historical dataset are:

```
# AGE  INCOME   TOTAL DEBT     . . .
1  67   48,000     200,000     . . .
2  23   18,000       4,000     . . .
3  52   70,000           0     . . .
```

The new record is:

```
# AGE  INCOME   TOTAL DEBT     . . .
4  54   65,000     154,000     . . .
```

To find the "nearest neighbors" for record 4, compare the value in each field with the value in the same field for the other records. Then add all the differences to get the total "distance" that separates the two records.

A quick look reveals three records that are fairly close in 2 out of 3 dimensions, and are thus "near neighbors,"

```
1   67   48,000     200,000    . . .  NEAREST
4   54   65,000     154,000    . . .  * NEW *
3   52   70,000           0    . . .  SECOND NEAREST
```

and one record that is more distant, and hence less useful for predictions:

```
2   23   18,000       4,000     . . .
```

We know, however, that not all fields are equally useful in predicting all behaviors. In terms of age- and income-related behaviors, record 3 is the nearest match to record 4. In terms of debt- and credit-related behavior, record 1 is much closer.

Also, fields do not offer the same possibilities for distance. Age, for example, cannot vary as much as income and debt. When figuring distances, Darwin automatically compensates for these differences in ranges.

The true metric for closeness, therefore, is the adjusted difference in value for a given field, multiplied by a number that represents the importance of that field in making a given prediction. This adjusted difference multiplied by its relative importance is called a *weight*, and is calculated by Darwin itself.

Fields that are of no value in making a prediction (for example, eye color in predicting credit risk) are given a weight of 0. Fields that are of some small value are given a small weight. Fields that are of great value are given higher numbers. The formula for calculating "nearest neighbors" then becomes, roughly, for each record

```
    (adjusted difference for field1 * weight for field1)
+   (adjusted difference for field2 * weight for field2)
+   (adjusted difference for field3 * weight for field3)
. . . and so on . . .
```

This adjusting (optimization) is what goes on in training a match model. The result is increased accuracy of the trained model.

Notes:

- To see which records Darwin has selected as nearest neighbors when making a given prediction, use the **View Neighbors** command.
- If data has been normalized, ranges for all fields are 0 to 1, so no further adjustments for range are required.
- If two records are identical, the distance between them is 0.

9.3 Advanced Options: Optimizing Match Models

There are three parameters that allow for further optimization of the model. These are weights, the number of neighbors, and bias.

On the Darwin user interface, these parameters are on the **Advanced Options** for **Match** dialog window. See *Using Darwin*, section 10.1.6.

9.3.1 Weights

You can specify a weights file, if you so choose. Otherwise, Darwin creates a default weights file, in which all fields are given equal starting weights.

The match weights file's first line should be exactly

```
DARWIN(tm) MATCH WEIGHTS
```

Starting on the next line, there should be a list of the weights, each between 0 and 1, one weight for each field in the model dataset, separated by whitespace. There is no specially formatted last line for the weights file.

You can also elect to have Darwin *optimize* the weights, which involves adjusting the weights to improve the model's effectiveness.

9.3.2 How Many Neighbors?

A key variable in making predictions is the number of nearest neighbors to use in making the prediction. (This number is known as *k*, as in *k*-nearest-neighbors.) If given no instructions to the contrary, Darwin uses two nearest neighbors, and computes the probability of its predictions accordingly.

If your data is very dense, you do not gain much by increasing the number of neighbors. If, however, your dataset is sparse, increasing the number of neighbors may very well be helpful.

Try your own hand at predicting the following cases:

	Distance of Neighbor	**Target Value of Neighbor**
Case 1:	2	YES
	17	YES
	24	NO
	46	YES
	82	YES
Case 2:	21	YES
	24	NO
	29	NO
	36	YES
	41	YES

What would you predict for the target value of your "new" record in each case? Would you feel confident with your prediction? What if you were using only three

nearest neighbors, rather than five? How would your predictions and confidence change?

Darwin calculates answers in much the same way you yourself probably did. It compares the number of instances of each target value, and the distance between the records containing that value and the record for which the prediction is being made.

The presence of one known record in which all values are extremely close to those of the new record is given a heavy value by Darwin in making a prediction. In this, match models differ from the other Darwin models. In some cases, this heavy reliance on particularly close matches can be useful. In other cases, it will be less so.

You may want to try creating models with various values for **neighbors**, to test the relative performance of each. Often, one tries out various values of *k* on a model to see which performs best. Values from 2 to 20 are common.

9.3.3 Biasing Results

You can also tell Darwin to bias (favor) its predictions towards one or another outcome, if you wish. Doing so affects the way Darwin calculates weights for the various fields.

Note: Setting a bias is optional. You probably want to familiarize yourself with default match models before using this feature.

Bias is set in the following manner:

0 < bias < 0.5 represents *negative bias*.

0.5 is neutral. (This is the default — no bias.)

0.5 < bias < 1.0 represents *positive bias*.

You might choose to bias results in any of the following situations:

- If a false positive result would be much more costly than a false negative, you might set a negative bias.
- If a false negative result would be much more costly than a false positive, you might set a positive bias.
- If in testing your model, you find that it produces an excess of false negatives, you might set a positive bias.
- If in testing your model, you find that it produces an excess of false positives, you might set a negative bias.

Note: If you know which fields you need to use, and you're willing to skip the training process (optimizing the weights), you can make predictions with Darwin based simply on the *k*-nearest-neighbors algorithm, without the benefit of optimized weights. Just supply a historical dataset and a new dataset to the **Predict with Match** command. Darwin then uses the historical data to find "neighbors" for the new data, giving equal weight to all fields.

9.4 The Model-Building Process

Figure 14 illustrates the basic procedure for using Darwin to build a match model, showing the input, commands used, and output at each step.

9.4.1 Before You Start

To build a match model, you start with

- A question you want answered, phrased so that the answer can be given as a value in a single field (the *target field*) within each data record.
- A source of *historical* data (that is, data for which the target values already exist), from which you create a Darwin dataset.
- Divide the Darwin dataset into three subsets: one for the model dataset, one for optimizing, and one for prediction.

