Data Mining:
An Inside View

Avi Drissman
Dr. Sethi
CSC 496
June 13, 1997

Corporate computers are pack rats, saving every last bit of data generated during the work they do processing transactions and running their businesses. For the most part, the companies were content to cooperate, adding storage capacity to hold the ever-expanding collection of facts. Recently, though, managers have started looking at their stashes of data as precious resources. Lost in the jungle of facts might be some tidbit worth knowing, some piece of information that might allow them to manage more efficiently, save millions of dollars, and maybe even get them a promotion.

Data mining is the process of finding important data, trends, associations and other noteworthy and useful patterns in stored data [2]. Statistics plays an important role in data mining, by analyzing large data sets and determining whether patterns detected are relevant. Another side of data mining is machine learning, an artificial intelligence (AI) field whose aim is to produce algorithms to "see" patterns in data similar to what humans do. Both are supported by database management, to handle the large quantities of data being analyzed.

Knowledge Discovery in Databases

Of course, to blindly apply data mining techniques to raw data is foolish. Not only must the data be prepared, but the right algorithm or technique must be decided on based on the objective. The whole process of performing data mining, including preparation beforehand and all analysis of the results afterward is called Knowledge Discovery in Databases (KDD) [3].

The steps generally involved in KDD are [3]:

1. Selection
2. Preprocessing
3. Data mining
4. Interpretation and evaluation

The step of selection [3] involves determining which records of a database to feed to the data mining algorithm used. All the records might be selected to conduct a massive search for unknown rules or relationships. Alternatively, if there was some significant change made in the quality of the data, perhaps only the more recent data

would be useful.

Preprocessing [3] involves the steps involved in preparing the data for the algorithm used. Removal of outliers, which are data falling far outside the expected range, is often performed here. If the data mining procedure used is intolerant of errant values, then they need to be dealt with here. Of course, if the object of the data exploration is to examine the data extremes, then the outliers may actually be the important data we seek and shouldn't be disturbed. The type of action to take on outliers often must be decided case by case. In addition, if the database has empty fields, a strategy to either approximate them or ignore them must be taken here.

At this point, we are finally ready to take the data and analyze it [3]. Based on the desired goal, an algorithm is picked that suits the data involved and can provide the types of results that are needed. This step of data mining takes up surprisingly little of the KDD effort.

And finally, some interpretation and evaluation of the results is required [3]. Not all algorithms provide results that are understandable immediately. Someone (or something) must apply statistical tests to see if the discovered patterns are relevant, eliminate redundant conclusions, and provide the end user with understandable results.

Only after the entire KDD process is complete, then, can the discovered knowledge be taken and used to predict trends and identify patterns. Since the steps of KDD are fairly straightforward and don't have many variations, this paper concerns itself with the discussion of different algorithms involved in the data mining step.

There are many different objectives that could be selected for data mining [4, 5]. For database segmentation, the objective is to find how the data naturally groups itself into clusters. Link analysis is used to determine the relationships between pieces of data and to find interesting sequences of occurrences. And in deviation detection we find data items with values that are statistically questionable in order to target problems. But in this paper we concern ourselves with predictive modeling, with which we try to determine what causes membership in a class.

Predictive modeling

Predictive modeling uses the database as a source of information about the past in an attempt to understand the causes of an event or property. More specifically, in predictive modeling, the algorithm used is presented with records, tagged as positive or negative examples of some category or class. The job of the algorithm is to induce which attributes and which values are required for a record to have in order for it to be a member of the category [4, 5]. The entire database might be tagged, or a subset of the records might be used in order to deduce rules to make predictions for the rest of the database. There are several algorithms that can be used for the job of prediction; which one to use depends on the quantity and type of data and what kinds of results are desired.

k-Nearest Neighbors

The k-nearest neighbor algorithm is one of the simplest algorithms. With k-nearest neighbors, the set of tagged records *is* the rule that is used to predict whether or not certain records would fall into the category.

