A Linear Method for Deviation Detection in Large Databases

Andreas Arning
IBM German Software Development Laboratory
Boeblingen, Germany

Rakesh Agrawal and **Prabhakar Raghavan**
IBM Almaden Research Center
San Jose, California 95120, U.S.A.

Abstract

We describe the problem of finding deviations in large data bases. Normally, explicit information outside the data, like integrity constraints or predefined patterns, is used for deviation detection. In contrast, we approach the problem from the inside of the data, using the implicit redundancy of the data.

We give a formal description of the problem and present a linear algorithm for detecting deviations. Our solution simulates a mechanism familiar to human beings: after seeing a series of similar data, an element disturbing the series is considered an exception. We also present experimental results from the application of this algorithm on real-life datasets showing its effectiveness.

Index Terms: Data Mining, Knowledge Discovery, Deviation, Exception, Error

Introduction

The importance of detecting *deviations* (or *exceptions*) in data has been recognized in the fields of Databases and Machine Learning for a long time. Deviations have been often viewed as outliers, or errors, or noise in data. There has been work in Statistics on identifying outliers (e.g. (Hoaglin, Mosteller, & Tukey 1983) (Johnson 1992)). There has been work in extending learning algorithms to cope with a small amount of noise (e.g. (Aha, Kibler, & Albert 1991)). Additionally, some work has been done to determine the impact of erroneous examples on the learning results: there is experimental work as in (Quinlan 1986) and quantitative theoretical work as in (Angluin & Laird 1988), extending the valuable work of (Valiant 1984).

Rather than being considered outliers that need to be tolerated during the main-line processing, deviations play the leading part in this paper: the one and only purpose of our proposed method is to discover them. Usually, explicit information sources residing outside the data like integrity constraints (Chamberlin 1996) or predefined error/non-error patterns (Val 1995) are used for deviation detection. We approach the problem from the inside of the data, using the implicit redundancy in the data to detect deviations.

The problem of detecting deviations can be considered to be a special case of clustering of data into two clusters: deviations and non-deviations. However, the usual clustering methods (see (Shavlik & Dietterich 1990)) are biased toward discarding deviations as noise; we, in contrast, try to isolate small minorities. Moreover, these methods generally require the existence of a metrical distance function between data elements. Such is the case with even refined clustering algorithms as (Fisher 1987) (Hanson & Bauer 1989) (Rumelhart & Zipser 1985). Michalski (Michalski & Stepp 1983) replaces the usual Euclidian distance measure between data elements by enriching the measure with conceptual similarity, but still requires a symmetrical distance measure and imposes restrictive constraints on permissible representations. We, on the other hand, only require a function that can yield the degree to which a data element causes the "dissimilarity" of the data set to increase. This function does not have to fulfill the conditions to be metrical.

The literature on Kolmogorov complexity (Li & Vitanyi 1991) is relevant to the problem of deviation detection. We are looking for the subset of data that leads to the greatest reduction in Kolmogorov complexity for the amount of data discarded. Our problem is also related to the Minimum Description Length (MDL) principle (Rissanen 1989). The MDL principle states that the best model for encoding data is the one that minimizes the sum of the cost of describing the data in terms of the model and the cost of describing the model.

Below we give a new measure for deviation detection and an algorithm for detecting deviation. The proposed algorithm has linear complexity. This is a desirable property for a data mining algorithm, as we know from (Agrawal, Imielinski, & Swami 1993) that presents a method for discovering frequent cooccurrences of attribute values in large data sets. But the goal of that algorithm is quite different, and the performance of this system ((Agrawal & Srikant 1994)) is a consequence of ignoring early the seldom occurring values – thus the possible exceptions.

Problem Description

We begin by giving a formal definition of the deviation detection problem.

Exact Exception Problem

Given:

- a set of items I (and thus its power set $\mathcal{P}(I)$);
- a *dissimilarity* function $\mathcal{D}$: $\mathcal{P}(I) \rightarrow \mathbb{R}_0^+$;
- a *cardinality* function $\mathcal{C}$: $\mathcal{P}(I) \rightarrow \mathbb{R}_0^+$, with $I_1 \subset I_2 \Rightarrow \mathcal{C}(I_1) < \mathcal{C}(I_2)$ for all $I_1, I_2 \subseteq I$.