  Note: If the target field is binary, normalize the data. Use the **Randomize** and **Normalize** commands from the **Dataset** menu's **Transform** commands before you use **Split** to divide the dataset into subsets. If the target field is multiclass or ordered, do not normalize the data; use **Randomize** and then **Split**.

  The binary version always gives a response of 1 ("yes") or 0 ("no"). If the genuine target values happen to be 1 and 2, use **Replace** (one of the **Transform** commands) to replace the values with 0 and 1, appropriately.

  The three subset datasets you create could be, for example, `demo.train`, `demo.test`, and `demo.pred`. With a match model, the training dataset becomes part of the model.
- The name of the *target field:* that is, the name of the field whose value you want to predict.

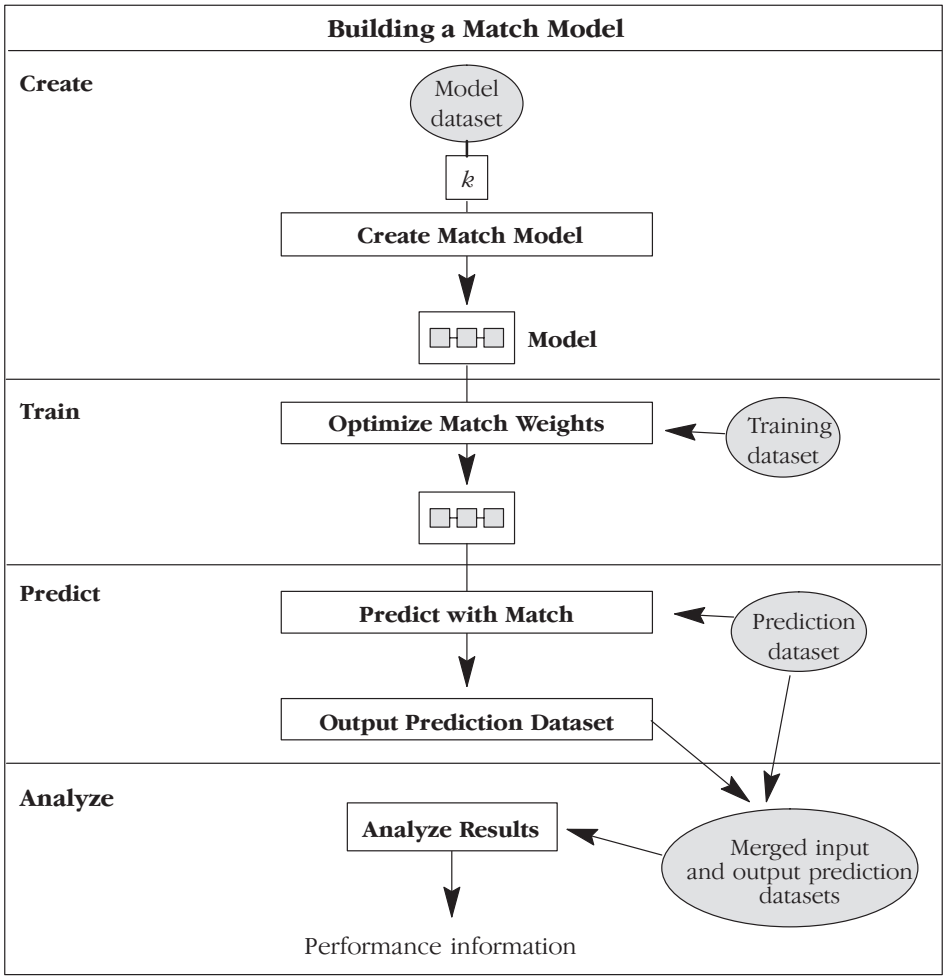


Figure 14. Building a match model.

10 Analyzing Results

Darwin provides six analysis tools for evaluating results. Five of these are available as commands on the **Analysis** menu: **Evaluate**, **Summarize Data**, **Frequency Count**, **Performance Matrix**, and **Lift Computation**. The sixth, **Sensitivity**, is an option on the dialog box for testing/evaluating a model. Table 4 provides a brief description of each.

This chapter describes the analyses. For the specifics of using the Darwin graphical user interface to execute these analyses, see *Using Darwin,* chapter 9. You can also use the **Evaluation Wizard** to perform these analyses.

Table 4. Analysis tools.

Evaluate	Evaluates the performance of a given model on a given dataset, when working on known data for test or evaluation purposes.
Summarize Data	Provides a statistical summary of the values taken by data in the specified fields of a given dataset.
Frequency Count	Provides information on the frequency with which particular data values appear in a dataset. (For ordered [continuous] fields, provide the information in the form of a histogram.)
Performance Matrix	Performs a more sophisticated version of a two-field frequency count.
Lift Computation	Provides values (numbers) for lift, margin, and ROI, showing the benefit of using a given model.
Sensitivity	Provides a graph showing the relative importance of attributes used in building a model.

10.1 Evaluate

This command evaluates the predictions made by one or more Darwin models. It works on a dataset that contains, for each model, both the original target field, with its actual values, and a field containing the predicted value for that field.

The *actual value* comes from the target field(s) in the dataset that was used for the practice prediction using the model. (This is the *source dataset*.) The *predicted value* comes from the output dataset of a prediction made on the source dataset.

To use the **Evaluate** command, you need both actual and predicted values within a single dataset. You create this dataset as follows:

1. Use the **Dataset** menu's **Transform** command **Merge** to merge the input and output prediction datasets, i.e., the original prediction subset of the original dataset and the output dataset created by the using the prediction command on that original prediction subset.

2. If you are comparing multiple predictions, use the **Dataset** menu's **Transform** command **Append** to append each additional model's prediction dataset (plus actual values, if they differ from those for the first model) to the new dataset.

3. Use the dataset resulting from the **Merge** command for evaluating a single prediction, or the dataset resulting from the **Append** command(s) for multiple predictions) as input to the **Evaluate Prediction** command.

When you invoke **Evaluate Prediction**, the dialog window asks you for the following information:

- **Dataset:** Name of the merged dataset.
- **Actual:** Name(s) of source field(s) (the field(s) that contain the actual values).
- **Predicted:** Name(s) of the target fields (the fields that contain the corresponding predicted values).

