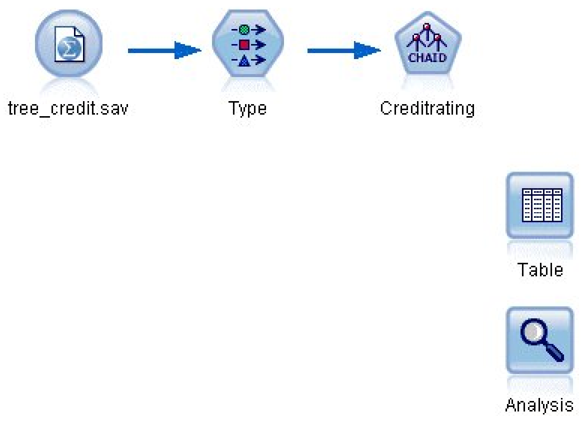
Building the Stream

Figure 1. Modeling stream



To build a stream that will create a model, we need at least three elements:

- A source node that reads in data from some external source
- A source or Type node that specifies field properties, such as measurement level (the type of data that the field contains), and the role of each field as a target or input in modeling.
- A modeling node that generates a model nugget when the stream is run.

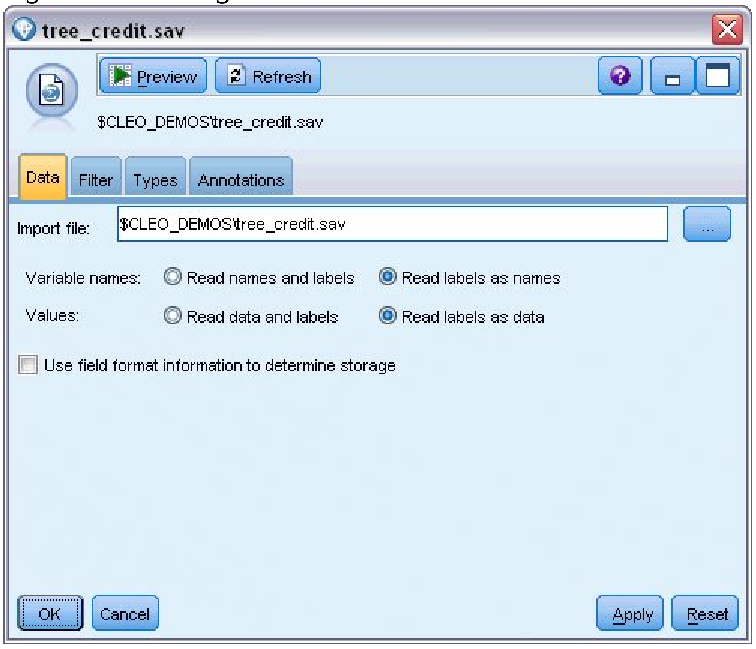
In this example, we're using a CHAID modeling node. CHAID, or Chi-squared Automatic Interaction Detection, is a classification method that builds decision trees by using a particular type of statistics known as chi-square statistics to work out the best places to make the splits in the decision tree.

If measurement levels are specified in the source node, the separate Type node can be eliminated. Functionally, the result is the same.

This stream also has Table and Analysis nodes that will be used to view the scoring results after the model nugget has been created and added to the stream.

The Statistics File source node reads data in IBM SPSS Statistics format from the *tree_credit* data file, which is installed in the *Demos* folder.

Figure 2. Reading data with a Statistics File source node



tree_credit.sav

Preview | Refresh

$CLEO_DEMOS\tree_credit.sav

Data | Filter | Types | Annotations

Import file: $CLEO_DEMOS\tree_credit.sav

Variable names: Read names and labels [✓ Read labels as names]

Values: Read data and labels [✓ Read labels as data]

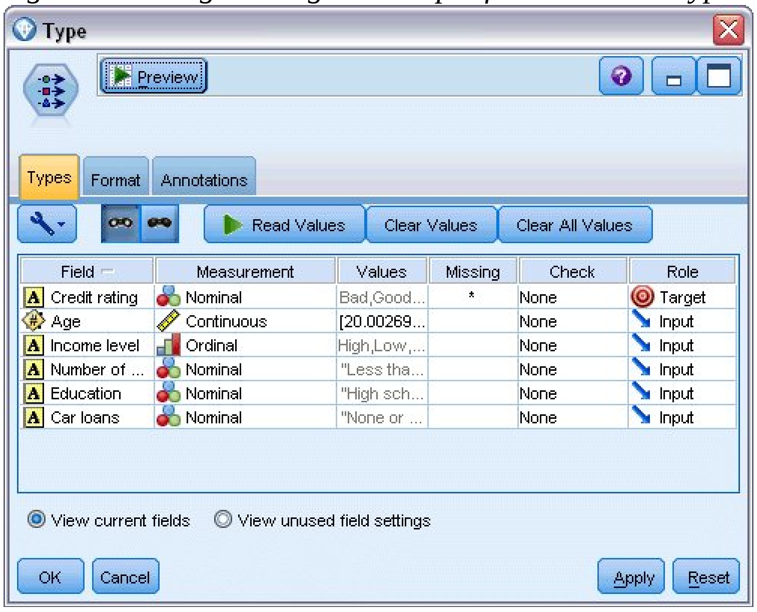
☐ Use field format information to determine storage

OK | Cancel | Apply | Reset

The Type node specifies the **measurement level** for each field. The measurement level is a category that indicates the type of data in the field. Our source data file uses three different measurement levels.

