ACAMS Anti-Money Laundering Technology RESOURCE DIRECTORY

Transaction Monitoring Under The Hood: Decision Trees, Neural Networks, and Statistical Profiling

by Ed Levy, J.D.

In this article, we will briefly explore the inner workings of some of the more advanced software approaches used by transaction monitoring systems to detect money laundering. If you are in the market for one of these sophisticated, expensive systems, it is important to have some understanding of the underlying technology, if for no other reason than to get past the tag-lines.

For example, you will hear that neural networks are black boxes that don't give the user any understanding of why they are singling out a transaction as suspicious. In fact, the offerings of the market-leading vendors of all of the advanced transaction monitors have engineered their systems to address the limitations of simplistic academic implementations, such as the black box limitation of a simple neural network. In order to evaluate an offering, it is necessary to understand how the system implements the technology it uses and how the implemented system works in the real world. No system can be judged superior simply on the basis of whether it calls its technology decision tree, neural network, or statistical profiling. Still, if you decide to use transaction monitoring software, the technological approach the software uses to detect money laundering is a factor you should take into consideration when choosing which system to use.

An effective anti-money laundering strategy is comprised of policies to satisfy "Know Your Customer" and Enhanced Due Diligence requirements, training and awareness programs, and reporting and record-keeping procedures. Software systems should be seen as one tool to use as part of this comprehensive strategy.

Software vendors offer systems that check potential clients and counter-parties against lists of known prohibited and high-risk individuals and entities (e.g., lists such as the OFAC list and databases of Politically Exposed Persons). Other vendors offer systems that check against rules that are packaged with the system; most of these allow you to customize rules for your institution. The vendors that offer the transaction monitoring approaches based on decision trees, neural networks, and statistical profiling incorporate list checking and rule processing, but they go even further than that. Whether or not the advanced technological approaches offered by these vendors are appropriate for your institution is a question you need to address as a part of your overall anti-money laundering strategy.

This article discusses decision trees and neural networks, two of the techniques used in what is sometimes called data mining or knowledge discovery. It also discusses statistical profiling. The vendors that use these approaches claim that their

Ed Levy, J.D., is a member of the ACAMS Technology Task Force. He is a Business Analyst, System Architect, Data Modeler, and Project Manager working in the anti-money laundering area. For questions or comments, email edlevy@impactaml.com.

software can discover new methods of money laundering on their own, even when rules that identify those cases haven't specifically been programmed into them. I hardly blame you if you're skeptical, but this article should give you some sense of how this might be possible.

The utility of these sophisticated approaches lies at least as much in their day-to-day role in detecting existing patterns of money laundering as in their detection of new patterns. If computers are able to recognize all the known patterns, without detectiing new ones, they can still compare transactions as they happen with a wealth of historical data and analyze the patterns in near-real time, in a way that would be completely beyond human capacity. We are focusing on learning new patterns because it will allow you to understand both how these systems learn new patterns and how they detect existing patterns. The first two approaches we'll discuss, decision trees and neural networks, are derived from the field of machine learning.

Can machines learn? Deep Blue, the computer that beat Kasparov at a regulation chess match in 1997, is one example that made the headlines. Computers have been built that have learned to analyze astronomical phenomena, to predict recovery rates in pneumonia patients, to perform insurance risk-management analysis, and even to use vision sensors to drive vehicles on public highways at up to 70 miles per hour (hopefully not while I'm on the road). One of the areas in which machine learning approaches have been applied very successfully is the detection of credit card fraud. A number of the companies that are now using machine learning to detect money laundering started by using it for fraud detection.

Since decision trees and neural networks as they are applied to the detection of money laundering are operating in the same domain, they face similar problems, including issues surrounding selection of independent variables, training, sample selection, overfitting, and noisy data. In many ways, they are more alike than they are different. Although statistical profiling is not strictly a machine learning approach, in the field of money laundering

Machine Learning versus Artificial Intelligence

So much emotionally charged controversy has surrounded the field of Artificial Intelligence that it has been divided into two branches: strong AI and weak AI.

Believers in strong AI claim that a computer could be built that would think at (or even beyond) the level of humans and maybe even be conscious of itself, as HAL was in the movie 2001. ("I'm sorry, Dave. I can't let you do that".)

In 1950, Alan Turing proposed the Turing Test, related to strong AI. His operational definition of artificial intelligence was the ability of a computer to achieve a level of performance in all cognitive tasks such that it could fool a human investigator communicating with it by teletype into thinking that it was human. To pass this test, a computer would need all of the following abilities: the ability to process natural language, the ability to represent knowledge so that it could store information given to it during the investigation, the ability to reason so that it could answer questions and draw new conclusions based on information given to it by the investigator, and the ability to learn.

Note that the Artificial Intelligence abilities that a computer needs to be useful in detecting money laundering are much more limited than those required to pass the Turing Test. It will be acceptable to most of us for the computer to report suspicious transactions in a form that does not require it to have a conversation with us that would fool us into thinking it was a person!

Believers in weak AI claim that computers can be built with thinking-like abilities. This is the kind of artificial intelligence we are concerned with: the ability of the computer to adapt to new circumstances and to detect and extrapolate patterns, that is, to learn. The claims of believers in weak AI that computers can learn in this sense have already been proven correct.

detection, it too faces most of the same problems.

The general framework for the understanding of machine learning approaches immediately following this section lays out the issues. Separate sections on decision trees, neural networks, and statistical profiling will outline how each of these approaches handle the issues.

A General Framework for Understanding Machine Learning Approaches

Specification of a Well-Defined Learning Problem

We will define **learning** as improvement of performance with experience, or, more formally:

Learning: A software system learns from experience if it is able to improve its performance on a task with experience in performing the task, without outside intervention.

In the example of the car that learned to drive on the highway by itself, the steering system and vision sensors of the car started out knowing nothing. Through exposure to a variety of driving scenarios represented by visual stimuli and feedback on the correct actions to take given those scenarios, the system eventually trained itself to steer so that it could successfully drive on the highway.

An anti-money laundering learning task could be expressed in this framework as follows:

Task: Recognizing cases of money laundering;
Performance: Percentage of successfully classified cases; and
Experience: Exposure to a set of cases with the cases that constitute money laundering pre-identified.

The idea is that having learned how to classify the pre-identified cases, the software system can correctly classify not only cases that are identical to those it has been exposed to, but also cases that are different. A key point is that the software system is not simply memorizing the specific cases that it has been presented. Rather, it is developing a method by which to classify future cases.

Before going further, we'll refine the specification of the task. A correct task specification is one of the keys to successful machine learning. It's unrealistic to expect that the computer is going to definitively recognize cases of money laundering. There are too many factors that could never be included in the data to make a definitive identification. Instead, we could define the task as identifying cases in which a Suspicious Activity Report (SAR) should be filed. Even this is too ambitious. We can't expect to have so much confidence in our program that we will allow it to automatically file a SAR without human intervention. A more realistic goal is recognizing cases in which an expert would say that the institution's policy would require that someone should investigate whether a SAR should be filed. (In fact, many of the software systems reverse their viewpoint on the problem and, rather than learning to recognize the case of suspected money laundering, they learn to recognize normal patterns of activity; then they report on departures from those normal patterns. We will discuss the advantages of this after illustrating how decision trees and neural networks solve the learning problem.) Meanwhile, our new specification of the **Task** is to recognize whether a transaction should be investigated.

We will consider a number of important issues involved in building software that could successfully learn the task. The first set of issues involves the organization of the experience that the software will train itself on.

☞ (continued on page 4)

Association of Certified Anti-Money Laundering Specialists™
ACAMS

First Ever CAMS Certification Exam!
***Deadline to Apply is February 12th**
Miami Beach, Florida
March 26, 2003

DesignofTheTrainingExperience: Variables, Data Items, and Choice of Sample Data

The organization of the training experience involves the choice of the data that will be available to the software. The choice of which data items to include in the training experience is perhaps the most important choice of all. The data items that are chosen as predictors in the training experience can be called the independent variables; the predicted classification can be called the dependent variable. In our examples, **Investigate?** will be the dependent variable. We will call **Investigate? = Yes** a positive result and **Investigate? = No** a negative result.

The term "predict" may seem odd in this context. It is generally used in discussing the application of machine learning techniques to time-series data. The idea is that we are trying to predict a future outcome, such as, whether a patient will later recover from pneumonia, or, in our example, whether a transaction would later be classified as **Investigate? = Yes** by an expert.

If the independent data items we choose for analysis are not sufficiently related to the dependent classification variable, the software has no chance of being able to learn to successfully classify any cases beyond those presented in the training experience. Taking an extremely obvious example, if the only data items we included as independent variables in the training data were name, address, date of deposit, and account type, the program would be unable to generalize beyond the training experience. Some of the data items that might be used as indpendent variables are shown in the shaded box below.