  Note: If you are evaluating a single prediction, you will specify a single pair of fields. If you are evaluating multiple predictions, you will supply more than one pair of fields.

If you want the information saved to a file, specify the name of an output file. Otherwise, the information appears on your terminal screen.

The **Evaluate Prediction** command compares predicted and actual results for all the pairs you have made and reports the maximum error, minimum error, the absolute error, and the root mean square error. The absolute error is the average of

the absolute value (1–norm). For binary target fields with the values 0 and 1, it is also the error rate.

The information is displayed in tabular form. If you are using more than one pair of fields, you can plot the results. You can also print either the table or the plot.

10.2 Summarize Data

This command examines specified data fields in your dataset and provides you with basic statistical information. It displays output in tabular form. Each field has one line of output, which displays

- **Name**: The name of the field.
- **Type**: The data type of the field.
- **Form**: Categorical or ordered.
- **Nulls**: The number of instances in the dataset in which this field has a null value.
- **Max Val**: The largest value of the field in this dataset.
- **Min Val**: The smallest value of the field in this dataset.
- **Avg Val**: The average of the field values in this dataset.
- **Avg Sq:** Used in calculating standard deviation.
- **Std Dev**: Standard deviation — a measure of the spread of values within the field; the wider the range of values, the larger the standard deviation.

Note: Average value and standard deviation are calculated on the full precision of the field values, not on the displayed values, which may have been rounded off.

To read all columns in the table, scroll the table horizontally. To read all rows in the table, you may need to scroll the table vertically, using the vertical scroll bar on the right side of the display.

10.3 Frequencies

This command provides information on the distribution of values within your dataset. It produces a table that shows the number of instances of each value that the field has within the dataset being analyzed.

If you have more values than you can handle in a categorical field, and if the values do have a numerical order, you can use the **Dataset** menu's **Transform** command **Set Form** to re-label the categorical field as an ordered field, and then perform a histogram on it.

You can perform a frequency count on a single field, or on two fields. If you use two fields, you get a 2-dimensional matrix, showing the number of instances in which a given value or range from field A appears in the same record with a given value or range from field B. For example, if your dataset records sales of a particular hat in colors `black`, `white`, and `tan`, and in sizes `small`, `medium`, and `large`, a 2-field frequency count of `Size` and `Color` would tell you the number of small hats in each color, the number of medium hats in each color, and the number of large hats in each color: nine bins in all.

You specify the granularity of the display in one of two ways, using options that are on the **Advanced Options** menu for **Analysis**:

- For ordered fields, specify the set of bins by giving the starting value for the first bin, the end value for the last bin, and the size of the step to be taken from bin to bin. For instance, if you expect values for a field to range from 32 to 68, and you wanted 4 bins, you might specify a start of 30, an end of 70, and a step of 10.

  Note: Darwin defines endpoints as values greater than or equal to minimum and less than the maximum, so you should always set your endpoint to a value larger than your highest expected value. For example, if the endpoints for a bin are A and B, a value x is in the bin if $x \geq$ A and $x <$ B.

- For categorical fields, you can simply specify the number of bins you want used. Darwin assigns one value to each bin.

  If you specify fewer bins than there are values, Darwin assigns the first N values to bins, then groups all other values under the heading *Other*.

  If you specify more bins than there are values, Darwin displays only as many bins as there are values.

Two-field frequency counts can be performed on any combination of ordered and categorical fields.

Two-field frequency counts can be handy for analyzing the correct and incorrect predictions made for each value within a target field. In this case, the two fields to be compared are actual and predicted versions of the same field, which you have excerpted from their original datasets using the **Transform** command **Select** and then merged to create a new dataset for analysis by this command and/or the **Lift** command.

10.4 Performance Matrix

Performance Matrix is like **Frequency Count**, with the following exceptions:

- It always expects two fields as input.
- It allows computation on a function of the value of each field, not just on the value itself. For instance, one function is the value of Field 1 minus the value of Field 2. You select the function for each field from a list.

Performance Matrix is used to compare fields, or simple functions of fields. It is similar to but somewhat more sophisticated than **Frequency Count**. For example, you can use **Performance Matrix** to compare the value of the field "minutes of usage" to the error (actual minutes of usage minus predicted) to see whether errors are more frequent for large usage levels than for small usage levels.

10.5 Lift Computation

Lift represents the benefit gained from a model's predictive ability. It can be thought of as the profits accruing from an accurately targeted campaign: for example, sending a direct mailing to only that 20% of customers who will provide 90% of the positive response.

Lift analysis tests predicted values against actual values in known data. Thus, the analysis can be performed on a test population, to predict lift, or at the end of some period during which the predictive model has been in use and the data resulting from its use has been recorded.

To measure lift, Darwin first sorts the population according to the predicted value of each record:

- Records predicted to be responders, with a high probability of correct prediction, come first.

- Records predicted to be responders, with a low probability of correct prediction, come next.
- Records predicted to be nonresponders, with a low probability of correct prediction, come third.
- Records predicted to be nonresponders, with a high probability of correct prediction, come last.

The population is divided into quantiles, with the user setting the granularity by choosing the number of quantiles. Darwin then compares the number of actual responders for a quantile against the number that would be expected if responders were distributed randomly throughout the quantiles. Calculations are cumulative: Thus, for a given quantile *N*, lift represents the results for all quantiles up to and including *N*.

When the lift provided by a good model is graphed, it forms a curve, with the correctly predicted responders beginning well above the random baseline at the left of the curve, and the two lines meeting at the end, when both include all the responders in the population.

When lift for a good model is expressed numerically, lift has large numbers in the first quantile(s), and ends with the number 1.

10.5.1 Target Expansion

You can correct for a difference in the composition of your testing/evaluation and prediction datasets by setting a value for the parameter **Target Expansion** (on the **Advanced Options** menu for **Analysis**. Expansion is calculated as the number of target records in the testing dataset divided by the expected (estimated) number of target records in the prediction dataset. The default is no correction.