A **Continuous** field (such as the *Age* field) contains continuous numeric values, while a **Nominal** field (such as the *Credit rating* field) has two or more distinct values, for example *Bad*, *Good*, or *No credit history*. An **Ordinal** field (such as the *Income level* field) describes data with multiple distinct values that have an inherent order—in this case *Low*, *Medium* and *High*.

Figure 3. Setting the target and input fields with the Type node



Type

Preview

Types | Format | Annotations

Read Values | Clear Values | Clear All Values

Field	Measurement	Values	Missing	Check	Role
Credit rating	Nominal	Bad,Good...	*	None	Target
Age	Continuous	[20.00269...		None	Input
Income level	Ordinal	High,Low,...		None	Input
Number of ...	Nominal	"Less tha...		None	Input
Education	Nominal	"High sch...		None	Input
Car loans	Nominal	"None or ...		None	Input

[✓ View current fields] View unused field settings

OK | Cancel | Apply | Reset

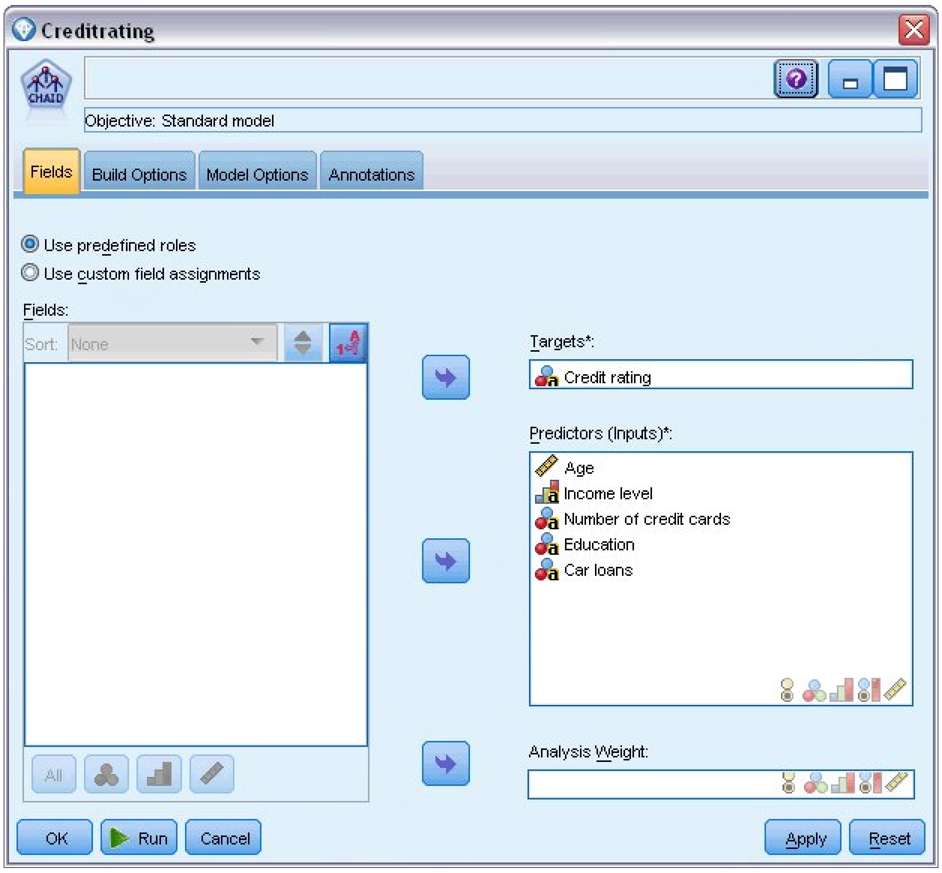
For each field, the Type node also specifies a **role**, to indicate the part that each field plays in modeling. The role is set to *Target* for the field *Credit rating*, which is the field that indicates whether or not a given customer defaulted on the loan. This is the **target**, or the field for which we want to predict the value.

Role is set to *Input* for the other fields. Input fields are sometimes known as **predictors**, or fields whose values are used by the modeling algorithm to predict the value of the target field.

The CHAID modeling node generates the model.

On the Fields tab in the modeling node, the option Use predefined roles is selected, which means the target and inputs will be used as specified in the Type node. We could change the field roles at this point, but for this example we'll use them as they are.