One strength of machine learning is that if you

Selection of Data Items to Use as Indpendent Variables is Critical

A few of the independent variables related to the account would be:

- Account Type (personal vs. corporate, savings vs. checking, type of checking or savings);
- Account Holder Entity Type (individual personal account vs. joint personal account vs. business, and type of business);
- Account relationships with other accounts;
- Stated expectations of account usage;
- Account location, with risk rating of geographical region of account holder and related accounts including habitual counter-parties;
- Periodic and current account balances.

A few of the independent variables related to individual transactions would be:

- Transaction Date, Time, and Amount;
- Transaction Type (deposit vs. withdrawal, monetary vs. non-monetary, teller vs. non-teller, purchase or deposit of negotiable instrument with instrument type, wire transfer with to and from locations);
- Comparison with frequency and volume of transaction types over specified periods (days, months, years);
- Calculated comparisons to normal transactions;
- Calculated comparison to normal and current account balances;
- Words used in comments or description fields, sometimes given a risk rating.

In addition to using these and other independent variables relative to accounts and transactions at the individual account level, most of the systems would also compare transactions made by peer groups of similar accounts.

include a large number of independent variables some of which may have no relationship to the dependent variable and others of which have complex relationships to the dependent variable, machine learning approaches can learn to predict the correct classification based on complex relationships that might not be obvious to people.

The selection of data items to include is one of the areas in which domain expertise comes into play. Domain experts can examine the data items selected to at least make sure that nothing they think might be relevant is left out. In the real world, data for money laundering software comes from multiple sources. The issues surrounding what data items are selected, and the formats and timeframes in which they are made available to the software are critical both in the system's knowledge discovery capacity and its ongoing operational detection of money laundering transactions in production. The issues involved in making the data formats consistent and providing things like universal account and transaction numbers that will be unique across all the systems are the same issues that would be involved in a data warehouse project.

The sample data items that we'll use in our examples are in the shaded box below.

It's important to avoid choosing independent variables that have a direct but non-predictive relationship to the dependent variable. Taking an extreme example, if we made universal, unique transaction identification number an independent variable, the computer could easily develop an algorithm to correctly predict all the outcomes for the cases in the training data set, but it would have no ability to predict any outcomes for cases not in the training data set. Much less obvious examples of this occur frequently when choosing variables from production data, since the meaning of a data item is not always clear. One symptom of this kind of problem is results that are too good on the training data set but poor on any other set of data.

We've chosen these four calculated variables so that our examples can have some meaning with just a few variables. In a real-life situation it is highly advisable to insure access to data at the most granular level possible (as in any data warehouse project). Data can always be summarized; summarized data generally can't be decomposed back into its more granular form, and that granular form might be desirable at some point, either on its own or to perform a different summary.

In addition to the choice of data items, the design of the training data set also involves the choice of sample cases. We are all familiar with the notion of random sampling. In this case, random sampling is generally not appropriate. A random sample would

Independent Variables for Use in Examples

We'll use a sample training data set with just 4 independent variables for purposes of illustration in this article. Variable names will appear in the font shown below. The variables we chose are:

- **Geo Risk**, which will be a Risk Rating of the Country of the Account Holder;
- **Trans Amt/Bal**, will be based on the Transaction Amount relative to Average Individual Account Balance over the last six months;
- **Relative Peer Trans Amt/Bal**, which will be based on the (Transaction Amount) relative to (Average Individual Account Balance over the last six months) compared to an analogous Peer Group measure of (Average Transaction Amount for this month for the Peer Group) relative to (Average Account Balance over the last six months for the Peer Group); and
- **Relative Peer Trans Freq**, which will be based on the (Transaction Frequency for the past 30 days of the Account that made the Transaction) relative to the (Transaction Frequency for the past 30 days of the Accounts in the Peer group).

show an overwhelming majority of negatives (**Investigate? = No)** and very few positives. In a data set where **Investigate = No** was the outcome 97% of the time, any methodology would achieve 97% accuracy merely by predicting that **Investigate?** is always **No**. In order to learn to identify the positives, the training data set will very likely have to contain a higher proportion of positives than we would find in a random sample. (Even if the software was taking the opposite approach of identifying the normal transactions and reporting on exceptions, the sample would have to be skewed towards exceptions in order to learn to distinguish normal from abnormal, unless it purely took the approach of reporting all departures from normality.)

Search Space and Valid Hypothesis Space

In computer science, developing an algorithm for predicting a result is often viewed as a search through a set of hypotheses to find the hypothesis that most accurately predicts the result. We will adopt the following restricted definition of **Hypothesis**:

Hypothesis: A statement or set of statements that predicts one or more dependent variables based on the values of a set of independent variables.

We will enclose hypohtheses in curly braces to set them off, for example: *{If* **Geo Risk = High**, **Investigate = Yes***}*.

We will define a **Case** as follows:

Case: Given a set of independent variables, a case is a particular set of values for each of the independent variables. For example, taking **Geo Risk**, **Trans Amt/Bal**, **Relative Peer Trans Amt/Bal**, and **Relative Peer Trans Freq** as the set of independent variables, (**Geo Risk=High**, **Trans Amt/Bal=Normal**, **Relative Peer Trans Amt/Bal=Low**, and **Relative Peer Trans Freq=Normal**) would be a case. Case is synonymous with the mathematical term, vector.

The total set of hypotheses is called the **Search Space**, which we will define as follows:

Search Space: The complete set of possible hypotheses that can be constructed to predict a dependent variable for the set of all possible cases, given a set of independent variables.

Within the total set of hypotheses, and also within the set of hypotheses for a particular search algorithm, there will generally be more than one hypothesis that predicts the outcomes in the data with equal degrees of success. For the purposes of this article, we'll define this set of hypotheses as the ***Valid Hypothesis Space***:

Valid Hypothesis Space: The set of hypotheses in the search space in which all the hypotheses that are members of the set are consistent with a given set of cases to a degree greater than or equal to a specified level of accuracy.

In the decision tree section of this article, the four independent variables described above will be coded as having 2 values for **Geo Risk** and 3 each for **Trans Amt/Bal**, **Relative Peer Trans Amt/Bal**, and **Relative Peer** Frequency. Table 1 shows an example training data set consisting of only four cases.

Based on this very small set of cases, the Valid Hypothesis Space would be very large. One valid hypothesis would be: *{If any value is* **High**,

	Geo Risk	Trans Amt/Bal	Relative Peer Trans Amt/Bal	Relative Peer Trans Freq	Investigate?
1	High	High	Low	Normal	Yes
2	Normal	Normal	Normal	Normal	No
3	High	Normal	Normal	Normal	Yes
4	Normal	Low	High	High	Yes

Table 1: Very small example training data set

Investigate = Yes}. Another would be: *{If* any two values are **High**, **Investigate = Yes**}. This illustrates that there will frequently be a more specific and a more general version of a valid hypothesis. Another would be: *{If* **Geo Risk** is **High**, **Investigate = Yes**}. Another would be: *{If* **Geo Risk** is **High** AND NOT **Trans Amt/Bal = High**, **Investigate = Yes**}. Another hypothesis that would be consistent with the data presented is: *{If at least one of the values is* **Low**, **Investigate = Yes**}. As humans, we can instantly see that this last hypothesis, while it is consistent with the data, would have very poor predictive value for cases outside of the training data set. Our example seems trivial because there are so few cases, and, in general, the more cases in the training data set, the more likely that the set of consistent hypotheses will be predictive for a larger universe of cases. But there will almost always be a number of hypotheses consistent with the set of cases (and therefore in the valid hypothesis space), even when the training data set is very large.

Limiting the search space is important because, as shown above, the ways of combining the cases for even a simple set variables is extremely large. With the small set of variables and values specified above, the set of possible cases is only 2 * 3 * 3 * 3, or 18. However, the number of all the ways to combine the hypotheses that cover these cases is much larger. Mathematically, the number of ways of combining all the hypotheses that use propositional Boolean logic (as decision trees implicitly do) would be called the set of all the subsets of the cases, also known as the power set. The number of members in the power set of a set of cases can be calculated as $2^{(\text{number of cases})}$. In this example, that is 2^{18} which is 262,144. And, if you expand the ways of constructing hypotheses to include assigning real number weightings to the independent variables, as neural networks do, and expand the set of independent variables to a more realistic number, the set of possible combinations becomes impossibly large to consider and still maintain computability, even with the speed of today's computers.

The set of hypotheses within the search space that the computer can consider is limited by the approach that the computer uses to solve the problem. The constraints that decision trees, neural networks, and statistical profiling provide the limits required to keep the search space manageable; the particular ways in which they limit the search space is one of the things that differentiates them.