10.5.2 Margin and ROI

Lift is purely a statistical measure that does not take costs or profits into account. Darwin's lift analysis lets users specify the profits or cost savings to be gained from a correct prediction, and the costs (and/or lost profits) associated with an incorrect prediction. It then uses these numbers to provide two measures of monetary benefit: *margin* and *return on investment* (ROI).

Margin is a measure of the net profit associated with the use of each quantile of population as predicted by the model. *ROI* is a measure of gross profit for the quantile: It sums the profit and the avoided cost for each correct prediction. For example, assume a company gains $20 in profit from each response to a direct

mail campaign, and loses $2 in costs for each nonresponder. Then, for each quantile, the margin for using that quantile is calculated on the basis of $20 per responder, while ROI is calculated on the basis of $22 per responder. Darwin's lift analysis provides both measures, allowing the user to pick the one best suited to each analysis.

10.5.3 Reading the Output

The output from **Lift** is a table, with one row for each quantile of the dataset and the following columns:

- **Lift:** The number of actual responders in this quantile divided by the number of responders in a random quantile. (This is cumulative, so the ratio in the bottom quantile is 1.)
- **Targets:** The number of records in this quantile with *actual* (not *predicted*) value of 1, where 1 is the positive target value.
- **Non-Targets:** The number of records in this quantile with *actual* (not *predicted*) value other than 1, where 1 is the positive target value.
- **Targets cum:** The total number of actual responders in this and preceding quantiles (cumulative) divided by the number of positive responders in the entire dataset.
- **Target Density cum:** The number of responders in the quantile divided by the total number of records in the quantile and above.
- **Target Density:** The number of responders in the quantile divided by the number of records in this quantile.
- **Margin:** The net profit you gain by using this quantile of data. Margin is calculated using the cost/profit information you supply for correct predictions.
- **ROI:** The gross profit you gain by using this quantile of data: that is, profit as a proportion of cost. ROI is calculated using the cost/profit information you supply for correct and incorrect predictions.
- **Confidence:** The cutoff accuracy level.

10.6 Sensitivity Analysis

Sensitivity Analysis measures the relative importance of attributes (or fields, variables) used in building a model. This analysis is not on the **Analysis** menu; it is available on the testing/evaluation dialog that you use in building a model.

The results of **Sensitivity Analysis** let you know which attributes are most and least important in your model. If, for example, you need to build a compact model, the results of this analysis can advise you about which attributes are your best predictors, which attributes you could safely leave out, and which ones are working against the prediction performance.

With all modules, **Sensitivity Analysis** compares the effect of "integrating out" (removing the effect of) each of the attributes for the model in terms of increasing or decreasing error rate.

The interpretation of the sensitivity scores depends on the dataset used in obtaining the sensitivity scores.

When a testing dataset (or any dataset other than the training dataset) is used, sensitivity scores may be positive, negative, or zero. A positive value indicates that the attribute is important in prediction; the larger the value, the greater the importance. A negative value indicates that the field contributes noise. A value of zero indicates that the attribute is irrelevant in prediction. The sensitivity score for each attribute indicates the relative increase or decrease in the prediction error if that attribute is removed (integrated out).

When a training dataset is used, sensitivity scores take only positive values. The value indicates the relative increase or decrease in the training error if the corresponding attribute is not used in prediction.

When the sizes of the training and testing dataset are about equal, the difference between the sensitivity scores obtained with training and testing dataset is proportional to complexity error caused by the attribute in question. With large sample size, the sensitivity scores converge to the same value.

As an example, consider these sensitivity values from a testing dataset for a Tree model:

```
              age    0.0306958
        workclass    0
           fnlwgt    0.118008
        education    0
    education-num    0.237381
   marital-status    0
       occupation    0
     relationship    0.295362
```

```
          race   -0.00272851
           sex    0
  capital-gain    0.331514
  capital-loss    0.130969
hours-per-week    0.0115962
native-country    0.0000132
```

The sensitivity value of 0.237381 for the attribute `education-num` means that this attribute is important in predicting. Specifically, the value means that the relative increase in the error resulting from removing (integrating out) that particular attribute will be roughly 23.7%. This means. for example, that if the model had a test error of 20%, then the effect of removing `education-num` would be to degrade the error rate to 24.7%.

The sensitivity values for the attributes `age`, `fnlwgt`, `education-num`, `relationship`, `capital-gain`, `capital-loss`, and `hours-per-week` are all positive, and not near zero, therefore they are the most important ones for the model.

The values for the attributes `workclass`, `education`, `marital-status`, `occupation`, `sex`, and `native-country` are zero, or near zero, which means that they are irrelevant. The variable `race` has a negative value, which means it contributes noise; however, the value is very low, which suggests this variable is more irrelevant than troublesome.

11 Generating Model Code

Darwin permits you to generate C, C++, or Java code for Tree and Net models so that you can include the models in application programs or embed them in Web applets.

11.1 Overview

Once you have perfected a Darwin model, you may want to embed it in an application program so that other users can take advantage of it. Darwin can generate C, C++, or Java code for a Tree or Net model so that you can call a prediction function from an application program written in C, C++, or Java.

You can also use the generated Java code to embed a model in a Web applet.

To generate code for a model, select **Code Generation** on the **Options** pull-down menu.

If the request to generate model code succeeds, Darwin creates, in the specified target directory, code for the prediction function and any required header files.

11.2 Requirement for Exporting Models

Code generation is not, by default, enabled. If you plan to export Darwin models, you must purchase Darwin Exported Model Packs. Contact your Thinking Machines Corporation sales professional or Thinking Machines Customer Support for information.

You will also need a license file, which you obtain from Thinking Machines Customer Support. They will email you a text file, which you simply add to the Darwin Client folder on any PC from which models will be exported.

11.3 Restrictions

The generated code performs the prediction on only one data record at a time.

Validation of input data is the responsibility of the calling program.

The generated code does not use any parallel processing even if the model was built using Darwin on a parallel processor.

11.4 Generating Model Code

Before you generate code for a model,

- select the model for which you wish to generate code
- check that the correct model type is activated (to generated code for a tree model, tree must be activated)

To generate code for a model, select **Code Generation** from the **Options** pull-down menu or press the **F8** key.