Define for each $I_j \subset I$, the *smoothing factor*:

$$SF(I_j) := \mathcal{C}(I - I_j) * (\mathcal{D}(I) - \mathcal{D}(I - I_j)).$$

We say that $I_x \subset I, I_x \neq I$ is an *exception set* of I with respect to $\mathcal{D}$ and $\mathcal{C}$ if

$$SF(I_x) \geq SF(I_j) \text{ for all } I_j \subset I.$$

Notes

- Intuitively, an *exception set* is the subset of items that contributes most to the dissimilarity of the itemset I with the least number of items.
- The *smoothing factor* indicates how much the dissimilarity can be reduced by removing a subset I_j of elements from the set I. Observe that SF may be negative for some subset I_j if the dissimilarity of $I - I_j$ is higher than that of the original set I.
- While $\mathcal{D}(I - I_x)$ — the dissimilarity of the reduced set — is important in finding the exception set I_x, the dissimilarity $\mathcal{D}(I_x)$ of the exception set itself is not considered. Nevertheless, it may be valuable to take into account $\mathcal{D}(I_x)$ in further evaluation of the exception set.
- For $\mathcal{D}$, any function can be used that returns a low value if the elements of I are similar to each other, and a higher value if the elements are dissimilar. This function does not require a metrical distance between the items.
- In the most simple case, the function $\mathcal{C}(I - I_j)$ may return $|I - I_j|$, the number of items in the set $I - I_j$. However, to favor small exception sets, this function may be defined by a formula such as $\mathcal{C}(I - I_j) = \frac{1}{|I_j|+1}$.
- The exception set need not be unique. It can also be empty, e.g. if all items in I are identical. The smoothing factor is 0 in that case.

Example Let

- the set I be the set of integer values {1,4,4,4};
- the dissimilarity function $\mathcal{D}$: $\mathcal{P}(I) \rightarrow \mathbb{R}_0^+$ be the variance of the numbers in the set, i.e. $\frac{1}{N} \times \sum_{i=1}^{N} (x_i - \overline{x})^2$;
- the cardinality function $\mathcal{C}$: $\mathcal{P}(I) \rightarrow \mathbb{R}_0^+$ be the count of elements in the set.

By computing for each candidate exception set I_j the smoothing factor $SF(I_j)$, we get:

I_j	$I - I_j$	$\mathcal{C}(I - I_j)$	$\mathcal{D}(I - I_j)$	$SF(I_j)$
{}	{1,4,4,4}	4	1.69	0.00
{4}	{1,4,4}	3	2.00	–0.94
{4,4}	{1,4}	2	2.25	–1.13
{4,4,4}	{1}	1	0.00	1.69
{1}	{4,4,4}	3	0.00	5.06
{1,4}	{4,4}	2	0.00	3.38
{1,4,4}	{4}	1	0.00	1.69

Thus, the candidate set I_j = {1} and only this one qualifies as *exception set* I_x with respect to the specified $\mathcal{C}$ and $\mathcal{D}$ functions.

Sequential Exception Problem

There are choices of the functions $\mathcal{C}$ and $\mathcal{D}$ for which the exact exception problem is trivial to solve. There are others for which it is non-trivial but still polynomial-time. For example, given a set of numbers, consider finding the subset whose removal results in the greatest reduction in the variance of the residual set, per element removed (or any similar cardinality function). A priori it is not clear how to do this, but the following schema works: sort the numbers in decreasing order of the distance from their mean. Discard the numbers one by one in this order and at each point compute the smoothing factor. After we have examined all the numbers in this fashion, we would have seen the optimal subset at some point. This example is interesting because it is a case where the optimal order for considering subsets can be computed efficiently.

However, one cannot hope to always find an efficient ordering for examining the elements that yields the optimal solution for every $\mathcal{C}$ and $\mathcal{D}$. Indeed, there exist $\mathcal{C}$ and $\mathcal{D}$ for which the problem is NP-hard by a reduction from Maximum Independent Set in a graph (Garey & Johnson 1979). This motivates the definition of the sequential exception problem.

