

Modeling Methodology 2: Model Input Selection

by Will Dwinnell

Introduction

In this installment of the modeling methodology series, we explore model input selection. This is the process of choosing the input variables that will be used to create a model for making predictions and arriving at recommendations. When building models such as neural networks, it seems natural to assume that having more information is always better than having less. Instinctively, one would think that our model-building tool should do no worse if we add input variables because we have not removed any vital information. The reality of the situation is counter-intuitive: Adding inputs gives the model more things to consider, thus extra variables can confuse and dilute the outcomes.

Working with many variables expands the size of what is called the input space. The input space consists of all the possible combinations of input values. Assuming that the input variables are not correlated (a point we will return to), the addition of each new input multiplies rather than adds to the size of the input space. If we use a common rule of thumb, we need 10 examples for a one input problem and not 20, but 100 examples, for just a two input problem. With three inputs, the required number of samples jumps to 1000 (see figure).

Before long, this number grows quite large. Moreover, the number of required examples for uncorrelated inputs grows exponentially with the number of input variables. In the technical literature, this is termed (somewhat dramatically) the “curse of dimensionality.” You can read more about this problem in Hardle’s book *Applied Nonparametric Regression* (Cambridge Press).

Knowing this, we might reason that even if more inputs don’t help, we really shouldn’t do any worse with the extra inputs since we can always choose to ignore them. Although this is true in some sense, think about what this means in practice. Many of the inputs may be useless, but figuring out which ones to keep and which ones to drop is a daunting task.

With five inputs that can each be either used or ignored, there are 32 possible combinations of inputs. This means that an exhaustive search of all possibilities would require that 32 separate models be built and tested. With 10 candidate inputs, this number climbs to over one thousand! Few people who have worked with neural networks would relish the idea of waiting for this many models to finish training!

Several factors mitigate this problem to some extent. Real-world data sets tend to have some degree of correlation among their variables, and this reduces the effective dimensionality of the input space. Correlated inputs imply that the data will cluster in a subset of the entire space. How much this reduces the problem depends on the particulars of the data set.

It is also possible that the mapping from inputs to outputs is relatively simple and, thus, amenable to analysis or modeling with less than the theoretical maximum required number of examples. Nonetheless, many real problems start with a very large number of candidate inputs (hundreds or even thousands). Even if this number were reduced substantially, there would still be a large input space to cover.

Since practical experience clearly shows that paring down the number of inputs often results in models that are more accurate or more robust, it is necessary to find sound ways to reduce the quantity of candidate inputs.

Some Solutions

In general, reducing the number of inputs requires trial and error. We will need to try different combinations of inputs to find a good subset for our model to use. To assist in this searching process, several automated strategies have been developed, including exhaustive search, ordered search, genetic search, and heuristic search (among others).

Exhaustive search is the only method guaranteed to find the optimal subset for an arbitrarily complex problem. For most practical situations, however, this method is too slow. Still, if the number of inputs is reasonably small or you have time to wait, this may be a viable option. More than the other strategies, though, exhaustive search runs the risk of overfitting by building myriad models. Even if a single model avoids overfitting the data, the large number of models increases the chances of accidental overfit.

Ordered search involves systems like forward selection and backward elimination, which are often employed with multiple linear regression. A forward search starts by trying all possible models that use a single input. The best single input is retained and a search begins on the remaining candidate inputs to become the second input. The input that most improves the model is kept and so on. This process ends either when the model ceases to improve or we run out of candidate inputs.

A backward search works exactly like a forward search, except that it moves in the other direction. Backward searching begins with a model that uses all the inputs and then removes input variables one at a time. Interestingly, forward and backward searches may not result in the same set of inputs.

Many variations on these searches are available. In statistics, stepwise regression is a popular combination of forward and backward searches. Stepwise selection can either add or remove variables at any given stage of the search. Many statistics packages (Minitab, SPSS, SAS, etc.) support these types of searches. They involve a relatively small number of model builds, so they are not too slow to conduct.

When used with nonlinear models, stepwise regression reduces the risk of overfitting the model to the data. For nonlinear problems, though, these searches can be weak (meaning that they fail to find good combinations of inputs). For the simplest forward and backward stepwise searches, variables are considered only one at a time and no backtracking is allowed (once a variable is selected for inclusion or deletion, the decision is irreversible).

Genetic Search is a procedure driven by genetic algorithms (GA) -- powerful systems that are very good at handling difficult optimization problems. GAs cycle through many iterations and, within the context of input selection, require more stringent testing to ensure that they haven't accidentally stumbled onto a bad solution which merely looks good.