A dialog window appears and prompts you for the following information in two sections, **Using** and **Create Files**:

- **Using:**
  - **Model:** The name of the Tree or Net model for which you are generating code; the default is the current model.
  - **Subtree:** For Tree models only, the number of the subtree that you wish to use; the default is 0 (the full tree).
  - **Language:** The language of the exported code (C, C++, or Java); the default is C++.
  - **Function:** The name of the prediction function or class that will be generated; the default is *`modelname`*. If you are generating C or C++ code, you can specify the name of the prediction function. If you are generating Java code, the function is named `predict`; you can specify the class name.

- **Create Files:**
  - **Named:** The name of the created source file(s) that will be generated; the default is *`modelname.language`*, where *`language`* is `c` (for C), `cpp` (for C++), or `java` (for Java). The default name of the include (header file) is *`modelname`*`.h` for C or *`modelname`*`.hpp` for C++.
  - **Local**: Check this box if you want to write the generated files to folder on your PC. The generated files are always written to your project directory on UNIX.
  - The last field is the folder on your PC where the generated files will be written. If you want to change the default folder, click **Change**; a dialog appears that permits you to specify a different folder. (If you are generating Java code, no include file is created.)

11.5 The Generated Models

The generated model consists of two files if you are generating C or C++ code: an include (header) file and a file containing the prediction function, or one file (containing the generated class) if you are generating Java code. The exact format of the prototype for the predication function depends on the language in which the model is written. For the location of sample generated models, see section 11.6.

11.5.1 C++ Models

The generated model consists of two files: a header (include) file and a file containing the prediction function.

Tree Models

The header file consists of declarations for all the input arguments, three output arguments (`prediction`, `confidence`, and `rule_number`), and three error codes.

The error codes are:

```
#define DW_TREE_SUCCESS 0
#define DW_TREE_NULL_PTR –1000
#define DW_TREE_NO_VALID_RULE –1001
```

The prediction function has the following prototype:

```
int predict_funct   (type field1,
                     type field2
                      ...
                     type fieldn,
                     int& prediction,
                     double& confidence,
                     int& rule_number);
```

type is `int`, `float`, `double`, or `char`, depending on the field.

The prediction function consists of a list of `if` statements; a true statement sets values for `prediction`, `confidence`, and `rule_number` (node number) and returns.

The generated code requires only the standard C++ libraries to compile and link.

Net Models

The header (include) file consists of declarations for all of the input arguments, all outputs (targets) and a confidence for each target named `confidence_output-fieldname`. If there is only one target, the outputs are the target and `confidence`.

The prediction function has the following prototype, assuming that the target field is categorical and takes on *k* values:

```
void predict_funct        (type field1,
                           type field2
                           ...
                           type fieldn,
                           type& target_field_name,
                           double& confidence);
```

type is `int`, `float`, `double`, or `char`, depending on the field.

The prediction function creates the input vector; normalizes the input, and then calls `predict_network` to make the actual prediction. `predict_network` is nested `for` loops that use the weights to calculate a value.

The generated code requires only the standard C++ libraries to compile and link.

11.5.2 C Models

The generated model consists of two files: a header (include) file and a file containing the prediction function.

Tree Models

The header file consists of declarations for all the input arguments, three output arguments (`prediction`, `confidence`, and `rule_number`), and three error codes.

The following error codes are defined:

```
#define DW_TREE_SUCCESS 0
#define DW_TREE_NULL_PTR –1000
#define DW_TREE_NO_VALID_RULE –1001
```

The prediction function has the following prototype:

```
int predict_funct   (type field1,
                     type field2
                          ...
                     type fieldn,
                     int* prediction,
                     double* confidence,
                     int* rule_number);
```

type is `int`, `float`, `double`, or `char`, depending on the field.

The prediction function consists of a list of `if` statements; a true statement sets values for `prediction`, `confidence`, and `rule_number` (node number) and returns.

The generated code requires only the standard C libraries to compile and link.

Net Models

The header (include) file consists of declarations for all of the input arguments, all outputs (targets) and a confidence for each target named `confidence_outputfieldname`. If there is only one target, the outputs are the target and `confidence`.

The prediction function has the following prototype, assuming that the target field is categorical and takes on *k* values:

```
void predict_funct       (type field1,
                          type field2
```

```
 ...
type fieldn
type *target_field_name,
double* confidence);
```

type is `int`, `float`, `double`, or `char`, depending on the field.

The prediction function creates the input vector, normalizes the input, and then calls `predict_network` to make the actual prediction. `predict_network` consists of nested `for` loops that use the weights to calculate a value.

The generated code requires only the standard C libraries to compile and link.

11.5.3 Java Models

For Java code, the prediction function is wrapped in a class. You can specify the name of the class, but not the name of the prediction function, which is always named `predict`.

Tree Models

The generated Java class looks like the following:

```
/* DARWIN(tm) CODE FILE*/
class class_name
{
public int prediction
public double confidence;
public int rule_number;

/* Error Codes */
public static int DW_TREE_SUCCESS = 0;
public static int DW_TREE_NULL_PTR = –1000;
public static int DW_TREE_NO_VALID_RULE = –1001;


/*Prediction Function */
public int predict (input_args) {...};
}
```

The prediction function consists of a list of `if` statements; a true statement sets values for `prediction`, `confidence`, and `rule_number` (node number) and returns.

The input arguments are `field1`, ..., ; the output attributes are `prediction`, `confidence`, and `rule_number`.

The generated code requires only the standard Java libraries to compile and link.

Net Models

The generated Java class for a net with one target looks like the following:

```
/* DARWIN(tm) CODE FILE*/
import java.lang.Math;

class class_name
{
/* output fields */
public int target;
public double confidence;

/*private data and methods associated with the model */

/*Prediction Function */
public void predict (input_arguments) {...};
}
```

The prediction function creates the input vector; normalizes the input, and the calls the private method `predict_network` to make the actual prediction. `predict_network` is nested `for` loops that use the weights to calculate a value.

The input arguments are `field1`, ..., `fieldn`; the output attribute is `confidence`.

The generated code requires only the standard Java libraries to compile and link.

11.6 Examples of Generated Code

For examples of exported models, including an example of Java code that is embedded in a Web browser, see `http://www.think.com/models` on the World Wide Web.