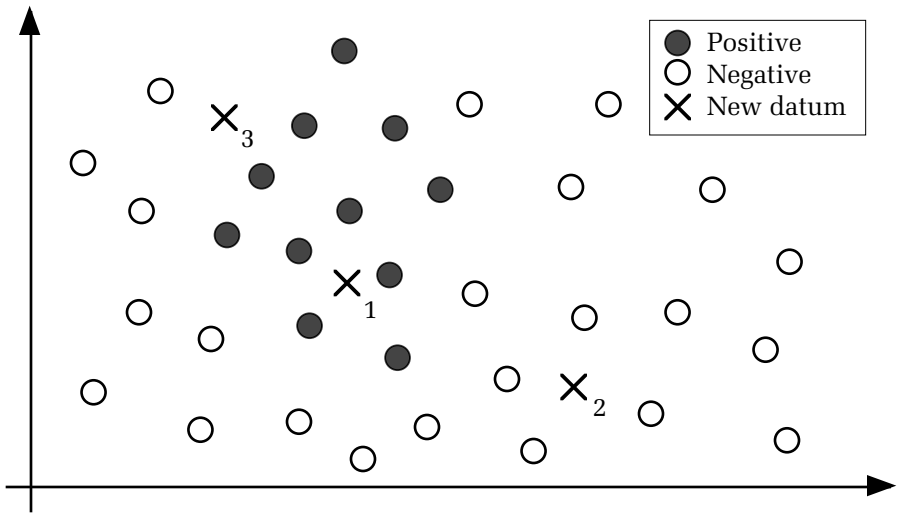

Figure 1: New data are compared with existing data in k-nearest neighbors. If three neighbors were compared in this case, data items 1 and 3 would be judged to be in the class, while item 2 would not.

In k-nearest neighbors, the *k* data items from our original set that are the "closest" to our new data item are determined, and the new data item is ruled to be a member of the class if a majority of its k-nearest neighbors are also a member of the class.

The advantages of this scheme are easy to see. It takes very little up-front processing time, and it is easy to understand conceptually. Unfortunately, the disadvantages are severe. The most important problem is that there is no rule that is induced. The rule "if it's similar to others of the class, it belongs to the class" does not provide any better understanding of the problem than what existed at the beginning of the analysis. Also, finding distances between a new data item and its neighbors can be more compute intensive than deciding membership in the class based on some real rule.

Another problem is that relative importance of the attributes can be taken into consideration by weighting the distances. The weights used, however, must come from experimentation or prior knowledge, since the algorithm does not calculate the weights from the data. And finally, the concept of "distance" does not lend itself well to attributes that are not continuous but belong to discrete classes. Therefore, k-nearest neighbors can be useful for a quick-and-dirty analysis of a small data set, but it is inappropriate for use on a large database.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are a creation of the artificial intelligence field of computer science. Since, according to biology, the brain is composed of millions of independent processing units each sending simple messages to many others, an analogous computing model was tried in artificial intelligence. The resulting ANNs are successful and useful in solving problems in the lab and the real world, but probably have been abstracted and evolved beyond similarity to the real brain.

The form of ANNs most used in data mining is the multi-layer perceptron (MLP), which is often used successfully as a classifier. In a MLP, there is a layer of input nodes, one for each attribute, a layer of output nodes, one for each output class, and one or more layers in-between. For each node, each of the inputs has a weight (of importance). All the inputs to the node are weighted and added. The result is passed through a squashing function which normalizes the output to between 0 and 1 or –1 and 1. Finally, the result is passed as an input to the next layer.

For MLPs to recognize and classify data, it must be trained. A technique called error

back-propagation is used, in which the attributes of a data item are applied to the inputs of the MLP and the output classification is examined. If the MLP misclassifies the item, the algorithm proceeds backwards through the connections, and adjusts the weights of the connections that helped cause the faulty classification. The process is then repeated until the error rate is down to an acceptable level. Once the MLP starts providing satisfactory results, the training can stop and the network put to use. (For more information on ANNs, two good introductory texts are [1] and [6].)