Anecdotal evidence indicates that genetic searches are very good at finding useful combinations of inputs. One modeling tool that supports this type of search on model inputs (as well as forward, backward, and exhaustive searches) is PRW (Pattern Recognition Workbench) from Unica.

Heuristic Search modeling systems (like CART and ModelQuest) perform their own input selection as part of the modeling process. Symbolic machine learning systems (like those that search for IF...THEN rules) do this implicitly in their selection of variables to be used in the IF side of conditional statements. Other systems provide numerical variable importance measures, which are something like correlation or sensitivity measures in statistics.

Such modeling systems can be operated either on their own or as variable selectors for other modeling systems. Vendors of such tools occasionally give examples of using their products as input selectors for neural networks. The remainder of this article will concentrate on this hybrid approach using an example as an illustration.

An Example

We will compare the performance of a neural network on a sample problem using all available inputs and only a subset of the available inputs. Selection of the inputs will be performed by a tree-induction system. Variables used in the finished decision tree will be considered selected and become the inputs for building the reduced neural network.

The data set is a synthetic one with 2500 examples, 10 real input variables, and one binary output variable (a classification problem). The 10 input variables consist of normal, exponential, and uniform pseudorandom numbers, of which only three are relevant to the output. The cases are divided more or less evenly between output 0 and output 1.

The relationship between the inputs and the output is noise-free and there are no missing values. Of the 2500 examples available, 2000 will be used in building the neural network and 500 will be employed as a validation set to assess model performance. Of the 2000 examples used in building the models, 1800 will be used for training and 200 will be used for determining when to stop training.

The neural network development tool is BrainMaker Pro v3.1, from California Scientific, with most settings at the default (a few were changed: heavy weights, automatic learning tuning, and automatic tolerance tuning were turned on). For variable selection, CART for Windows v2.5.1, from Salford

Systems, is used with the default settings. CART is a capable tree-induction algorithm that is well-known in both the statistical and machine-learning fields.

The hardware platform is a 486 PC running at 120MHz with 32 megabytes of RAM. The first neural network, which uses all available inputs, has a topology of 10-70-1. This number of hidden neurons allows for enough representational power in the neural network to solve the problem. Choosing some other number of neurons should not affect the results for our purposes unless so few hidden neurons were employed as to cause underfitting.

In this experiment, the optimal CART tree used three of the variables (as it turns out, the three that are actually known to be relevant). Another neural network, with architecture 3-70-1 was trained on these three selected variables. Both neural networks were trained for 5000 passes through the data.

The following data indicate neural network performance (number of training passes and time in minutes) for the optimal network, as measured by root mean squared error:

	Passes	Time
Neural Network, all inputs	2340	159 minutes
Neural Network, CART-selected inputs	2640	132 minutes

Given that CART took about four and a half minutes to run, this would result in a net time savings of approximately 15%, ignoring other factors like set-up time. Notice that the second neural network completed more training passes through the data even though it took less time since its total number of connections (351) is less than that of the first network (841).

The neural network's output is actually a continuous value between 0.0 and 1.0. If the network output is 0.5 or less, then the output is interpreted as class 0, otherwise, it is interpreted as class 1. Here are the network performance results in terms of accuracy on the 500 validation examples:

	Good	Bad
Neural Network, all inputs	493	7
Neural Network, CART-selected inputs	500	0

How acceptable are these results? The second network appears to have done better with 100% accuracy on the validation set versus 98.6% accuracy for the first network, but a further analysis is necessary.

Testing the Results

The model using less inputs appears to be more accurate, but how do we know that the difference is significant? After all, the change may be accidental. What is required is called a significance test in statistics. Conventionally in statistics, this involves assuming that the variables in question (in this case, the errors of the models) follow some known probability distributions (most often a normal distribution), which can then be easily analyzed mathematically. Many real world variables, however, do not obey such convenient distributions, so the results of this type of analysis can be questionable.

Instead of the traditional form of significance testing, we will use a newer method called bootstrapping. Bootstrapping works for many real world variables that do not conform to special distributions. Further, regardless of which distributions are being analyzed, the bootstrapping method is always applied the same way. In contrast, different formulas are required for each distribution cataloged by statistics, resulting in a lack of testing uniformity.

A complete explanation of bootstrapping is beyond the scope of this article, but the basic idea is that the data is resampled and tested (as in a Monte Carlo simulation) repeatedly. The more times the data is sampled this way, the better the results of the bootstrapping process.

Thus, the cost of bootstrapping is the tedious work of sampling the data many times. With computer technology bringing cheaper and cheaper computer power, this cost becomes less of an issue. Today, even a cheap (<\$1000) desktop PC is capable of performing useful bootstrap analysis.