Appendixes

A Data Design

This appendix* discusses how one might define dimensions of data for use in several types of applications. For illustrative purposes, it takes its examples from retail sales and from marketing, but the principles involved carry over to many other fields as well.

A.1 Retail Sales

In studies analyzing aspects of retail sales efforts, data dimensions may include all aspects that influence sales. The following list suggests some dimensions, together with some of the variables (that is, fields) that each dimension might contain:

- **Store**: ID, size, age, type of store
- **Location:** ID (country, region, city, etc.), plus demographic information
- **Product:** ID, brand, size, color, supplier
- **Price:** what you paid, what you received (retail, markup from wholesale), sale prices, recent price changes, competitor prices
- **Sales:** for current period, recent averages, recent upward or downward trends
- **Promotions:** campaign ID, cost, media, prominence, agency
- **Period/Season:** could be quarter, month, week within month, day of week, hour, periods preceding and/or following holidays, etc.
- **Staff:** names of managers, numbers and names of salespeople

* This appendix is based on information provided in the Information Harvester™ *A User's Guide to Data Mining*, by Ralphe Wiggins, Ph.D.

- **Economy:** for the area (employment, unemployment) or the customer population (disposable income, credit aversion, savings, etc.)
- **Weather:** either running statistics, such as temperature, precipitation, and wind chill, or general descriptions that allow the data mining tool to take note of weather that directly affects sales, such as blizzards, heat waves, etc.

Not all studies will include all these dimensions, and in some cases (such as weather and/or staffing) information may be unavailable. The following paragraphs suggest how various types of studies might use this data.

A.1.1 Forecasts

For sales forecasts, your choice of data dimensions will be governed largely by your selection of a *time frame* (hour, day, week, month, quarter, year?) and a *range*: that is, the number of future periods for which the forecast is desired. *Scope* also comes into play. If you are making forecasts for a chain of stores (or for some other grouping across a particular region), you will want to roll up sales numbers from individual stores and work with overall sales numbers.

Short-range forecasting depends almost entirely on seasonal effects (time within day, week, month, or year; days after or before the nearest local pay day or holiday; name of holiday) and recent sales history (yesterday, same day last week, same week last month, etc.)

Long-range forecasting, while still dependent on seasonality, is less dependent on recent sales trends and much more dependent on overall economic conditions: gross domestic product, disposable income, interest rates, savings, credit usage, employment, unemployment, age of population, and so on. Both absolute levels and trends may be useful.

In addition, *promotional activity* is an obvious influence on forecasts. You may compare particular promotions or agencies simply by name, or you may look at various properties of different campaigns (media, timing, etc.) to determine effectiveness.

A.1.2 Stock Selection

At the highest level, stock selection predicts relative sales for different departments — that is, the ratio of sales in a particular department to overall sales. The premise is that it is easier to first forecast the proportion of sales for each department, and then let the store-level forecast predict the overall sales level. Overall

economic conditions may affect the store-level forecast more strongly than they affect the way customers divide their money among different departments.

At the next level, stock selection concerns the brands, sizes, colors, and types of items that will sell best within departments at different locations. For these studies, one needs both data on the proportion of sales for each category or division being forecast, and demographic data about the customer population for each store. (Most frequently, the latter comes from census information.)

Stock selection studies may have either of two very different goals. One goal is to get a quantitative forecast of sales for each micromarket segment. The other goal is to understand what properties (of merchandise, stores, regions) drive sales, thus providing training information for buyers and store managers.

Cross-Selling

One specific area of stock selection concerns where items are displayed for best sales. What selection of products should a store or department offer? What products have a positive influence on the purchase of certain other products? Which have a negative influence? In general, these studies are carried out in hierarchical fashion. First, you determine which general classes of products influence other classes, then look at characteristics within a class (that is, what brands, colors, sizes, etc., influence which other brands, colors, sizes, etc.).

Other Studies

Data mining can also be used in retail applications to study patterns of store traffic, staff effectiveness, and promotion effectiveness. Studies can judge the overall impact of these items on sales, or they can do what-if analyses to design better promotions or traffic patterns.

A.2 Marketing

Marketing problems tend to deal with a smaller universe of influences than retail sales, but frequently work with huge amounts of data. For areas such as telecommunications or credit cards, this may mean millions of customers and/or large numbers of interactions with each customer. The data dimensions are often similar to those for retail sales: information on product, price, promotions, seasonality, demographics, economy, etc. However, more information is likely to be known

about each individual customer. Indeed, studies must take care to use only legally usable information about purchases.

Marketing problems frequently take the form of extrapolating the results from a test marketing population to a more general population. Thus, they compare each potential customer with customers who have already purchased a particular item or service.

- Studies on *cross-selling* compare each current client who has not purchased a given product with a set of current clients who have purchased that product.

- Studies on *loyalty* compare clients who have been repeat customers to those who have discontinued a service. These studies examine *dynamic information* about recent interactions with the customer (frequency, recency, and quality).

- *Fraud detection* combines all the above elements, comparing the properties of the general population of clients to those who are known to have committed fraud. These studies use basic marketing data to describe *foundation groups* of clients, then use dynamic behavior patterns to identify suspected fraudulent behavior within each foundation group.

  Note that this approach compares current behavior patterns with known examples of fraud; a current pattern that matches a known fraudulent pattern is suspicious.

  An alternative approach looks at historical data in which nonfraudulent behavior is the norm, and looks for behavior that departs from that norm. (Put another way, one looks not at the mean, but at the outliers.) The theory here is that anomalous behavior may indicate fraud.

B Glossary

Note: Italicized terms appear as entries elsewhere in this glossary.

Analysis menu — Provides commands that evaluate predictions, summarize information about datasets, calculate frequencies, calculate performance, and calculate lift.

attribute — A *field* in a Darwin dataset.

artificial neural network — See *neural network*.

backpropagation — The most popular form of *neural network*. Backpropagation networks are feed-forward multilayer networks that use a supervised training algorithm (backward propagation of errors) to adjust the connection weights.

byte — Eight bits.

categorical (field) — A *field* is categorical if it take on a finite number of unordered values; compare with *ordered (field)*.

classification — The act of predicting that a *record* belongs to a particular group. For example, in direct marketing, predicting that the record of one customer belongs to the group of records of people who would buy gourmet coffee. Classification is one of the two methods by which data mining makes predictions; the other is *forecasting*. The difference between the two is that classification predicts membership in sets, whereas forecasting predicts values within a series.