Overfitting the Training Data Set

A hypothesis (A) **overfits** the data when there is a hypothesis (B) in the valid hypothesis space that performs better on a larger set of data, even though hypothesis (B) does not fit the training data as well hypothesis (A). Hypothesis (A) is learning the training data too well, mistaking the training data set for the real data. This can happen for a number of reasons. One reason is that there are coincidental relationships in the training data that do not hold up in the larger data set. For example, if we had made currency an independent variable, and there was one transaction in Greek Drachmas in the training data set that was pre-identified as suspicious, the hypothesis that all Drachma transactions should be tagged as suspicious would be in the valid hypothesis space for the training data set. Without that hypothesis, it is possible that no other hypothesis in the valid hypothesis space would predict that the Drachma transaction would be suspicious (assuming the specified level of accuracy for the valid hypothesis space is less that 100%, as it might have to be if the factors that made the Drachma transaction suspicious were not included in the training data set as independent variables).

In the training data set above, it was obviously a coincidence that both of the cases that had values of **Low** and **Normal** for at least one of the variables were positive. Eliminating the hypothesis that **Investigate = Yes** for any transaction if **Low** and **Normal** are included as two of the values of independent variables would not reduce the predictive accuracy of the valid hypothesis set for the training data set. But it would almost certainly increase the accuracy of the valid hypothesis set on cases outside of the training data set.

In the larger data set we are imagining as part of this example, we will now postulate that it is also a coincidence that any case that has a value of **High** for **Geo Risk** is positive. If we looked at more cases in which **Geo Risk** is **High**, without any other factor leading to **Investigate = Yes**, **Investigate** would be **No**. How could that be true? One way it could be true would be if there were factors that were not included in the analysis as independent variables, as with the Drachma transaction above. Without some of those factors indicating a suspicious transaction, **Geo Risk** alone would not lead to a positive outcome. It happens that in the case in the training data set, these unseen factors were positive so the case was correctly classified as positive, but in other cases, those factors are not always positive. If that were true, the hypothesis: *{If* **Geo Risk** is **High**, **Investigate? = Yes***}* would increase the rate of successful predictions for the training data set, but decrease it for real data.

The graph in Figure 1 illustrates the effect of overfitting. The accuracy of the model generated during the first part of the training rises rapidly, almost as rapidly on the test data as on the training data. At some point, as the model is trained to fit more and more accurately to the training data, its accuracy when applied to the test data starts to decrease. The optimal model is the one that reaches maximum accuracy on the test data.

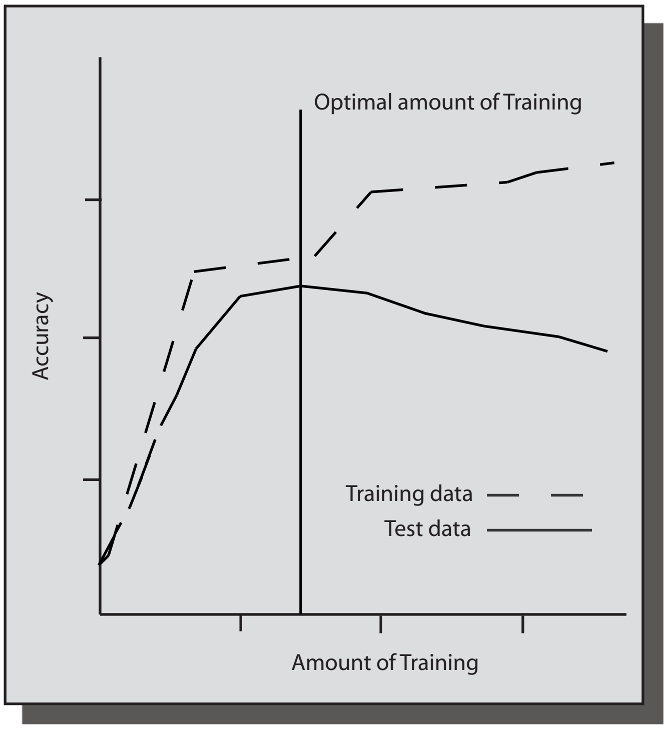


Figure 1: Effect of Overfitting

Notice the use of the words "test data" rather than "real data". Almost all the approaches to the problem of overfitting involve developing a model on a training data set, and then testing that model on other sets of data to determine the point at which overfitting becomes a problem. A very frequent approach is to divide the initial sample into three independent data sets: a training data set, a test data set, and an evaluation data set. The evaluation data set is used to once again test the accuracy of the model after the effects of overfitting have been identified and reduced using the test data set.

Overfitting is related to another problem. Each of these approaches may find a locally optimal solution that is not globally optimal, that is, the first solution they find as a result of where they start their evaluation and their underlying assumptions may not be the best solution they could achieve. Even within an approach (decision tree, neural network, or statistical profiling), the model that the computer arrives at by traversing the hypothesis space in a particular way might be better than the other ones that the computer evaluated, but might not be better than some others that the computer has not evaluated. This is discussed for each approach in the section that covers that approach.

Noisy Data and Missing Values

Another source of inaccuracy in the model generated on the training data set is noisy data. In the real world, data contains errors and inconsistencies. If there are errors and inconsistencies in the training data set, the model learns from the disinformation they provide. Generally, the same techniques used to combat overfitting also work to reduce the impact of noisy data. Some approaches are very vulnerable to the effects of noisy data. However, for decision trees, neural networks, and statistical profiling, there are effective methods of dealing with noisy data in the training data set.

Sometimes values are missing in some cases either in the training data set, test data set, evaluation data set, or during actual system operation. Common methods for dealing with this situation are to ignore the case, to assign a default value of the independent variable, or to assign a probability to each of the possible values rather than simply assigning a default. When a system assigns a default it frequently uses some criteria of similarity to other cases to decide what default to assign. While ignoring a case may work during the learning process, it is not a good alternative during system operation, since it would risk skipping over a potential money laundering transaction, and in some cases, such as a case in which a work telephone number is missing, the very fact that the data is missing could contribute to flagging a transaction as suspicious.

Value Weighting Outcomes

There are frequently situations in which it is more important to accurately predict one outcome or classification than another. In the detection of money laundering, as in cancer screening, a false negative is a bigger problem than a false positive. Further investigation exposes a false positive. A false negative is not investigated further, and the costs can be very high. Of course, there is a balance. If the level of false positives gets too highg, it creates an insupportable burden on the investigators. Still, it is desirable in money laundering detection to err on the side of caution.

One obvious technique for favoring false positives over false negatives is simply to lower the threshold for reporting of positives. However, more sophisticated techniques can actively incorporate value weighting outcomes at each step in the process, which may result not just in a lower threshold, but actually in a different model.

Decision Trees

Decision trees are one popular, effective approach to classification and prediction problems. Decision trees have been used successfully to detect fraud, to identify borrowers who will declare bankruptcy in the next 12 months, to predict which molecular structures will be effective against the HIV virus, and to predict the crashworthiness of potential new automobile designs. A decision tree works like a game of Twenty Questions. In Twenty Questions, one person thinks of something and the other players are allowed to ask a series of questions to guess what the person is thinking of. The goal of the game is to be able to guess what the person is thinking of with the minimum number of questions.

For the decision tree, the allowable set of questions is limited to the independent variables. What the algorithm is looking for is the ideal order in which to test the variables.

To illustrate the decision tree approach, we'll code the values of the independent variables specified in the General Framework section above as follows:

- **Geo Risk** will have 2 possible values: **High** or **Low**;
- **Trans Amt/Bal** which will have 3 possible values: **High**, **Normal**, and **Low;**
- **Relative Peer Trans Amt/Bal** will have 3 possible values: **High**, **Normal**, and **Low;** and
- **Relative Peer Trans Freq** have 3 possible values: **High**, **Normal**, and **Low.**

Figure 2, on the next page, represents a decision tree based on these independent variables.