How does all of this apply to our work with modeling? We can apply this technique to the validation results to see if there is a significant difference between the results obtained from the two neural networks. To do this, we can use a program called Resampling Stats, from a company of the same name.

Using this software, 15,000 simulated (resampled) experiments were run and only 14 (0.0093%) of them had the first neural network getting all 500 validation examples correct. From this, it is possible to establish that the neural networks are behaving differently, with improved accuracy from the second neural network.

In this particular case, there is a relatively easy way to perform the significance test using a statistical formula. For other situations, however, the story is different.

A pair of models with numeric outputs can be compared in terms of their mean errors without difficulty using the conventional approach. That requires implying an assumption which may or may not be true. The bootstrapping technique would be far more accessible than its conventional counterpart if, for instance, we were comparing the median of continuous errors of two models.

It is worth noting that even had the performance of the two models been similar, we still might be interested in the model requiring less inputs. Why? Because in many real-world cases information is not free (nor even cheap!).

For example, if this problem involved monitoring mechanical equipment in a factory, it would cost more money to set up the machinery with all 10 sensors than it would to install only three of them. Other situations are similar, such as organizations that often buy their data from the government, demographic sources, credit rating bureaus, and so forth. If model performance can be maintained in the face of reduced sets of inputs, model cost can be likewise reduced.

The technical literature records many cases where models using a subset of the available input variables are more accurate, faster to train, more robust, or some combination of these. It is worth learning and understanding techniques for quickly and effectively performing this subset selection.

Conclusion

Things you should take away from this article:

- Despite popular belief, as few as 10 input variables can be quite a lot to work with. Less can indeed be more.
- The corollary to the above is that (also despite popular belief) as many as millions of examples can be insufficient to fill a large input space.
- Care must be taken when searching for good inputs not to overfit the data by building too many models.
- There exists a wide array of methods for reducing the size of the input space, and the heuristic approach often strikes a nice balance between optimality and speed.
- Bootstrapping offers advantages over conventional techniques in the form of simplicity and broad applicability, at the (rapidly diminishing) cost of computer power. Where applicable, however, conventional statistical methods do offer a concise formula.
- Even models that perform no better with input selection may be of great interest given the potential for lower cost.

The data file used in this article may be found on the *PC AI* Web site at www.pcai.com/pcai.

Will Dwinnell is a consultant and writer in the area of machine learning and optimization. He may be reached at predictor@compuserve.com.

Figure Caption:

Variables can be thought of as geometric dimensions, as in a scatterplot. To maintain a given density of coverage in higher dimensional problems, exponentially increasing numbers of samples need to be acquired. Here, we see that using 10 samples per dimension causes our data requirement to jump from 10 to 100 samples. Even with lower density of sampling per dimension the required data grows rapidly as the number of dimensions (variables) increases.

Sidebar: Terminology Confusion

Unfortunately, terminology in modeling overlaps that used in other fields. People who work in fields like statistics and database design apply different meanings to the same terminology.

input variable:

independent, explanatory, regressor, or predictor variable (statistics)

output variable:

dependent or response variable (statistics)

variable:

field, column (database)

dimension (geometry, mathematics)

example:

record, row (database)

case, sample, observation, individual (statistics)

model parameter:

degree of freedom (engineering, control theory)

regression coefficients (statistics, mathematics)

weights (statistics, neural networks)

To add to the mix, confusion frequently arises regarding the most basic of terms -- input and output. While input is most often used to refer to the input (independent) variables themselves, it also sometimes is used to mean entire example patterns (independent and dependent variables). Why? Because the entire example is an input of the model-building process.

Output, on the other hand, refers alternatively (and confusingly) to either the dependent variables or the model output that is intended to approximate that particular dependent variable (which are two distinct things). The neural network literature makes use of the terms pattern, target, and actual variables in this context.

Additionally, some vendors have introduced new terminology of their own. ModelWare from Triant (formerly Teranet IA) refers to all variables as

elements, because its UPM modeling algorithm does not distinguish between variables as inputs and outputs. Active elements are model inputs in the sense that they contribute to the modeling process.

Passive elements, in contrast, do not. Although one might think of ModelWare's active elements as input variables, predictions are made on them, as well as, the passive elements. ModelWare calls examples images -- likely stemming from the software's process control origins. In other words, each example could be thought of as an image of the system at some particular point in time.

Vendors of commercial modeling tools also often invent their own measures of association among variables (like correlations in statistics) and measures of the importance of input variables (something akin to sensitivity analysis in statistics).