Classification and Regression Trees (C&RT) — A computer software technique that finds rules for making predictions by repeatedly breaking up historical examples of data into ever-smaller subgroups.

client	The part of Darwin where the graphical user interface runs; compare with *server*. The Darwin client runs on a personal computer.
clustering	The process of grouping similar input patterns together using an unsupervised training algorithm.
code generation	Generating C, C++, or Java code for tree and net models so that the models can be included in application programs or embedded in Web applets. Not all versions of Darwin permit code generation. Use the **Code Generation** command of the **Options** menu to generate code.
command log	See *macro*.
continuous (field)	Same as *ordered*. A *field* is continuous if it takes continuous values; compare with *categorical (field)*.
`darwin2sas`	A UNIX command that converts Darwin datasets to SAS files.
`darwinDG`	A UNIX command that automatically generates a descriptor file from a data (text) file.
Darwin software	A set of integrated tools that support data mining by creating *neural networks*, *match models*, and *tree models* using all the information contained in very large databases.
database	A self-describing collection of integrated *record*s.
data cleansing	A processing step during which missing or inaccurate data is replaced with valid values.
data mining	The process of applying intelligent algorithms, such as *artificial neural networks* or *C&RT*, to large collections of historical data to find patterns, predict trends such as customer behavior, and achieve accurate results that are not available using traditional methods. (See *traditional database processing*.)
dataset	In Darwin, a collection of *record*s sharing a common format, with a *dataset descriptor* as their header. All data is stored in datasets. Datasets are created as objects in virtual memory and may be saved as files.
dataset descriptor	A file containing information that allows Darwin tools to read and manipulate the *record*s in a Darwin *dataset*.

Dataset menu	Provides commands for creating, exporting, and transforming Darwin *dataset*s.
data warehouse	A large database where an organization's historical data is kept for long-term online storage.
dependent (variable)	In general, a variable whose value is determined by one or more *independent variable*s; the field whose value is to be predicted in a Darwin model. Also known as *target field* or response variable.
descriptor file	The file associated with a Darwin dataset that specifies the name and data type for each *field* in the dataset. Creating a Darwin dataset requires a data (text) file and a descriptor.
distributed dataset	A dataset that is a distributed across the processors of a multiprocessor machine.
distributed file	A file that consists of several components that reside on different processors in a multiprocessor system or a file that resides in a shared file space.
Edit menu	Provides commands for various text editing functions (undo, cut, copy, and paste).
Euclidean distance	The usual way of measuring distance between points in space based on the Phythagorean Theorem.
Evaluation Wizard	The Darwin *wizard* that guides you through the process of analyzing a model.
feed-forward	In a neural net, using the input values to calculate the output values, without using backpropagation.
field	The components of a *record*. Each field contains one or more items of data and becomes a variable for data analysis. A field is either *ordered* or *categorical*.
field form	In a dataset *descriptor*, a property of a field: either *categorical* or *ordered*.
forecasting	The method of predicting values via regression, and usually referred to as *regression*. Forecasting is one of the two methods by which data mining makes predictions; the other is *classification*. The distinction between the two is that forecasting predicts values within a series, whereas classification predicts membership in sets.

fuzzy logic — A method of reasoning that allows for partial or "fuzzy" descriptions of rules. For example, the truth of a proposition such as "Company X is a medium-sized company" might vary over a range of from "completely false" to "completely true."

genetic algorithms — Techniques that use the principles of genetics and evolution to make increasingly accurate predictions about a given database.

gigabytes of data — 2^{30} (approximately one billion) bytes (a byte is a unit of information consisting of 8 bits).

Help menu — Provides access to the Darwin online help and to information about the Darwin version.

independent (variable) — In general, one or more variables that determine the value of the *dependent variable*. In a Darwin model, one of the fields used to predict the *target field*.

input (layer) — The layer of a neural network that consists of the fields used to calculate the output.

input (for a model) — The datasets used to train the model.

importance — The relative contribution of an input attribute to the prediction of a particular model. The importance/sensitivity analysis performed by Darwin analyses a predictor function and a dataset producing a table that contains the input variables and their relative ranking. See also *sensitivity*.

k-nearest-neighbor algorithm — An algorithm that predicts values for a field using the values of the *k* records in the model nearest to the prediction record; nearness is measured using Euclidean distance.

knowledge discovery in databases (KDD) — The extraction of previously unknown information from a database; *data mining* is one phase of the KDD process.

level of confidence — Degree of certainty of result.

macro — A command log containing the commands issued during a Darwin session. You can execute the macro to reissue all the commands. You turn macros on and off using the **Macro** command of the **Options** menu or clicking the **Macro** icon.

massively parallel processing (MPP)
: A technology for doing parallel processing. MPPs can harness hundreds of processors to work together on a problem because of the way they are hooked together. This allows them to do *scalable computing*.

match model
: A model created by Darwin using a k-*nearest neighbors algorithm.*

Match Model mode
: The component of Darwin that creates models using a parallel weight-adjustable k-*nearest-neighbor algorithm*; to create a match model, click the **Select Match Model mode** icon or click **Match** on the **Model** menu.

MBR
: *Memory-based reasoning.*

megabyte
: 2^{20} (approximately one million) bytes (a byte is a unit of information consisting of 8 bits).

memory-based reasoning
: A technique for classifying records in a database by comparing them with similar records that are already classified.

menus
: The lists of commands in the Darwin user interface.

Model menu
: Permits you to select the type of model to create (tree, neural net, or match); provides commands to create, copy, test, and predict with Darwin models.