Figure 4. CHAID modeling node, Fields tab



Objective: Standard model

(● Use predefined roles) ○ Use custom field assignments

Targets*:
Credit rating

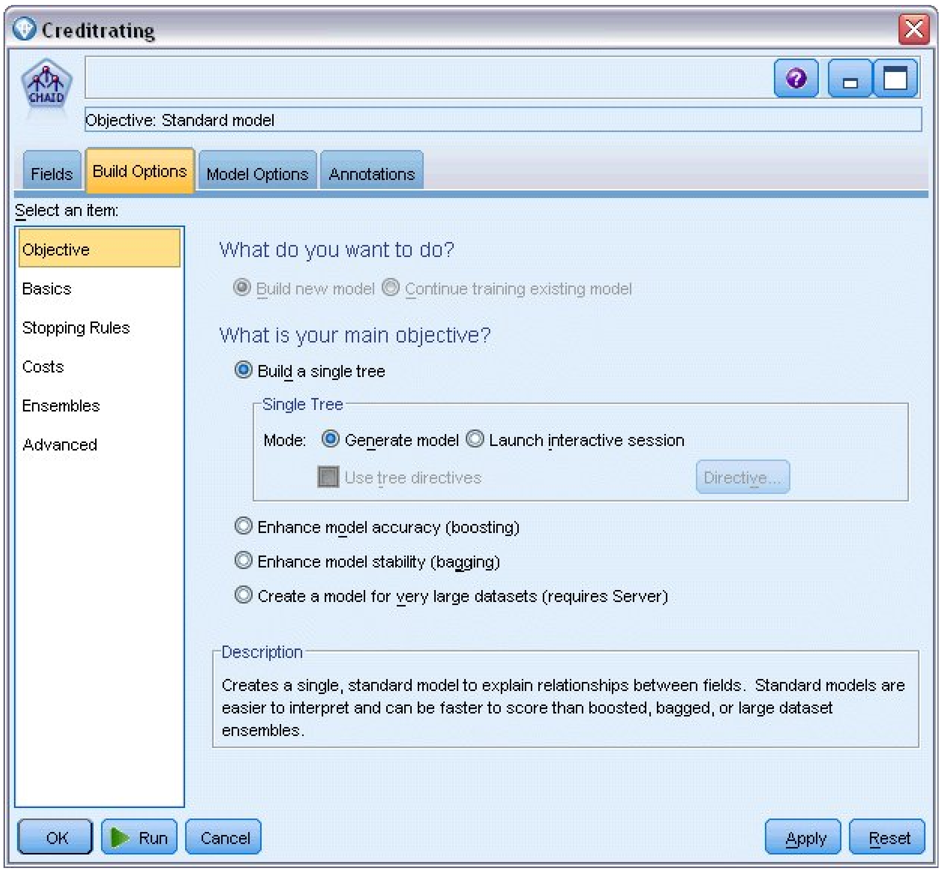
Predictors (Inputs)*:
Age
Income level
Number of credit cards
Education
Car loans

Analysis Weight:

We also just want a single, standard decision tree model without any enhancements, so we'll also leave the default objective option Build a single tree.

While we can optionally launch an interactive modeling session that allows us to fine-tune the model, this example simply generates a model using the default mode setting Generate model.

Figure 5. CHAID modeling node, Build Options tab



Creditrating

Objective: Standard model

What do you want to do?

[✓ Build new model] Continue training existing model

What is your main objective?

[✓ Build a single tree]

Single Tree

Mode: [✓ Generate model] Launch interactive session

Use tree directives

Enhance model accuracy (boosting)

Enhance model stability (bagging)

Create a model for very large datasets (requires Server)

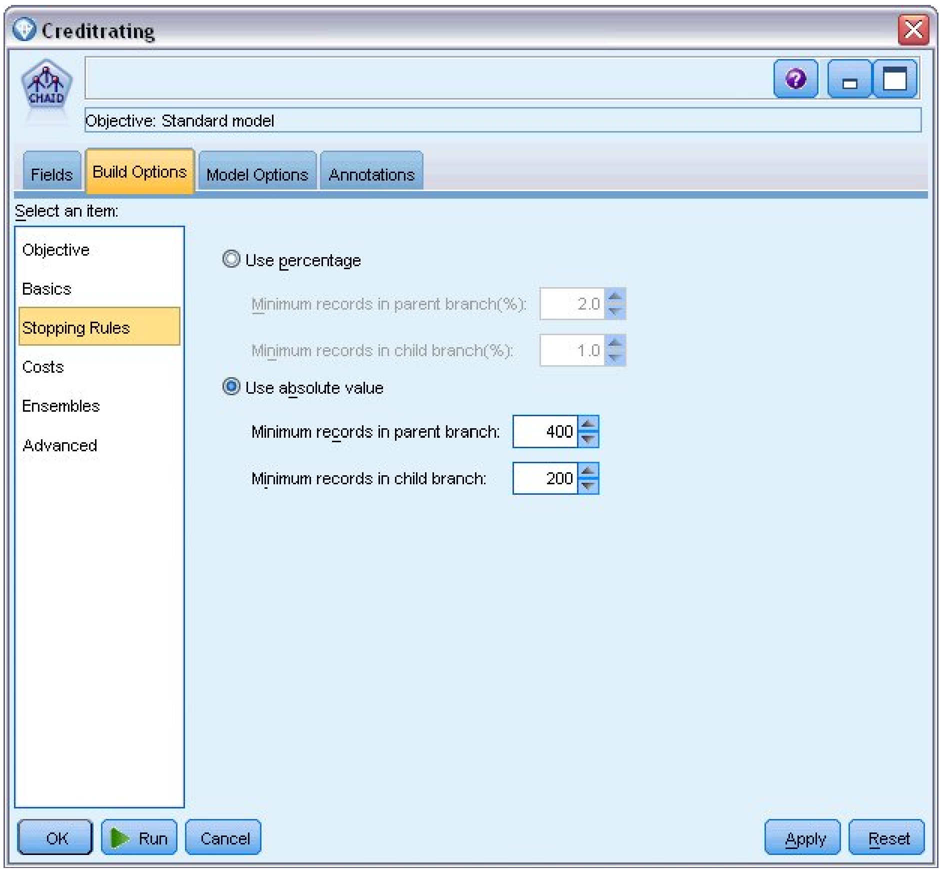
Description

Creates a single, standard model to explain relationships between fields. Standard models are easier to interpret and can be faster to score than boosted, bagged, or large dataset ensembles.