Decision trees start at a root node, usually drawn at the top. In Figure 2, the root node for classifying a transaction is **Relative Peer Trans Amt/Bal**. The transaction is then followed down the tree through the child nodes corresponding to the values of its attributes. Just as in Twenty Questions, the independent variable to examine at each child node depends on the variable that comes before it. A case is classified by moving down a tree branch until you reach the end of the branch (called a leaf node). In Figure 2, the left-

Association of Certified Anti-Money Laundering Specialists
ACAMS

☞ (continued on page 10)

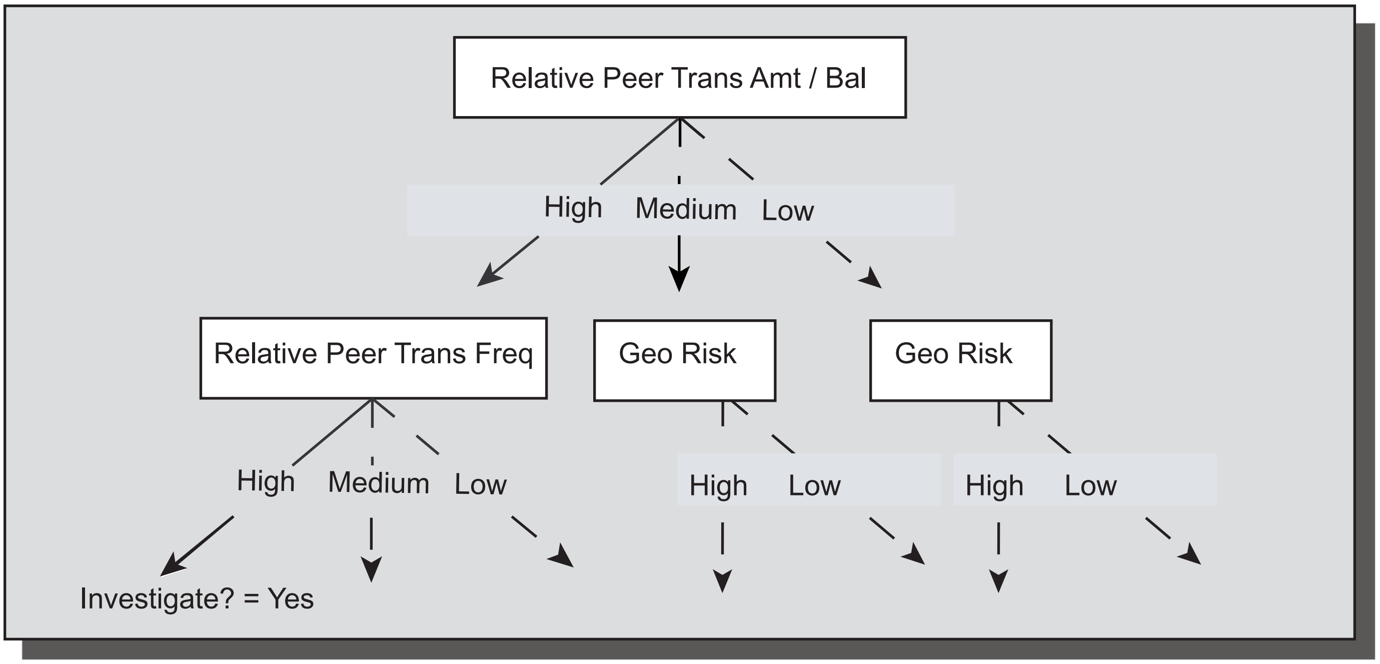


Figure 2: Top levels of a decision tree for determining the value of **Investigate?**

most branch has already reached a leaf node. If a case has a high **Relative Peer Trans Amt/Bal**, the next independent variable to test is **Relative Peer Trans Freq**. If that is also high, this tree will classify the case based on just those two independent variables as **Investigate? = Yes**. In other words, no matter what the values of the other two independent variables are, the transaction will be investigated.

If a case has a low **Relative Peer Trans Amt/Bal**, the next independent variable to test is **Geo Risk**. If **Geo Risk** is high, the tree will proceed to the next independent variable, since it has not yet determined the classification based on those two answers by themselves. It will continue to follow down the branch until it either reaches a leaf node with an **Investigate?** value or until it exhausts the independent variables without having reached a leaf node with a determinate value. A leaf node with an indeterminate value indicates that the training data set does not contain enough information to classify the combination of values of the independent variables represented in the branch definitively as positive or negative.

Notice that the variables have been coded as discrete, non-continuous values. This is a requirement of the decision tree approach, because the decision at each node must be a decision between one mutually exclusive classification and another. We've taken the standard approach for implementing decision trees based on values that are continuous, namely, dividing the continuous values into ranges, such as **High**, **Normal**, and **Low**. There will inevitably be a loss of information, which may impact the efficacy of the model. The values can be divided into narrower categories, but the finer the division, the more complex will be the resulting decision tree, which defeats the goal of making the decision tree simple. This is one of the trade-offs built into the use of decision trees, although some advanced implementations have sophisticated methods for handling continuous values that minimize the impact of this problem.

Splitting and Independent Variable Test Order Criteria

One of the most important differentiations among alternative decision tree algorithms is the function they use to select the order in which to place each of the independent variables in each of the tree branches. Many different decision trees are members of the valid hypothesis space. The principle that the basic decision tree approach generally uses to choose between these trees is the same principle that determines who wins in Twenty Questions; the tree with the fewest nodes (that is, the simplest tree, corresponding to the fewest questions in Twenty Questions) is judged by the decision tree generation algorithm to be the best predictor of outcomes. The choice of the ordering and splitting principle is generally geared to produce the simplest possible tree. (This method of searching the hypothesis space is known as a greedy search, as opposed other types of searches such as breadth-first or depth-first, because it tries to get the most right answers as quickly as it can.)

The choice of the simplest tree as the best is based on an assumption associated with the philosophical principle of Occam's razor, much used in philosophy and science since 1320, when William of Occam formulated it. Occam's razor postulates that the simplest hypothesis that fits a set of data is the best. In practice, Occam's razor is often, but not always, correct. When it is incorrect, the decision tree approach will not choose the optimal decision tree from the set of possible decision trees. There are a number of remedies for this, including generating trees with different splitting functions and testing the resulting trees on the test data to see which performs the best.

The aim of the splitting function is to select the independent variable that is most useful for splitting the cases at each node. The root node should test the independent variable that most effectively splits the outcomes. Each child of the root should test the independent variable that most effectively splits the remaining cases, and so on. One function sometimes used to make this determination is based on the information theory measure called entropy, which is the result of a mathematical function to measure the extent to which the groups are homogeneous after a split. The remaining independent variables are tested at each node and the one that results in the most homogeneous groups is selected until the group has a completely homogeneous value of **Investigate?= Yes** or **Investigate? = No**, at which point it becomes a leaf node.

One of the most widely employed decision tree algorithms is called CART (Classification And Regression Trees). CART produces binary trees (that is, trees that only allow two classifications at each split, like the version of Twenty Questions in which only yes or no questions are allowed). CART originally used a splitting function quite similar to entropy called Gini, named after the Italian economist who invented it. A different algorithm that can also be used with CART is called "Twoing" which produces trees that may be more balanced than trees produced using

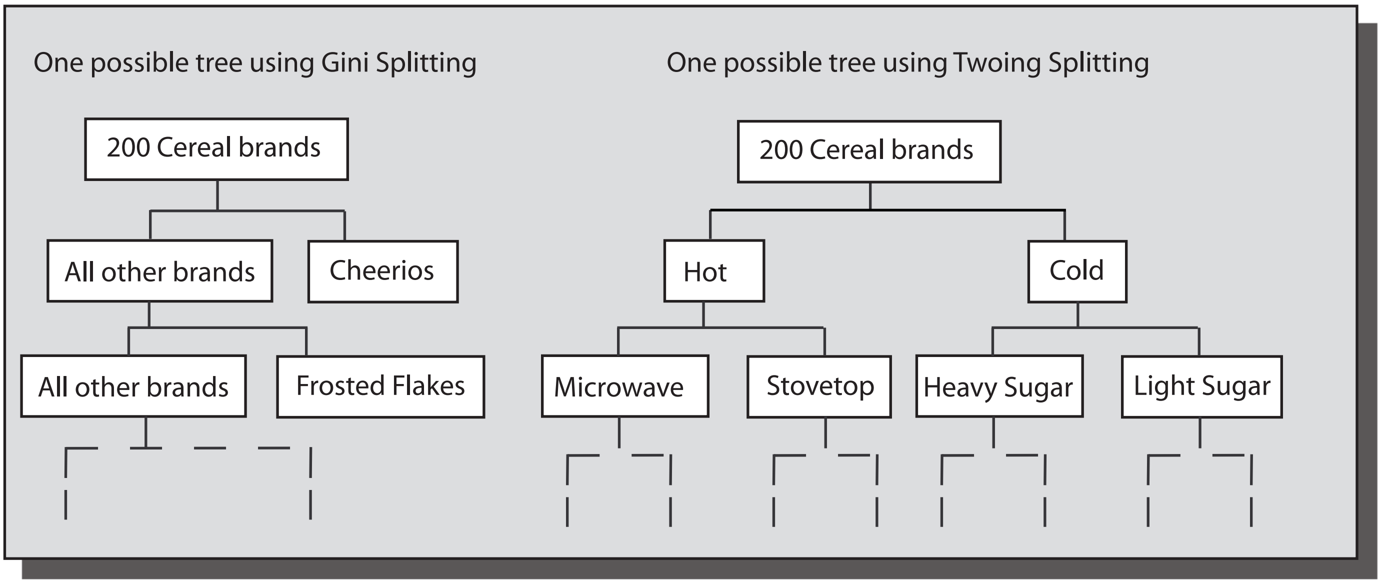


Figure 3: Decision Trees based on different splitting algorithms

Gini or entropy as the split ordering function.