Modeling Wizard
: The Darwin *wizard* that guides you through the process of creating a Darwin model.

multiclass
: Refers to items, such as classification or prediction, that are associated with more than one class.

net model
: Same as *neural net model*.

neural networks
: Techniques that make predictions by analyzing the relationships among data elements in historical data. The name is derived from the fact that artificial neural networks are similar in structure to biological neural systems. Darwin uses this technique to create *neural net* models. (Also known as *artificial neural networks.*)

neural net model
: A model created by Darwin using the techniques of neural networks. Also known as net model.

Net Model mode — The component of Darwin that creates models using a parallel implementation of feed-forward *neural network*s; to create a neural net model, click the **Select Net Model mode** icon or click **Net** in the **Models** menu.

Open Database Connectivity (ODBC) — The relational database that Darwin can connect to and use to create datasets, described in *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide* (Redmond, Washington: Microsoft Press, 1994).

Options menu — Permits you to set and change advanced options, turn on *macro*s, and invoke *code generation*.

ordered (field) — A *field* is ordered if it takes ordered values; compare with *categorical (field)*. Ordered fields are sometimes referred to as "continuous fields."

output — The output of a model is the value or values being predicted (along with an indication of the confidence in the prediction).

parallel processing — A technology that allows a more efficient processing of information, just as mass production allowed a more efficient processing of manufactured goods. Rather than funnelling information through a single processor, like conventional "vector" computers, parallel computers send tasks through multiple processors simultaneously. Darwin supports *symmetric multiprocessors*, or SMPs.

perturbation — In the context of neural networks, refers to changing the weights (values).

platform — Computer hardware on which software runs; Darwin runs on Sun Microsystems and Hewlett Packard platforms.

prediction — In the data mining context, *prediction* refers to the use of information gained from some number of known values to estimate further values.

project — In Darwin, a collection of related datasets, models, and tables. All work in Darwin takes place in the context of a project. A project is either a directory or a directory that is also linked to a distributed directory.

Project menu — Contains the commands that manage projects, display reports and graphs, control database connectivity, and exit Darwin.

query (a database) — The act of retrieving information from a database, either as a list of *record*s or as a summary of the information in the records.

record — An element of a database that groups together a set of named values called *field*s. For example, in a credit card database, each cardholder would have one record. The fields would probably include name, address, income, and amount of last payment.

regression — The method of predicting values via regression (sometimes referred to as *forecasting*). Regression is one of the two methods by which data mining makes predictions; the other is *classification*. The difference between the two is that regression predicts values within a series, whereas classification predicts membership in sets.

Relational Database Management System (RDBMS) — A type of database or database management system that stores information in tables and conducts searches by using data in specified columns of one table to find additional data in another table.

`sas2darwin` — A UNIX command that converts a SAS file to a Darwin text file and descriptor.

scalable computing — The property of a parallel computer that enables the user to build a more powerful machine by adding more processors and storage capacity. Scalable machine designs allow you to expand to work with problems that are hundreds of times larger while running exactly the same software.

sensitivity — The contribution to a model from each attribute. The importance/sensitivity analysis performed by Darwin analyses a predictor function and a dataset, producing a table that contains the input variables and their relative ranking. See also *importance*.

serial dataset — A dataset that is not distributed, that is, a dataset that resides on one machine or one system of a multiprocessor system.

serial file — A file that is not distributed, that is, a file that resides on one machine or one system of a multiprocessor system.

server — (1) The part of Darwin where data mining algorithms run; compare with *client*. Darwin servers run on UNIX. (2) One or

more collections of executables, a daemon, and a configuration file to which users connect when starting Darwin. (3) The physical machine on which Darwin runs.

SQL — The standard language used to create, manipulate (including *query*), and control relational databases.

supervised learning — A training paradigm in which the neural network is presented with an input pattern and a desired output pattern. The desired output is compared with the neural network output, and the error information is used to adjust the connection weights; compare with *unsupervised learning*.

symmetric multiprocessing (SMP)
SMP is a technology for doing parallel processing.

target (field) — The *dependent variable* for the model; the field that the model predicts.

targeted marketing — The marketing of products to select groups of consumers that are more likely than average to be interested in the offer.

terabyte — 2^{40} (approximately one trillion) bytes or one million megabytes (a byte is a unit of information consisting of 8 bits).

traditional database processing
A statistical approach to analyzing databases that requires an analyst's intuition to devise hypotheses for predicting customer behavior, and then uses small samples of historical data to test the accuracy of the hypotheses. (See *data mining*.)

transformation — One of the operations performed on a Darwin dataset using the **Transform** operator to create a new dataset from an old one.

transformation dataset
A Darwin dataset that is created from an existing dataset using one of the *Transform command*s.

Transform commands
The Darwin commands used to create new datasets from an existing one by performing various operations such as normalization, projection, randomization, etc.

tree model — A Darwin model created using the *C&RT* algorithm.

Tree model mode	The component of Darwin that creates models using a parallel implementation of the *C&RT* decision tree algorithm; to create a tree mode, click the **Set Tree Model mode** icon or click **Tree** on the **Model** menu.
unsupervised learning	A training paradigm in which the neural network is presented with input data, and it self-organizes to cluster or segment the data by learning to recognize statistical similarities; compare with *supervised learning*.
user interface	The screens, commands, menus, buttons, dialogues, and functions through which a user communicates with a piece of software.
View menu	Provides commands to invoke the two wizards, view *workflow*, customize the Darwin workspace, and refresh the display.
Window menu	Provides commands to control the display of the Darwin window (cascade, tile, and arrange icons).
wizard	A program that helps you perform a task such as installing software or creating models. Darwin includes two wizards, the *Modeling Wizard*, which guides you through the creation of a Darwin model, and the *Evaluation Wizard*, which performs the evaluation of a Darwin model. You invoke the Darwin wizards from the **View** menu or by clicking the wizard's icon.
workflow	A graphical display of the work done in the current Darwin session. To display the workflow, click the **Workflow** icon. Once you've displayed the workflow, use the **Workflow** menu or the icons to manage the display (zoom in, zoom out, and change the view).
Workflow menu	Provides commands for examining the workflow (zoom in, zoom out) and specifying what is displayed. This menu appears only when Workflow is active.

Index

Index

See also appendix B, "Glossary," for an alphabetized list of terms and their definitions.

D

E

F

G

H

I

Q

R

S

T

U

V

W