Sequential Exception

Given:

- a set of items I (and thus its power set $\mathcal{P}(I)$);
- a sequence S of n subsets $I_1, I_2, ..., I_n$ with $2 \leq n \leq |I|$, $I_j \subseteq I$, and $I_{j-1} \subset I_j$;
- a *dissimilarity* function $\mathcal{D}_S$: $\mathcal{P}(I) \rightarrow \mathrm{I\!R}_0^+$ with respect to S;
- a *cardinality* function $\mathcal{C}$: $\mathcal{P}(I) \rightarrow \mathrm{I\!R}_0^+$ with $I_1 \subset I_2 \Rightarrow \mathcal{C}(I_1) < \mathcal{C}(I_2)$ for all $I_1, I_2 \subseteq I$.

Define for each I_j in S, the *smoothing factor*:

$$SF'(I_j) := \mathcal{C}(I_j - I_{j-1}) * (\mathcal{D}_S(I_j) - \mathcal{D}_S(I_{j-1})).$$

We say that $I_x \subset I, I_x = (I_j - I_{j-1}), j > 1$ is a *sequential exception set* of I with respect to $\mathcal{D}_S$, $\mathcal{C}$, and the sequence S if

$$SF'(I_x) \geq SF'(I_j) \text{ for all } I_j \text{ occurring in } S.$$

Observe that the smoothing factor SF for each subset I_j considers the dissimilarity difference with the preceding subset in S instead of considering the dissimilarity of the complementary set $(I - I_j)$ and I.

The sequential exception simulates a mechanism familiar to human beings: after seeing a series of similar data, an element disturbing the series is considered an exception, even without being told explicitly the rules of well-formedness.

Algorithm

The algorithm we propose uses for S the sequence that selects elements of I in the order they appear in input (or a randomization of it). That is, I_{j+1} is derived from I_j by including the next item in the input. In this case, $n = |I|$. We also require that the dissimilarity function be such that $\mathcal{D}(I_j)$ is incrementally computable from the result produced by $\mathcal{D}(I_{j-1})$ and the jth item using constant space. The algorithm is as follows:

```
[1] Perform steps [2] and [3] $m$ times on randomized
    input ($m$ is an algorithm parameter).

[2] Performs steps [2.1] through [2.3]:

    [2.1] Get the first element $i_1$ of the item set $I$
          making up the one-element subset $I_1 \subseteq I$,
          and compute $\mathcal{D}_S(I_1)$.

    [2.2] For each following element $i_j \in (I - I_{j-1})$ in $S$,
          create the subset $I_j$ taking $I_j = I_{j-1} \cup \{i_j\}$;
          compute the difference in dissimilarity values
          $d_j = \mathcal{D}_S(I_j) - \mathcal{D}_S(I_{j-1})$.

    [2.3] Consider that element $i_j$ with the maximal value
          of $d_j > 0$ to be the answer for this iteration;
          if $d_j \leq 0$ for all $I_j$ in $S$, there is no exception.

[3] If an exception $i_j$ is found in step [2.3], do step [3.1]:

    [3.1] For each element $i_k, k > j$ compute
          $d_{k_0} = \mathcal{D}_S(I_{j-1} \cup \{i_k\}) - \mathcal{D}_S(I_{j-1})$ and
          $d_{k_1} = \mathcal{D}_S(I_j \cup \{i_k\}) - \mathcal{D}_S(I_j)$.
          Add to $I_x$ those $i_k$ for which $d_{k_0} - d_{k_1} \geq d_j$.

[4] We now potentially have $m$ competing exception
    sets $I_x$, corresponding to each iteration. Select the
    one with largest value of difference in dissimilarity
    $d_j$, scaled with the cardinality function $\mathcal{C}$.
```

Notes

- Steps [2.1] through [2.3] are performed m times on after randomizing the order of items in input to alleviate the sensitivity of results to the order of input.
- For a motivation for Step [3.1], consider a degenerate scenario. Suppose i_j is such that $\mathcal{D}_S(I_j) - \mathcal{D}_S(I_{j-1})$ is large so that i_j is a candidate exception. Now let i_{j+1} be identical to i_j. In that case, $\mathcal{D}_S(I_{j+1}) - \mathcal{D}_S(I_j)$ will be 0 and i_{j+1} will not be considered an exception. In general, there may be items i_k following i_j that would have caused increase in the dissimilarity, but for the presence of i_j. Step [3.1] adds all such items to the exception set.