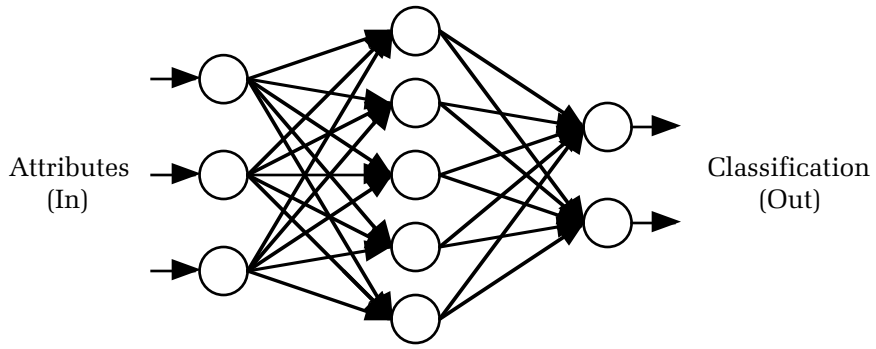

Figure 2: This MLP has an input layer of three units, a hidden layer of five, and an output layer of two. Each layer feeds the results only to the next layer.

At first, such an ANN would seem ideally suited for this job—just train it on the tagged data, and let it loose to predict the unclassified data. But although the results could potentially be excellent, there are several areas which require extreme caution. First, it takes a lot of data to properly train an ANN. There are techniques such as introducing a bit of noise to create more data than are available, but without enough data the network cannot achieve its potential. Also, there is the danger of overtraining the network. If the ANN is trained on a particular set for too long, it can start to learn the subtle quirks of the set and start to lose accuracy on real-world cases. Another important issue in using ANNs for data mining is that the rules that it uses for classifying the data is encoded in the weights of the connections between the nodes, and extracting that information in some kind of human-readable form is often difficult or impossible. So, if explicit rules will be needed, then an ANN might not be suitable. Otherwise, if there is enough data, an ANN could be a good choice for prediction.

Rule induction

Rule induction is a generic term covering the group of algorithms that attempt to discern, from the given database, rules which determine the classes of data items. The

algorithms are all different, and dependent on a particular representation of rules, but they all share a common base of ideas and concepts.

Each item in the database is marked as to whether it belongs to a certain class. Each item also has a set of attributes, called the *predicting attributes* [7]. From these predicting attributes an initial rule to predict whether the item is in the class is formed. The rule is then slowly modified by generalization and specialization until its accuracy is satisfactory.

Generalization of a rule [7] is the process of changing a rule so that it covers more objects. For example, moving from the rule “If seagull then can fly” to “If gull then can fly” is a generalization. Similarly, a specialization of a rule is a modification of a rule which decreases the number of cases it covers.

There are two main approaches [7] to rule induction: bottom-up and top-down. In bottom-up, the initial rule is simply all the positive examples in the set of data. Then, since the rule is very specific and complex, it is repeatedly generalized until it is general enough to be of use but is still accurate.

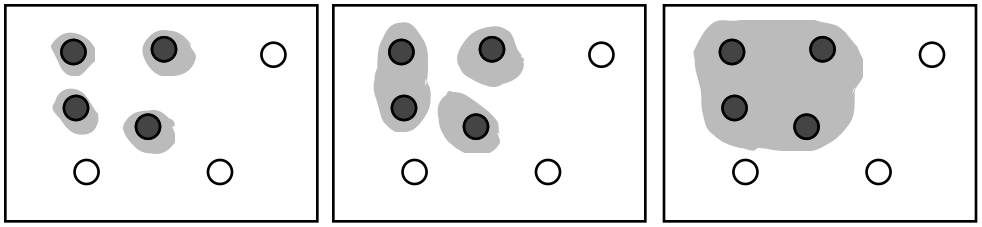

Figure 3: Bottom-up learning. Light gray indicates the rule. [7]

In contrast, an approach taken in other systems is the top-down method [7]. In top-down, an initial rule is taken. Then, step by step, the rule is modified, using both generalization and specialization, until it is of sufficient accuracy.