For example, say we're trying to predict which of 200 brands of cereal that a consumer is most likely to buy. On the left in Figure 3, on the previous page, is a decision tree based on a Gini algorithm, on the right a decision tree based on a Twoing algorithm.

Figure 3 also illustrates a possible alternative principle for preferring one decision tree over another. Assuming that the two trees in Figure 3 are both members of the valid hypothesis space with similar degrees of accuracy, the tree based on twoing might be preferred because its discriminations are much easier for people to understand. The more powerful decision tree packages provide a choice of splitting function. You can experiment with different splitting functions and decide which produces the result that is most useful for your purpose.

Generating Rules from Trees

One major appeal of the decision tree approach is that there is a virtually direct translation from the decision tree to rules that can be understood, evaluated, and applied by humans. A rule produced by the tree shown in Figure 2 would read:

{*If* **Relative Peer Trans Amt/Bal** = **High** AND **Relative Peer Trans Freq** = **High** THEN **Investigate? = Yes**.}

Dealing with Overfitting

There are two approaches to deal with overfitting of decision trees:

- stop growing the tree at some point; and
- post-pruning the tree.

Post-pruning the tree is more effective in practice because it's too hard to decide when to stop growing the tree without building the entire tree. The decision tree generated while training on the training data set will normally be post-pruned after it is applied to the test data. An error-reduction formula can be applied at each node to determine if that node improves the accuracy of the tree on the test set. Each branch of the tree will be pruned back from leaf nodes to the point at which the error rate for the test data starts to rise. This will prune the tree back to a point that approximates the optimal amount of training, illustrated in Figure 1 in the General Framework section, above.

As the tree is pruned back from the leaves, the purity of the outcomes will be reduced. The leaves were reached through a series of nodes that progressively worked their way down to the most homogeneous set of outcomes (either all positive or all negative). When the tree is pruned, the new leaves will have a mixture of positive and negative outcomes. At that point, the probability of a positive or negative outcome at that leaf node can be calculated based on the proportion of positive and negative outcomes. As is always the case with the outcomes of neural networks and statistical profiling, a threshold can then be set. Above a certain level of probability of a positive outcome, the outcome will be given a value of **Investigate = Yes**, possibly with an indication of the degree of certainty of the classification.

Another technique involves translating the entire tree to a set of rules and then pruning the rules rather than the tree. One advantage of this technique is that it allows pruning nodes in the middle of a branch, removing the limitation that a branch can only be trimmed back from the leaf node. Another is that improves the readability of the rules that can be derived from the tree. Once the rules have been derived, they can be tested individually and the order in which the rules are tested in the operational system can be tweaked to improve efficiency.

Summary of Strengths and Weaknesses of Decision Trees

The natural strengths of decision trees include:

- direct translation of models to understandable rules;
- good performance when the problem domain lends itself to rules based on the data items after they have been classified

into mutually exclusive values;

- easy handling of categorical variables with more than a single category, which pose problems for neural networks and statistical techniques; and
- good performance.

The natural weaknesses of decision trees include:

- difficulty of classifying continuous variables into discrete, mutually exclusive categories without information loss;
- poor performance when the problem domain does not lend itself to rules based on the data items after they have been classified into mutually exclusive values; and
- difficulty determining the best tree to use when that tree is not the smallest tree.

As with each of these approaches, the effectiveness of decision trees is heavily dependent on how you implement them, on what data you use and how you formulate that data for the decision tree's use, and on which of the variations of the approach you choose. The only definitive measure of effectiveness is testing on real data, and even then, the effectiveness will vary depending on the data you use as your test sample.

Neural Networks

A psychologist and a logician came up with the original artificial neural networks in the 1940s as part of an attempt to understand how the brain works. They observed that the brain is comprised of a very complicated interconnected network of brain cells. Emulating this structure, they designed a computer model consisting of a complex, interconnected network of independent computing units, each with a similar construction. Figure 4 is a graphical representation of a hypothetical neural network for predicting the probability of **Investigate?=Yes** from our four independent variables.

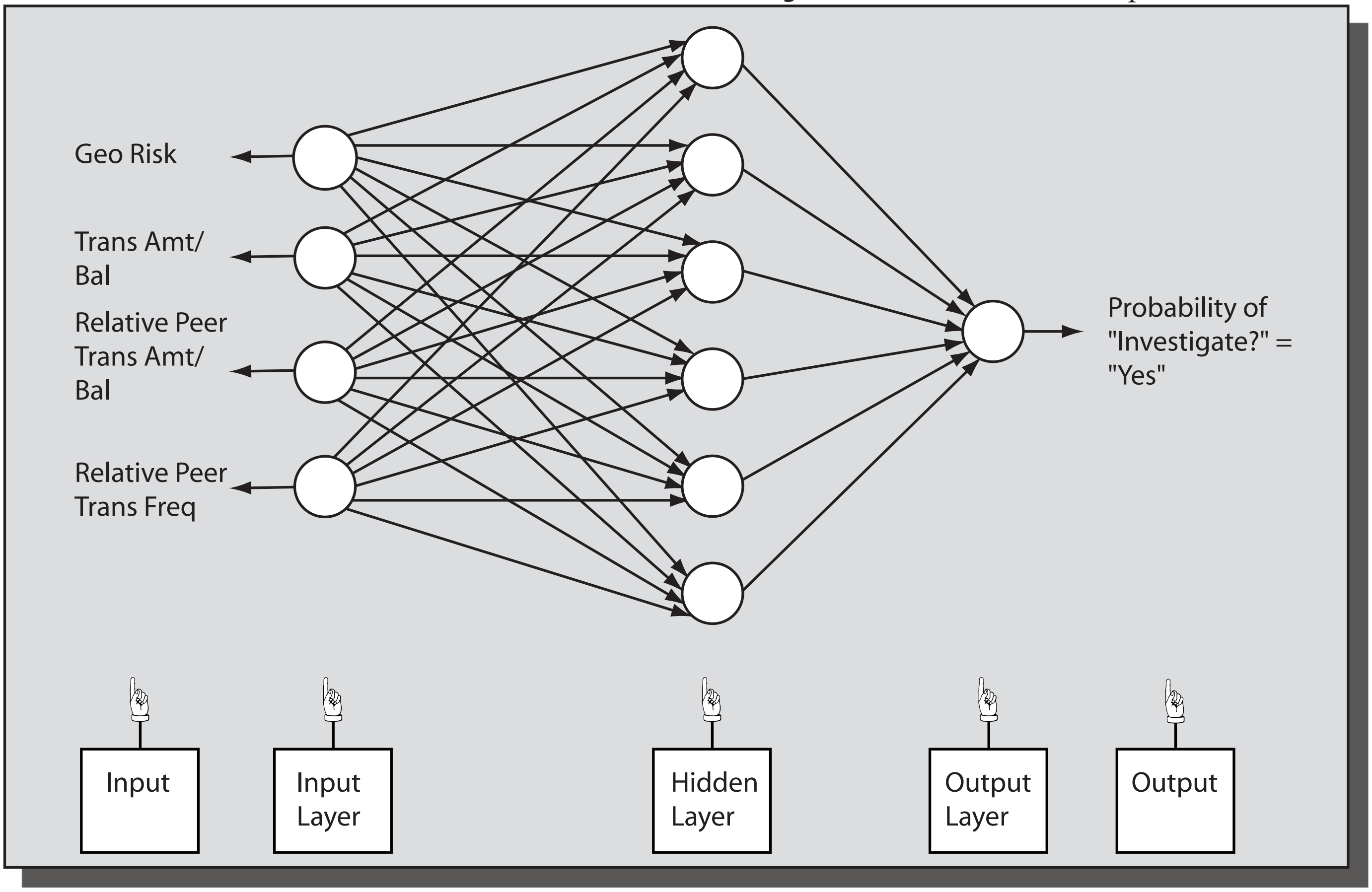


Figure 4: A neural network for predicting the probability of **Investigate? = Yes**

Each of the input variables is fed into one of the units in the input layer. The input layer unit processes the input and produces a single output. That output is passed on to each of the units in a second layer of units, called the hidden layer. The connection of the input layer units to each of the units in the hidden layer gives the network the power and flexibility to represent very complex relationships among the input variables. The units in the hidden layer are each connected to an output unit, which combines the values they produce into a single value, in our example, the probability that **Investigate? = Yes**.