Complexity

If the number m of runs with randomized order is kept small, and if the user provided function $\mathcal{D}$ is able to compute $\mathcal{D}(I_j)$ incrementally from $\mathcal{D}(I_{j-1})$ and i_j using constant time and space, then all steps but the randomizing step are constant in space and $O(N)$ in time consumption. Here N is the number of items in the input.

With random access to the data, the randomizing step can be done in $O(N)$ time. If we can rely on an already randomized sequence, we could do instead a random choice of the start record and process the data in wrap-around manner. In that case, this pseudo-randomizing step would also work in linear time.

Experimental Evaluation

We present next the results of applying the proposed algorithm on real-life datasets in order to examine if the algorithm indeed succeeds in isolating deviations.

The datasets are taken from the "UCI Repository of Machine Learning Databases and Domain Theories" (obtainable from `ics.uci.edu` via anonymous ftp in the directory /usr2/spool/ftp/pub/machine-learning-databases). We also consider the "fortune cookies" of IBM AIX version 2.2 and a table of IBM stock prices.

Each line in these datasets is considered an item, and the goal is to identify lines that violate frequently occurring character patterns. The same dissimilarity function was used in all the tests.

Dissimilarity Function

The dissimilarity function handles the comparison of character strings. It maintains a pattern in the form of a regular expression that matches all the character strings seen so far. Starting with a pattern that is the same as the first string seen, the pattern may have to be weakened by introducing wildcard characters as more strings are seen to cover them. The dissimilarity function captures this weakening in the covering pattern; the dissimilarity increases when the pattern covering the $j-1$ strings in I_{j-1} does not cover the jth string $\in (I_j - I_{j-1})$.

The dissimilarity function is defined as:

$$\mathcal{D}_S(I_1) := 0;$$

$$\mathcal{D}_S(I_j) := \mathcal{D}_S(I_{j-1}) + j \times \frac{\mathcal{M}_S(I_j) - \mathcal{M}_S(I_{j-1})}{\mathcal{M}_S(I_j)}$$

where $\mathcal{M}_S(I_j)$ is computed from the string pattern that covers all elements:

$$\mathcal{M}_S(I_j) := \frac{1}{3 \times c - w + 2}$$

with c being the total number of characters and w being the number of needed wildcards. For details, see (Arning 1995).

The auxiliary function $\mathcal{M}_S(I_j)$ computes a user customizable maximum value for the elements of I_j in order to find those elements that particularly increase this maximum. In the definition above, it grows with an increasing number of wildcards and a decreasing number of constant characters in a pattern.

Note that a weighting is done in the computation of the distance function by the position j of the string in input; this weighting captures that a weakening of the pattern is quite likely for elements with low index and improbable for elements with high index.

Results

The results reported below show different types of exceptions found on very different datasets. Note that the same dissimilarity function (outlined above) was used in all the tests. To give an idea of the data layout, before giving the exceptions discovered, we list a few initial items of the dataset.

shuttle-landing-control.data

```
2,*,*,*,*,*,2
1,2,*,*,*,*,1
1,1,2,*,*,*,1
1,1,1,*,*,*,1
1,1,3,2,2,*,1
1,*,*,*,*,4,1
2,1,4,*,*,1,1
2,1,4,*,*,2,1
2,1,4,*,*,3,1
2,1,3,1,1,1,1
2,1,3,1,1,2,1
2,1,3,1,2,1,1
2,1,3,1,2,2,1
1,1,3,1,1,3,1
2,1,3,1,2,3,1
```