For this example, we want to keep the tree fairly simple, so we'll limit the tree growth by raising the minimum number of cases for parent and child nodes.

1. On the Build Options tab, select Stopping Rules from the navigator pane on the left.
2. Select the Use absolute value option.
3. Set Minimum records in parent branch to 400.
4. Set Minimum records in child branch to 200.

Figure 6. Setting the stopping criteria for decision tree building


Creditrating

Objective: Standard model

Use percentage

Minimum records in parent branch(%): 2.0

Minimum records in child branch(%): 1.0

[✓ Use absolute value]

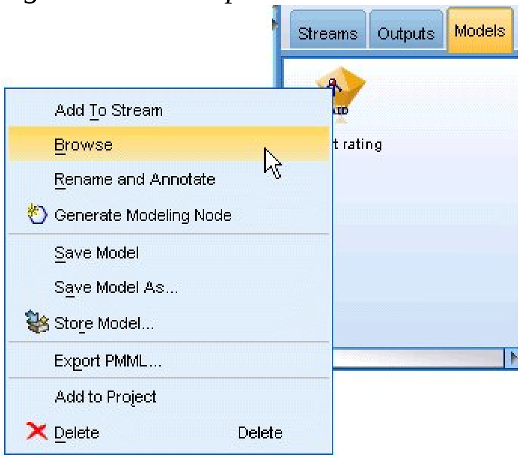
Minimum records in parent branch: 400

Minimum records in child branch: 200

Browsing the Model

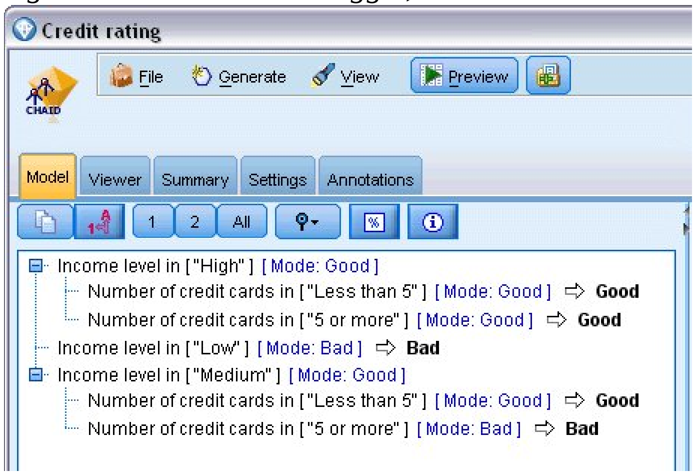
When execution completes, the model nugget is added to the Models palette in the upper right corner of the application window, and is also placed on the stream canvas with a link to the modeling node from which it was created. To view the model details, right-click on the model nugget and choose Browse (on the models palette) or Edit (on the canvas).

Figure 1. Models palette



In the case of the CHAID nugget, the Model tab displays the details in the form of a rule set--essentially a series of rules that can be used to assign individual records to child nodes based on the values of different input fields.

Figure 2. CHAID model nugget, rule set

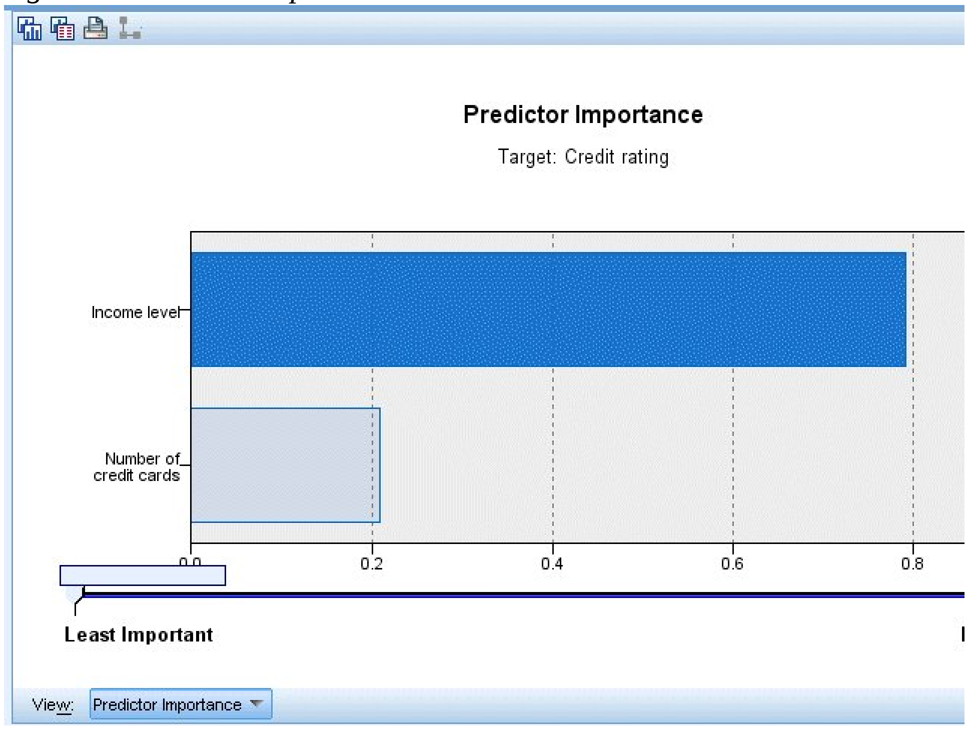


```
Income level in ["High"] [Mode: Good]
    Number of credit cards in ["Less than 5"] [Mode: Good] ⇨ Good
    Number of credit cards in ["5 or more"] [Mode: Good] ⇨ Good
Income level in ["Low"] [Mode: Bad] ⇨ Bad
Income level in ["Medium"] [Mode: Good]
    Number of credit cards in ["Less than 5"] [Mode: Good] ⇨ Good
    Number of credit cards in ["5 or more"] [Mode: Bad] ⇨ Bad
```