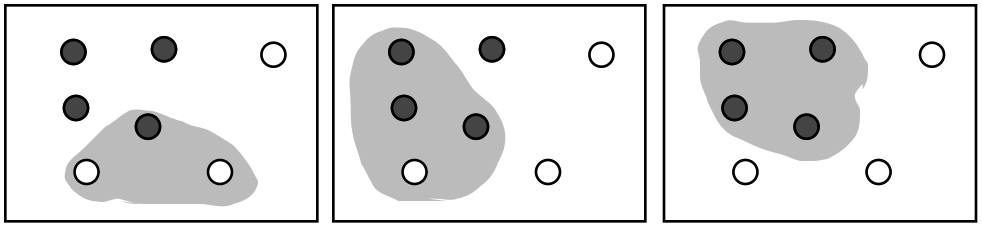

Figure 4: Top-down learning. Light gray indicates the rule. [7]

The prime advantage of the rule induction techniques is that they produce actual rules that both people and other programs can use to predict the class membership of other data items in the future.

Genetic algorithms

Genetic algorithms [7] are another area in which computer science was inspired by biology. As with rule induction, there are several ways to represent the data; we present here only the common concepts underlying the differing implementations.

An encoding [7] for rules is chosen such that each rule, which determines which data items are in a class based on the values of the predicting attributes, is represented by a string. Then an initial population of rule-strings is generated. The "fitness" of a rule to live is defined as how well it classifies the data items into the pre-defined classes. A rule that does a good job of classifying the data is considered very fit, while another which classifies poorly is unfit. For each generation of rules, the fitness of each rule is determined, and the rules which are least fit are removed from consideration.

The next generation of rules [7] is created by taking the most fit rules of the current generation and performing two different operations on them: *crossover*, in which two parent rule-strings combine to form a new string (e. g. characters 1–3 from parent 1, 4–17 from parent 2, 18–31 from parent 1, etc.), and *mutation*, which is a random change in a rule-string.

Overall,

> Genetic Algorithms outperform traditional learning techniques, especially when the descriptions that have to learned are complex.... However, GAs suffer from two important drawbacks: firstly, they outperform traditional techniques only when almost no domain knowledge is available. Although domain knowledge can be incorporated in the GA, ... the use of domain knowledge is limited, compared to traditional techniques. A second drawback is the number of evaluations. A GA typically requires 500–1000 samples of the search space, i.e. about 10 generations of 50–100 organisms each, before it finds qualitative good solutions. [7]

Therefore, genetic algorithms are a very useful tool, as long as they are used in a context in which they are appropriate.

Decision trees

Decision trees are sets of questions hierarchically arranged in such a way that answering a series of questions about a data item will enable it to be classified correctly. There are many algorithms available to create decision trees based on classified data; we will consider Quinlan's famous ID3 system.

We can draw an analogy between using a decision tree and playing the children's game "Twenty Questions." In Twenty Questions, one player picks any object. Another player must deduce the identity of the object (i.e. classify it), and in order to do so may ask the first player up to twenty yes-or-no questions. Since asking twenty times, "Are you thinking of *x*?" for some category *x* is unlikely to hit the answer, an alternative strategy must be used.

A successful approach to the Twenty Questions problem is to ask about the attributes of the item. For example, if it has been determined that the item is an animal, then specific questions such as "Does it fly?" and "Does it swim?" provide important information. The order of questions is also important, because initial questions must provide as much useful information as possible. Continuing our example, a first question of "Is it alive?" is much better than "Does it fly?" because the category of living creatures is large, and eliminating it from consideration or limiting consideration to it is very useful, whereas flying objects (comprising birds, insects, kites, etc.) is a much smaller class, and eliminating them from consideration is not worthwhile. On the other hand, "Does it fly?" might be an excellent second question if the object is alive because it partitions the set of living objects well.