The independent units in the classical form of neural network are called **Perceptrons**. The structure of each of the perceptrons is similar. In the original model, each perceptron would output a 0 or a 1. Perceptrons would determine whether to output a 0 or a 1 based on comparing the weighted sum of its input values to a threshold level. The biological analogy is that a neuron in the brain fires, if, and only if, it is sufficiently stimulated. When it fires, it sends an impulse through its dendrites and synapses to other neurons. Perceptrons split their computation into two components, a combination function and a transformation function. The combination function is generally some form of weighted sum of the inputs, with a separate weight assigned to each input. For our example, this would be expressed as follows:

$$\chi = \omega_1(\textbf{Geo Risk}) + \omega_2(\textbf{Trans Amt/Bal}) + \omega_3(\textbf{Relative Peer}\ \text{Freq}) + \omega_4(\textbf{Relative Peer Trans Amt/Bal})$$

where χ is the result of the combination function and ω_1 - ω_4 are the weights that the perceptron assigns to each of the inputs.

In the original model, this result would be compared to a threshold value to produce a 0 or a 1. In a neural network with one hidden layer, each perceptron in the hidden layer would output a 0 or a 1 to the output perceptron. The weights assigned to the inputs of each of the perceptrons in the hidden layer are independent of each other. The output perceptron would, in turn, apply its combination

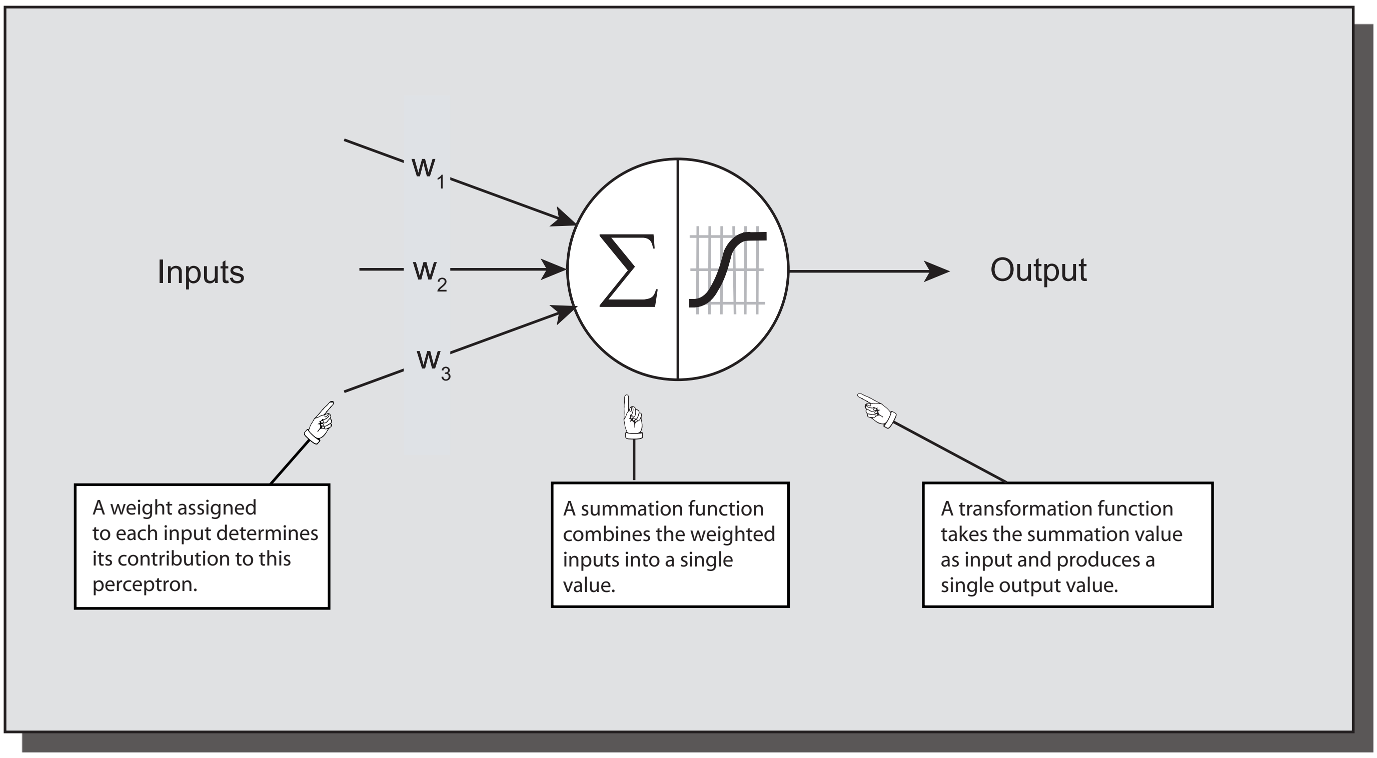

Inputs
w1
w2
w3
Σ
Output
A weight assigned to each input determines its contribution to this perceptron.
A summation function combines the weighted inputs into a single value.
A transformation function takes the summation value as input and produces a single output value.

Figure 5: A **perceptron**, *the independent unit in a neural network*

function to the inputs from each of the perceptrons in the hidden layer and then apply its transformation function to determine whether to output a 0 or a 1, which, in our example, would correspond to **Investigate? = No** or **Investigate? = Yes**. The structure of a perceptron is illustrated in Figure 5.

During the training process, the neural network modifies the weights assigned to each of the inputs of each of the perceptrons until the network can successfully predict the outcomes of the cases in the training data set. Looking at the model of the neural network shown in Figure 4, you can see that being able to adjust the weight of each input to each perceptron allows neural networks to model very complex relationships among the input variables.

Backpropagation

Work on neural networks stalled in the 1970s because the most powerful computers generally available at that time were inadequate for supporting neural network analysis of real world problems and because two MIT professors published a paper showing that the original neural network model had theoretical problems. The development of neural networks picked up momentum again in the 1980s as a result of two developments: John Hopfield invented backpropagation, which addressed the theoretical problems, and computers became much more powerful. Since then, neural networks have been widely employed in the commercial world.

The basic steps in backpropagation are:

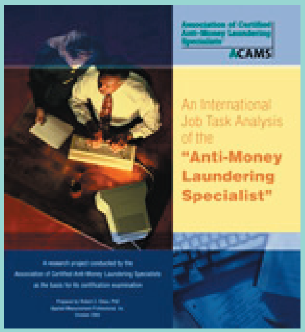
- the output for a case is calculated based on the existing weights in the system;
- the discrepancy between the output and the expected result is calculated;
- each unit is assigned a share of responsibility for the error, starting by assigning the entire error to the output perceptron, which then assigns a share of responsibility to each of the perceptrons in the hidden layer using mathematical procedures such as taking partial derivatives of the transformation functions; and
- the weights are nudged in a direction that would lessen the discrepancy in proportion to their share of responsibility for the error.

A factor called the learning rate controls how quickly the weights are changed. This usually starts out large and decreases as accuracy improves.

The process of adjusting the weights involves measuring how sensitive the output of a perceptron is to changing the weight on each input, and whether changing the input would increase or decrease the discrepancy. When the backpropagation algorithm changes the weights in a direction that will correct the discrepancy, it doesn't adjust them to exactly predict the outcome, because the final set of weights needs to work for all of the cases in the training data set, and also because exactly fitting all the cases in the training data set increases the likelihood of overfitting.

Variations on Basic Neural Networks

There are a number of variations on the basic neural network approach. Some neural networks are feed forward networks; others are recursive. In a feed forward network the links between the perceptrons are unidirectional. Perceptrons in one layer link to the next layer; there are no links from a perceptron to another perceptron in its own layer or to a perceptron in a previous layer. A recursive layer allows such links. The brain is a recursive network. The neural networks in commercial use are mostly the simpler feed forward networks.


Association of Certified Anti-Money Laundering Specialists
ACAMS
An International Job Task Analysis of the "Anti-Money Laundering Specialist"

First Ever CAMS Certification Exam!
***Deadline to Apply is February 12th**
Miami Beach, Florida
March 26, 2003

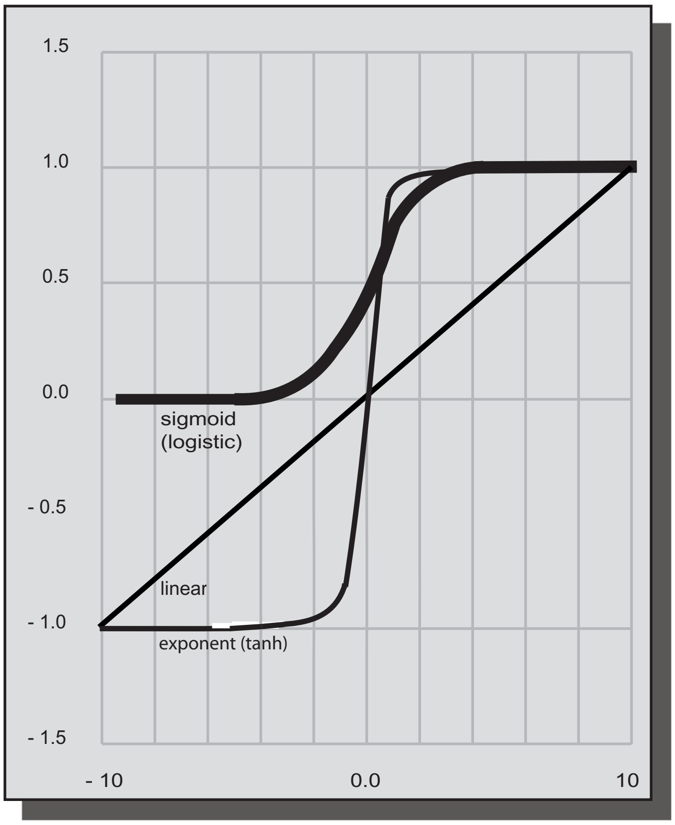


Figure 6: Three Transformation Functions

There are also different choices for the transformation function that converts the output of the combination function in a perceptron into the perceptron's output. There are three common choices for the transformation function: linear, hyperbolic tangent, and sigmoid. Figure 6 is a graphical representation of the difference between these functions.