For each decision tree terminal node--meaning those tree nodes that are not split further--a prediction of *Good* or *Bad* is returned. In each case the prediction is determined by the **mode**, or most common response, for records that fall within that node.

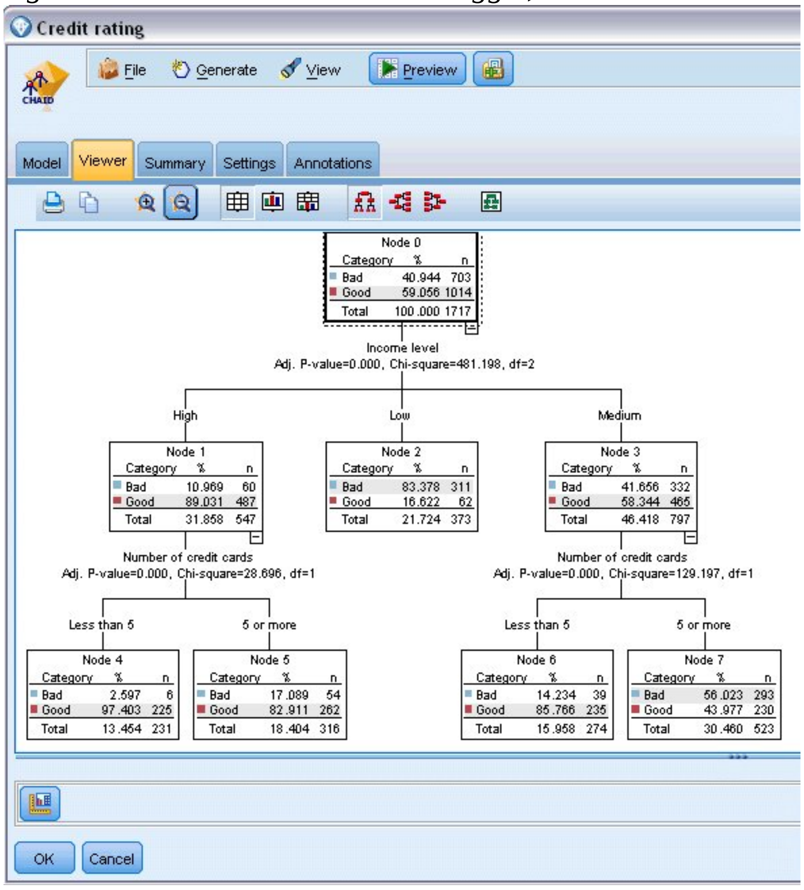
To the right of the rule set, the Model tab displays the Predictor Importance chart, which shows the relative importance of each predictor in estimating the model. From this we can see that *Income level* is easily the most significant in this case, and that the only other significant factor is *Number of credit cards*.

Figure 3. Predictor Importance chart



The Viewer tab in the model nugget displays the same model in the form of a tree, with a node at each decision point. Use the Zoom controls on the toolbar to zoom in on a specific node or zoom out to see the more of the tree.

Figure 4. Viewer tab in the model nugget, with zoom out selected



Node 0

Category	%	n
Bad	40.944	703
Good	59.056	1014
Total	100.000	1717

Node 1

Category	%	n
Bad	10.969	60
Good	89.031	487
Total	31.858	547

Node 2

Category	%	n
Bad	83.378	311
Good	16.622	62
Total	21.724	373

Node 3

Category	%	n
Bad	41.656	332
Good	58.344	465
Total	46.418	797

Node 4

Category	%	n
Bad	2.597	6
Good	97.403	225
Total	13.454	231

Node 5

Category	%	n
Bad	17.089	54
Good	82.911	262
Total	18.404	316

Node 6

Category	%	n
Bad	14.234	39
Good	85.766	235
Total	15.958	274

Node 7

Category	%	n
Bad	56.023	293
Good	43.977	230
Total	30.460	523

Looking at the upper part of the tree, the first node (Node 0) gives us a summary for all the records in the data set. Just over 40% of the cases in the data set are classified as a bad risk. This is quite a high proportion, so let’s see if the tree can give us any clues as to what factors might be responsible.