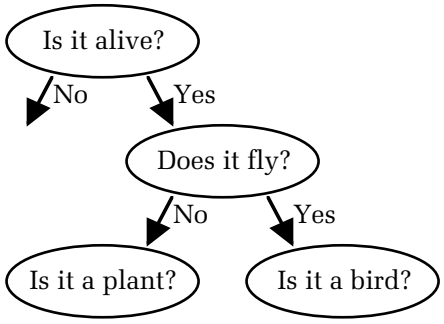

Figure 5: A Twenty Questions decision tree

A similar task faces an algorithm building a decision tree. It has access to a database of objects already tagged as belonging to one of *n* classes, and must figure out a hierarchy of questions to ask in order to efficiently determine to which class an object belongs.

One of the most famous decision tree algorithms is Quinlan's ID3 (Standing for *Induction of Decision Trees*) [7]. ID3 uses a top-down strategy, which means it splits the whole training set into pieces instead of working from the classes, and its choices are irrevocable, which means once it makes a decision it never changes it. The tree is constructed as follows [7]:

1. An attribute is selected as the root of the tree, and branches are made for all different values this attribute can have;
2. The tree is used to classify the training set. If all examples at a particular leaf belong to the same class, this leaf is labeled with this class. If all leaves are labeled with a class, the algorithm terminates;
3. Otherwise, the node is labeled with an attribute that does not occur on the path to the root, and branches are created for all possible values. The algorithm continues with step 2.

Since its choices are irrevocable and it wants a small tree, ID3 has to, at steps 1 and 3, pick the attribute that best divides the data set. Information theory [7, 8] says that given an item in a sample set *S*, which belongs to one of two classes *P* and *N*, the amount of information needed to correctly classify it is

$$I(p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

where there are p elements of S in P, and n elements of S in N. If using attribute A as the root of the tree will create v partitions, then the amount of information needed to classify an item of S given A as the root is the weighted average [7, 8]:

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

where set i has p_i elements of P and n_i elements of N. To have a first decision that provides the most information, we must pick the attribute A which minimizes the value of $E(A)$. We can repeat this process for each node in the tree.

Given noiseless data, ID3 can perform well, and many modifications made to it over the years have added features that improve its performance with real-world data. Since it produces rules that are human-understandable, it or one of its descendants can prove itself useful.

Conclusion

Years ago, computers helped create what was heralded as the information age—and soon people realized that the information was proliferating out of control. It is far too soon to declare that a combination of statistics and artificial intelligence will save us from the onslaught of data, but for now it is the only solution in sight.

References

1. Aleksander, Igor and Helen Morton. *An Introduction to Neural Computing*. Second Edition. Thompson Computer Press, London, 1995.

2. Data Mining, http://www.rpi.edu/~arunmk/dm1.html

3. Fayyad, Usama, Gregory Piatetsky, and Padhraic Smyth. "The KDD Process for Extracting Useful Knowledge from Volumes of Data." *Communications of the ACM*. 39(11), 27–34.

4. Gerber, Cheryl. "Dissecting Datamining." *Datamation*. May 1, 1996. Available from http://www.datamation.com/PlugIn/issues/1996/may1/DISSECTINGDATAMINING11.html

5. —. "Excavate Your Data." *Datamation*. May 1, 1996. Available from http://www.datamation.com/PlugIn/issues/1996/may1/05asoft3.html

6. Hertz, John, *et al*. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, Redwood City, California, 1991.

7. Holsheimer, Marcel, Arno Siebes. "Data Mining: The Search for Knowledge in Databases." Report CS-R9406, CWI, Amsterdam, The Netherlands, 1994. Available from ftp://ftp.cwi.nl/pub/CWIreports/AA/CS-R9406.ps.Z

8. Wüthrich, Beat. "Knowledge Discovery in Databases." Hong Kong University of Science and Technology, Kowlook, Hong Kong, 1996. Available from ftp://ftp.cs.ust.hk/pub/techreport/96/tr96-04.ps.gz

Yes, this paper is mine—I wrote it myself without assistance of any kind. Despite the pain, suffering, and anguish endured during its creation, I did not plagiarize *anything at all*, and all material I used from outside sources is properly attributed to the best of my knowledge and ability. This package is sold by weight, not volume. Some settling of contents may have occurred during shipping and handling.

_____

Avi Drissman