The linear function is not generally used because it can only represent linear relationships. Both the sigmoid function and the exponent (tanh) function can represent non-linear relationships. The main difference between them is that the exponent (tanh) function produces outcomes between –1 and 1, while the sigmoid function produces values between 0 and 1. The sigmoid function is by far the most commonly used transformation function. Usually the same transformation function is used in all of the perceptrons in a network.

Up to now we have talked about output values being 0 or 1. With the sigmoid function, the values are continuous between 0 and 1. In order to produce a result of **Investigate? = Yes** or **Investigate? = No**, we need a threshold value. This threshold value can be adjusted to produce greater sensitivity to **Investigate? = Yes** so that we don't miss any positives. (In addition, there are ways to incorporate value weighting of outcomes into the entire weight adjustment process.) Having values between 0 and 1 is advantageous because it allows us to see where the cases are on a spectrum of very normal to very suspicious. We would want to put a priority on investigating those transactions that are most suspicious.

There is also the option of having multiple hidden layers. In practice, a single hidden layer is usually more than sufficient, and more hidden layers increase the danger of overfitting. But for extremely complex domains more hidden layers may be useful. One of the largest neural networks ever deployed, AT&T's network for reading numbers on checks, had seven layers with a combined total of hundreds of thousands of individual units.

Backpropagation is the learning algorithm that got neural networks out of the theoretical slump of the 1970s, but, since the discovery of backpropagation, a number of other learning algorithms have been explored. One, PRCE (Probabilistic Restricted Coulomb Energy), which has been used in both credit card fraud detection and money laundering detection, transforms the independent variables into features and builds a multi-dimensional feature matrix. Features of two different classifications (such as **Investigate? = Yes** and **Investigate? = No**) can sometimes have an area of overlap. For example, as **Geo Risk** goes from **High** to **Low** in a continuous spectrum of values, a greater proportion of **Investigate? = Yes** transactions have **Geo Risk** on the high end, although there are still many **Investigate = No** that have a **Geo Risk** on the low end. During training, regions in the multi-dimensional feature matrix are identified as being more or less likely to belong to one classification or another. A probability density function is used to measure the probability that a transaction with a set of features that put it in a particular place in the matrix should be classified as **Investigate = Yes**.

Genetic algorithms have also received

attention as a technique for neural network learning. Genetic machine learning algorithms deploy competing neural networks and evolve them to see which survives as the fittest.

Data Preparation: 0 to 1

One characteristic of neural networks is that they have an affinity for input data that has been massaged so that all the values of all the input variables are in the range of 0 to 1. One reason for this is that, since all of the perceptrons in a network have the same structure, and the outputs of some of the perceptrons are the input to others, it is desirable to have all of the values in the same form, which is 0 to 1, if you are using the most commonly used transformation function - the sigmoid function. This is the neural network counterpart to the need to divide values into mutually exclusive categories for use with decision trees.

Massaging all the inputs to be in the range of 0 to 1 for neural networks is as hard as dividing them into mutually exclusive categories for decision trees. The most common technique for variables that have continuous values such as dollar amounts, averages, and ratios, is to predefine a range bounded by a minimum and a maximum value and define a value somewhat lower than the minimum as 0 and a value somewhat higher than the maximum as 1. Of course, the process of defining minimums and maximums is notoriously error-prone. There are several techniques for handling variables that represent mutually exclusive categories (ordered mutually exclusive values such as numbers of children or age need to be handled differently than unordered values such as gender and status codes). Dates present another issue. Of course, there are ways of handling all of this, but they are not without pain, and they introduce a potential source of error.

Overfitting and Premature Convergence on a Solution

One of the algorithms used to avoid overfitting in neural networks is a technique called weight decay, which consists of decreasing the weights by a small amount in each iteration. The effect of this is to keep the final weight values small, which militates against the selection of a complex, highly differentiated hypothesis, resulting in a neural network that is less likely to have been overfit to every nook and cranny of the training data set.

The best technique to avoid overfitting for neural networks, as for all of the approaches, is the use of additional data sets after training on the training data set. Use of a test data set and an evaluation data set, as described in earlier sections, is highly recommended. Additionally, there are various methods of cross-validation using the same data partitioned in multiple ways.

One problem faced by neural networks that is related to overfitting is that they sometimes converge prematurely on a less than optimal solution. This is the problem of the local versus global optimal solution referred to in the General Framework section. Unlike decision trees, neural networks generally start their search in a relatively arbitrary point in their overall hypothesis space. Their usual search algorithm is called a gradient descent search, as opposed to the greedy search algorithm used for most decision trees. One way to visualize a gradient descent search is to think of a topographical model with hills, mountains, valleys, and crevices. The lowest point in the topography represents the globally optimal solution. The gradient descent search keeps trying to move in a downward direction to find the lowest point. Once it can go no lower, it stops. The problem is that it may be in a valley on one side of a mountain that has a much deeper canyon on the other side. It won't automatically proceed up the mountain to find the canyon. A variant of the gradient descent search called a stochastic gradient descent search is designed to decrease this danger, but the best cure is, once again, to test against more data. A solution that is locally but not globally optimal will not perform well on data outside the training data set.

Some vendors offer a facility to re-train the system that can be executed by the user with little vendor assistance. This is very valuable because if new patterns emerge in the data, re-training can make the system significantly more accurate. Of course, as is the case with any system, the system is only as good as its input variables. If the change in patterns of activity involves data items that have not been selected for inclusion in the model, the vendor will need to be involved in adding these new independent variables. The re-training only readjusts the weights of the independent variables that are already defined to the system.

The Black Box Bogey

The criticism of neural networks most frequently heard from vendors who don't use them is that they are a black box; you don't know why a transaction has been reported as suspicious. It is true that you wouldn't be able to understand the weights that have been assigned in each unit of the neural network even if you used the advanced technologies that are now available to know what the weights were. The power of the network lies in the complexity of the interactions between the units and the processing of the weights by the non-linear mathematical functions. But there are multiple techniques for identifying which of the inputs contributed most to the high probability assigned to a suspicious transaction. The process of backpropagation itself relies on a measure of which weights in which units contributed most to the final verdict. A related technique called sensitivity analysis can be used to find out which of the variables in the suspicious transaction the network was most sensitive to. Mr. Bruce Fisher of ACI Worldwide, which uses PRCE as its learning algorithm, reports that ACI uses the probability density function mentioned earlier to assign three reason codes to a suspicious transaction. These reason codes describe the characteristics that made the point in the multi-dimensional feature space occupied by the transaction suspicious. If you know the three top reasons that a transaction was reported as suspicious, that gives you a good handle on what you should be looking at when you investigate it.

An additional issue related to the difficulty of understanding how a neural network works internally is that managers may want an explanation of how the system works to make a purchasing decision, and regulators (or, worse, courts) may want an explanation to justify its use. To deal with this, the techniques for figuring out why the network has identified a particular transaction as suspicious can be used to analyze which factors the system is most sensitive to in general. But the most powerful arguments for the use of neural networks are demonstrations of their effectiveness in identifying suspicious transactions, both those provided by the vendor, and, perhaps more importantly, those generated during benchmarking at your site during and after product evaluation.

Summary of Strengths and Weaknesses of Neural Networks

The natural strengths of neural networks include:

- ability to handle complex relationships among variables (including non-linear relationships) can result in a high degree of predictive accuracy;
- natural handling of continuous variables; and
- successful use in fraud detection, a domain that shares many of the characteristics of money laundering detection.

The natural weaknesses of neural networks include:

- difficulty in understanding why the system identified a particular transaction as suspicious (mitigated by the ability to identify the most likely factors that led to the identification);
- difficulty in understanding how the system generally arrives at its results and in explaining the internal workings of the system to managers and regulators (mitigated by the ability to identify which factors the system is most sensitive to and to cite track record statistics if the system is relatively very accurate);
- difficulty of massaging input variables into the range of 0 to 1; and
- possibility of converging prematurely on a solution that is locally but not globally optimal.