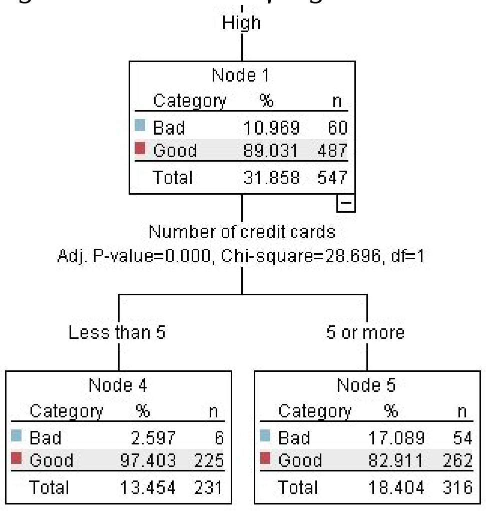
We can see that the first split is by *Income level*. Records where the income level is in the *Low* category are assigned to Node 2, and it’s no surprise to see that this category contains the highest percentage of loan defaulters. Clearly lending to customers in this category carries a high risk.

However, 16% of the customers in this category actually *didn’t* default, so the prediction won’t always be correct. No model can feasibly predict every response, but a good model should allow us to predict the *most likely* response for each record based on the available data.

In the same way, if we look at the high income customers (Node 1), we see that the vast majority (89%) are a good risk. But more than 1 in 10 of these customers has also defaulted. Can we refine our lending criteria to minimize the risk here?

Notice how the model has divided these customers into two sub-categories (Nodes 4 and 5), based on the number of credit cards held. For high-income customers, if we lend only to those with fewer than 5 credit cards, we can increase our success rate from 89% to 97%--an even more satisfactory outcome.

Figure 5. Tree view of high-income customers



Node 1

Category	%	n
🟦 Bad	10.969	60
🟥 Good	89.031	487
Total	31.858	547

Node 4

Category	%	n
🟦 Bad	2.597	6
🟥 Good	97.403	225
Total	13.454	231

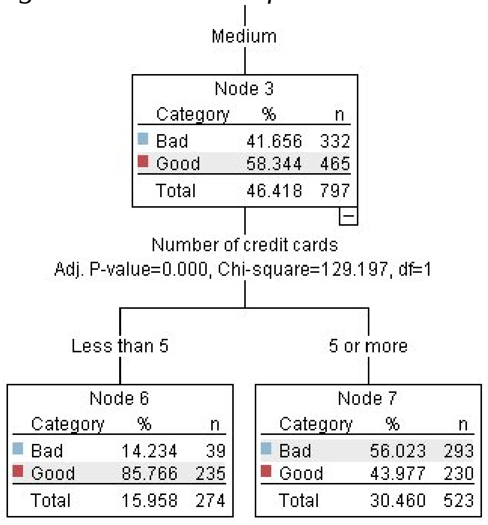
Node 5

Category	%	n
🟦 Bad	17.089	54
🟥 Good	82.911	262
Total	18.404	316

But what about those customers in the Medium income category (Node 3)? They're much more evenly divided between Good and Bad ratings.

Again, the sub-categories (Nodes 6 and 7 in this case) can help us. This time, lending only to those medium-income customers with fewer than 5 credit cards increases the percentage of Good ratings from 58% to 85%, a significant improvement.

Figure 6. Tree view of medium-income customers



Node 3

Category	%	n
🟦 Bad	41.656	332
🟥 Good	58.344	465
Total	46.418	797

Node 6

Category	%	n
🟦 Bad	14.234	39
🟥 Good	85.766	235
Total	15.958	274

Node 7

Category	%	n
🟦 Bad	56.023	293
🟥 Good	43.977	230
Total	30.460	523

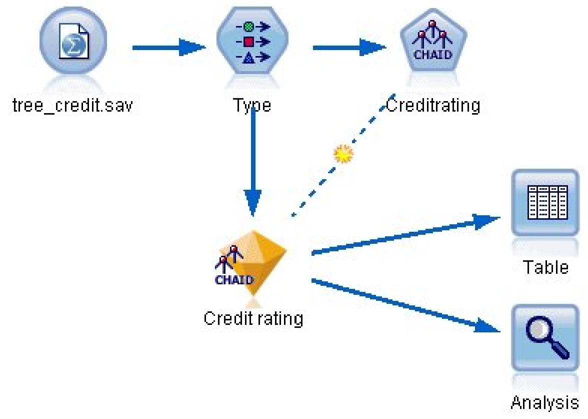
So, we've learnt that every record that is input to this model will be assigned to a specific node, and assigned a prediction of *Good* or *Bad* based on the most common response for that node.

This process of assigning predictions to individual records is known as **scoring**. By scoring the same records used to estimate the model, we can evaluate how accurately it performs on the training data—the data for which we know the outcome. Let's look at how to do this.

Evaluating the Model

We've been browsing the model to understand how scoring works. But to evaluate *how accurately* it works, we need to score some records and compare the responses predicted by the model to the actual results. We're going to score the same records that were used to estimate the model, allowing us to compare the observed and predicted responses.

Figure 1. Attaching the model nugget to output nodes for model evaluation

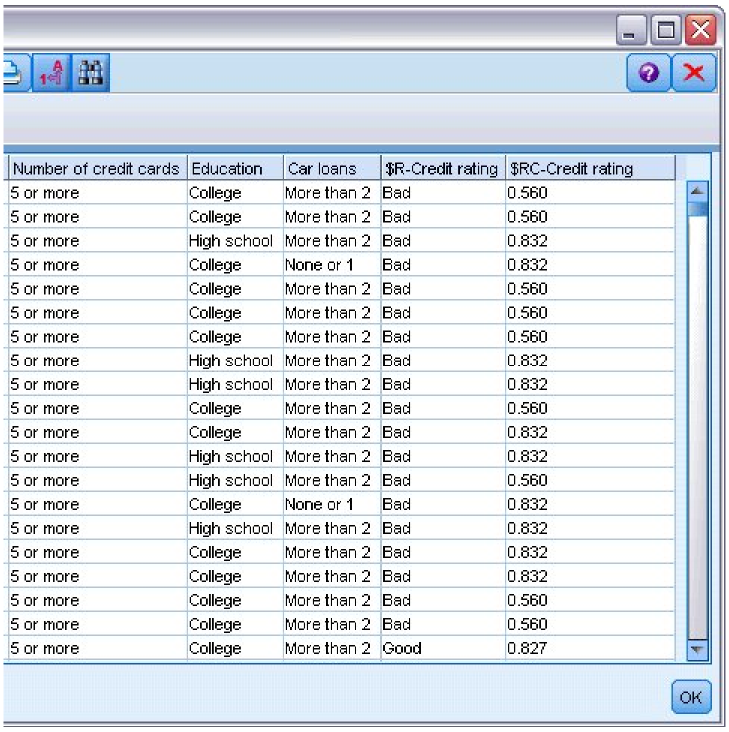


1. To see the scores or predictions, attach the Table node to the model nugget, double-click the Table node and click Run.

   The table displays the predicted scores in a field named *$R-Credit rating*, which was created by the model. We can compare these values to the original *Credit rating* field that contains the actual responses.

   By convention, the names of the fields generated during scoring are based on the target field, but with a standard prefix. Prefixes *$G* and *$GE* are generated by the Generalized Linear Model, *$R* is the prefix used for the prediction generated by the CHAID model in this case, *$RC* is for confidence values, *$X* is typically generated by using an ensemble, and *$XR*, *$XS*, and *$XF* are used as prefixes in cases where the target field is a Continuous, Categorical, Set, or Flag field, respectively. Different model types use different sets of prefixes. A **confidence value** is the model's own estimation, on a scale from 0.0 to 1.0, of how accurate each predicted value is.

Figure 2. Table showing generated scores and confidence values



Number of credit cards	Education	Car loans	$R-Credit rating	$RC-Credit rating
5 or more	College	More than 2	Bad	0.560
5 or more	College	More than 2	Bad	0.560
5 or more	High school	More than 2	Bad	0.832
5 or more	College	None or 1	Bad	0.832
5 or more	College	More than 2	Bad	0.560
5 or more	College	More than 2	Bad	0.560
5 or more	College	More than 2	Bad	0.560
5 or more	High school	More than 2	Bad	0.832
5 or more	High school	More than 2	Bad	0.832
5 or more	College	More than 2	Bad	0.560
5 or more	College	More than 2	Bad	0.832
5 or more	High school	More than 2	Bad	0.832
5 or more	High school	More than 2	Bad	0.560
5 or more	College	None or 1	Bad	0.832
5 or more	High school	More than 2	Bad	0.832
5 or more	College	More than 2	Bad	0.832
5 or more	College	More than 2	Bad	0.832
5 or more	College	More than 2	Bad	0.560
5 or more	College	More than 2	Bad	0.560
5 or more	College	More than 2	Good	0.827

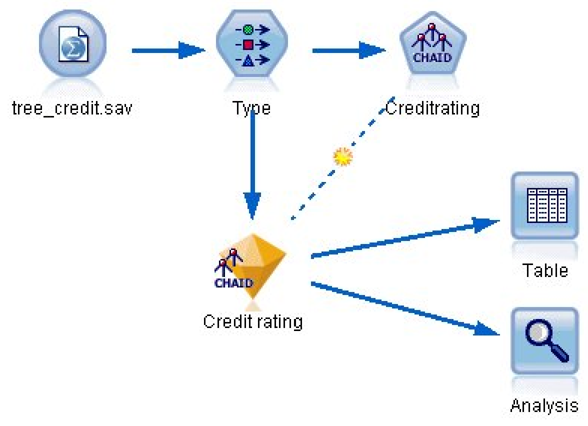
As expected, the predicted value matches the actual responses for many records but not all. The reason for this is that each CHAID terminal node has a mix of responses. The prediction matches the *most common* one, but will be wrong for all the others in that node. (Recall the 16% minority of low-income customers who did not default.)

To avoid this, we could continue splitting the tree into smaller and smaller branches, until every node was 100% pure—all *Good* or *Bad* with no mixed responses. But such a model would be extremely complicated and would probably not generalize well to other datasets.