If a neural network package were to turn out to be even slightly better at accurately identifying suspicious transactions as a result of its ability to represent complex relationships among the independent variables (not to say that it necessarily would be), that might more than make up for the difficulty in understanding its internal functioning. Trade-off decisions such as this are endemic to these technologies.

Statistical Profiling

Statistical profiling is conceptually more familiar than decision tree or neural network approaches, given that we are not going to go into the mathematics of statistical analysis. In order to evaluate a transaction, statistical profiling applies a statistical measure of variance to determine the extent to which the transaction deviates from previous transactions. Variance can be measured for many variables. A transaction can be evaluated based on a number of characteristics, such as, its absolute amount, its amount compared to the amounts of other transactions within a particular time period (daily, weekly, monthly), transaction frequency during the period the transaction is made, and relationships with counter-parties inside or even outside the institution. A transaction can be evaluated against past patterns of behavior for the account that made the transaction, for the account and its related accounts, and for accounts in the account's peer group.

Statistical profiling systems use various principles to determine an account's peer group. It is not unusual for a statistical profiling vendor to look to the institution at which it is implementing its package to define the groups based on account characteristics that the institution specifies as part of the configuration process. Once the peer groups are specified, their behavior can change dynamically and variance will be measured based on the current behavior of the group. Mr. Mark Kramer of NetEconomy makes the following analogy. "If a flock of birds is flying together, and they all make a sharp turn, there is nothing unusual going on with the birds that are turning. What's unusual and needs to be investigated is when one of the birds veers off from the flock, even by going straight when the rest of the birds turn." You would not be judged to be behaving in an unusual way if you went over your normal credit card balance at Christmas time. In fact, since many retail businesses experience a surge of activity during the Christmas season, if an account holder claims their account is for a retail business, but it does not show any change in activity at Christmas, the lack of change may merit attention.

One issue with classical statistical techniques such as linear, logistic, quadratic functions is that they do not in and of themselves have ways of representing complex relationships among independent variables. Vendors may have proprietary algorithms to use relationships between variables to come up with a final score that indicates the probability that a transaction should be classified as **Investigate? = Yes** or **Investigate? = No** by comparing the combined probability with a threshold level, but working with relationships among a large number of independent variables is not intrinsic to the statistical profiling approach, as it is to decision trees and neural networks.

Like decision trees and neural networks, statistical profiling can be used in conjunction with list checking and processing rules that flag transactions based on characteristics belonging to known patterns of money laundering. This is important for recognizing patterns such as the opening of an individual account followed by a cash deposit just under the reporting limit followed by a wire transfer out of the account followed by a substantial period of inactivity.

Another issue with the statistical profiling approach is that there is no intrinsic way to flag out-of-pattern transactions as false positives. Some vendors offer facilities to address this. For example, they may allow you to mark an account as a trusted account, exempting it from further testing, but that is a blunt instrument that could be dangerous. Another approach would be to use the trusted flag to reduce the probability scoring of the transaction rather than exempt the account from checking altogether. Some vendors also offer a facility for removing a suspicious transaction that has been investigated and found to be legitimate from an account's history so that it doesn't skew the pattern that the system will compare transactions against in the future. For example, when an individual purchases a home, you will want to remove the abnormally large cash transactions associated with the purchase from future statistical evaluation of the account's history.

A strength of the statistical profiling approach is that it is part and parcel of the system that it profiles normal transactions and reports departures from the norm. This approach naturally flags unusual transactions that could be part of new money laundering patterns, thus detecting new patterns that it hasn't been programmed to recognize.

As a result of using departure from the norm as its method of classifying transactions, overfitting takes on a somewhat different complexion with the statistical profiling approach. Of course, in order to identify what to report, it has to have some internal way to separate normal transactions from abnormal transactions. The contours of normal and abnormal transactions fit together like two irregularly shaped pieces in a jigsaw puzzle. The system has to recognize the boundary between the two pieces to know which side of the boundary to put a transaction on. In the case of statistical profiling systems, this would normally be the done through the process of configuration and benchmarking on what we have been calling a training data set. Configuration involves choosing the independent variables to be tested, adjusting the statistical algorithms including their sensitivity levels, and setting up account groupings for peer group comparisons, among other things. The configuration will then be tuned to detect the known instances of money laundering in the training data set with maximum accuracy (perhaps erring on the side of false positives rather than false negatives). After this process is completed, the configuration will be tested against other data sets, analogous to the test data set and the evaluation data set, to ensure that the configuration is correct.

Of course, as with all of these approaches, if data items other than those chosen in the configuration process are significant either in existing or in new patterns of money laundering, the system will not be able to recognize the suspicious transactions on the basis of those variables.

Summary of Strengths and Weaknesses of Statistical Profiling

The natural strengths of statistical profiling include:

- intuitive methodology that's easy to understand and to explain to managers and regulators (in principle, though not at the level of the statistical formulas);
- natural tendency to recognize new patterns of money laundering, since it flags anything that is outside of the norm;
- ease of understanding why a transaction was identified as suspicious (unless it was flagged due to relationships between the independent variables, see below);
- natural handling of continuous variables;
- efficient execution, as the calculations required for statistical analysis are easily processed by the computer; and
- minimal need to re-code data during configuration compared to other approaches (although it will probably be necessary to identify groups of accounts and transaction types for peer group comparisons).

The natural weaknesses of statistical profiling include:

- ability to consider complex relationships between independent variables is not part of the basic approach, and therefore it must be incorporated to whatever extent possible using statistical algorithms that are generally proprietary, and are therefore no more understandable than the operation of neural networks;
- difficulty in handling categorical variables with more than a single category; and
- difficulty in configuring some items such as account groups and transaction types in order to produce the best possible results.

Conclusion

Transaction monitoring software can be a valuable part of your overall compliance strategy. It provides the ability to analyze transactions as they occur by comparing their relevant characteristics to known money laundering patterns and to historical patterns in the transaction account and accounts similar to the transaction account. Partly as a result of computerization in all spheres of financial institutions' operations, especially in larger institutions, huge volumes of transactions are executed every day, and even every minute. Humans unaided by computers are simply unable to sort through the resulting mountains of data.

The sophisticated software approaches we discussed in this article – approaches that go beyond list checking and rules processing – help enable institutions to catch transactions that should be investigated. In money laundering, as soon a scheme is recognized and becomes generally known, clever criminals invent new techniques. In this article, we've discussed how decision trees, neural networks, and statistical profiling can spot transactions that may be using one of these new techniques even though the technique had not yet been invented when the system was implemented.

Unfortunately, these systems are expensive, and even after you've decided that a system of this kind should be part of your overall strategy, it is not easy to determine which system would be most appropriate for your institution. There are many factors to consider besides which approach a vendor uses to drive its core money laundering detection engine. You need to consider how the package will integrate with the institution's overall software architecture, how much effort it will take to install and configure, how the workflow component of the system fits in with your own workflow, and other factors outside the software itself, such as the support offered by the vendor and the vendor's long-term financial stability. Not to mention price.

Even the much narrower question of which of the three approaches we've discussed would be most appropriate is a hard one. As we've seen, each of the approaches has natural strengths and weaknesses, but how the approach is implemented can be even more important than which approach it is. That includes both how the approach is implemented in the package and how it is implemented at your institution.

All of the packages are heavily dependent on configuration. Which independent variables are chosen during the set up is particularly critical. Factors such as how data is massaged so that it can be processed by the system and how such things as peer groups of accounts are chosen are also very important. Installation of most of these systems at each institution involves a process very much like the process of training and adjustment we described throughout the article for each of the approaches:

- after the choice of independent variables and the initial configuration, the system is run against a training data set of cases comprised of the values of the independent variables and pre-identified correctly coded results;
- the system is tuned so that it can correctly predict the results from the independent variables;
- the system is run against other data sets to make sure it hasn't over-learned the peculiarities of the training data (see discussion of overfitting in the body of the article).

Implementation of these steps is an art as much as it is a science.

The idea of a bake-off, during which you would benchmark each of the systems, is an intuitively appealing solution to the challenge of evaluation. It's a great solution, but, unfortunately, benchmarking itself is expensive for your institution, even if you could get multiple vendors to do the work on their end for little or no charge. The process of choosing independent variables, extracting the data, and massaging the data so that it is in a form the different packages will accept would be costly. An additional problem is that the results may not be entirely clear-cut, since different technologies and configurations may work better on different data, and you would presumably be doing the benchmarking against one set of data.

While your understanding of the technological approaches involved in these packages should only be one factor in a software selection decision of this magnitude, your informed opinion regarding the technology hopefully can play some role in your decision making process, help you talk about the decision with managers, regulators, vendors, and staff, and can also make you feel more comfortable with the decision you ultimately make.