To find out exactly how many predictions are correct, we could read through the table and tally the number of records where the value of the predicted field *$R-Credit rating* matches the value of *Credit rating*. Fortunately, there's a much easier way--we can use an Analysis node, which does this automatically.

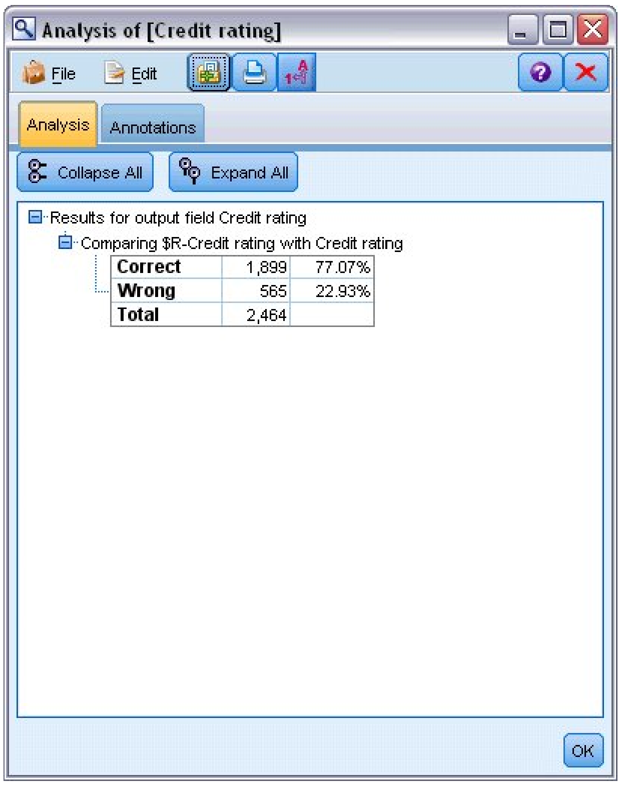
2. Connect the model nugget to the Analysis node.
3. Double-click the Analysis node and click Run.

Figure 3. Attaching an Analysis node



The analysis shows that for 1899 out of 2464 records--over 77%--the value predicted by the model matched the actual response.

Figure 4. Analysis results comparing observed and predicted responses



Results for output field Credit rating

Comparing $R-Credit rating with Credit rating

Correct	1,899	77.07%
Wrong	565	22.93%
Total	2,464	

This result is limited by the fact that the records being scored are the same ones used to estimate the model. In a real situation, you could use a Partition node to split the data into separate samples for training and evaluation.

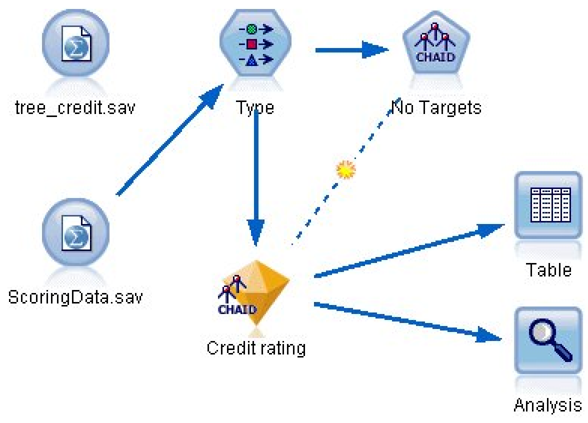
By using one sample partition to generate the model and another sample to test it, you can get a much better indication of how well it will generalize to other datasets.

The Analysis node allows us to test the model against records for which we already know the actual result. The next stage illustrates how we can use the model to score records for which we don't know the outcome. For example, this might include people who are not currently customers of the bank, but who are prospective targets for a promotional mailing.

Scoring Records

Earlier, we scored the same records used to estimate the model in order to evaluate how accurate the model was. Now we're going to see how to score a different set of records from the ones used to create the model. This is the goal of modeling with a target field: Study records for which you know the outcome, to identify patterns that will allow you to predict outcomes you don't yet know.

Figure 1. Attaching new data for scoring



You could update the Statistics File source node to point to a different data file, or you could add a new source node that reads in the data you want to score. Either way, the new dataset must contain the same input fields used by the model (*Age*, *Income level*, *Education* and so on) but not the target field *Credit rating*.

Alternatively, you could add the model nugget to any stream that includes the expected input fields. Whether read from a file or a database, the source type doesn't matter as long as the field names and types match those used by the model.

You could also save the model nugget as a separate file, export the model in PMML format for use with other applications that support this format, or store the model in an IBM® SPSS® Collaboration and Deployment Services repository, which offers enterprise-wide deployment, scoring, and management of models.

Regardless of the infrastructure used, the model itself works in the same way.

Summary

This example demonstrates the basic steps for creating, evaluating, and scoring a model.

- The modeling node estimates the model by studying records for which the outcome is known, and creates a model nugget. This is sometimes referred to as training the model.
- The model nugget can be added to any stream with the expected fields to score records. By scoring the records for which you already know the outcome (such as existing customers), you can evaluate how well it performs.
- Once you are satisfied that the model performs acceptably well, you can score new data (such as prospective customers) to predict how they will respond.
- The data used to train or estimate the model may be referred to as the analytical or historical data; the scoring data may also be referred to as the operational data.