shuttle-landing-control.data – found exceptions

```
2,*,*,*,*,*,2
```

echocardiogram.data

```
11,0,71,0,0.260,9,4.600,14,1,1,name,1,0
19,0,72,0,0.380,6,4.100,14,1.700,0.588,name,1,0
16,0,55,0,0.260,4,3.420,14,1,1,name,1,0
57,0,60,0,0.253,12.062,4.603,16,1.450,0.788,name,1
19,1,57,0,0.160,22,5.750,18,2.250,0.571,name,1,0
26,0,68,0,0.260,5,4.310,12,1,0.857,name,1,0
13,0,62,0,0.230,31,5.430,22.5,1.875,0.857,name,1,0
50,0,60,0,0.330,8,5.250,14,1,1,name,1,0
19,0,46,0,0.340,0,5.090,16,1.140,1.003,name,1,0
25,0,54,0,0.140,13,4.490,15.5,1.190,0.930,name,1,0
10,1,77,0,0.130,16,4.230,18,1.800,0.714,name,1,1
52,0,62,1,0.450,9,3.600,16,1.140,1.003,name,1,0
52,0,73,0,0.330,6,4,14,1,1,name,1,0
44,0,60,0,0.150,10,3.730,14,1,1,name,1,0
0.5,1,62,0,0.120,23,5.800,11.67,2.330,0.358,name,1
24,0,55,1,0.250,12.063,4.290,14,1,1,name,1,0
0.5,1,69,1,0.260,11,4.650,18,1.640,0.784,name,1,1
0.5,1,62.529,1,0.070,20,5.200,24,2,0.857,name,1,1
22,1,66,0,0.090,17,5.819,8,1.333,0.429,name,1,0
1,1,66,1,0.220,15,5.400,27,2.250,0.857,name,1,1
...
```

echocardiogram.data – found exceptions

```
,?,?,77,?,?,?,?,?,2,?,name,2,?
```

Fortune Cookies

```
* UNIX is a trademark of AT&T Bell Laboratories.
1 bulls, 3 cows.
10.0 times 0.1 is hardly ever 1.0.
A Puritan is someone who is deathly afraid that so
A bad compromise is better than a good battle.  --
A bird in hand is worth two in the bush.  -- Cerva
A bird in the bush can't relieve itself in your ha
A clash of doctrine is not a disaster -- it is an
A conservative is one who is too cowardly to fight
A fanatic is one who can't change his mind and won
A foolish consistency is the hobgoblin of little m
...
```

Fortune Cookies – found exceptions

```
Password:
Quack!
login:
```

Observe that the exception set in this case consist of three items. The algorithm could identify them as exceptions because they are the only ones with no blanks and caused a pattern of the form "* * * *" to be weakened to "*"

IBM stock prices

58/07/01	368.50	313.23	.022561
58/07/02	369.50	314.08	.022561
58/07/03	369.25	313.87	.022561
58/07/04	Market closed		
58/07/07	370.00	314.50	.022561
58/07/08	369.00	313.65	.022561
58/07/09	368.00	312.80	.022561
58/07/10	368.25	313.02	.022561
58/07/11	368.50	313.23	.022561
58/07/14	360.00	306.00	.022561
58/07/15	356.00	302.60	.022561
58/07/16	359.50	305.58	.022561
58/07/17	359.50	305.58	.022561
58/07/18	357.50	303.88	.022561
58/07/21	361.00	306.85	.022561
58/07/22	363.00	308.55	.022561
58/07/23	366.50	311.53	.022561
58/07/24	368.00	312.80	.022561
58/07/25	370.00	314.50	.022561
58/07/28	370.25	314.72	.022561
...			

IBM stock prices – found exceptions

58/07/04	Market closed	(317 instances)
59/01/06	2.5% Stock Dividend	(2 instances)
59/05/05	50% Stock Split	(7 instances)
73/10/09	IBM not traded	(1 instance)
88/09/26	Data suspect	(30 instances)

Out of the 9415 records, an exception set of 357 elements was created. We have shown only one exemplar of each exception type. Note that the different instances of an exception type are not duplicates, but differ in the date part.

A Failure and its Analysis

Applying the same dissimilarity function to the following data set, the algorithm did not identify any exception:

house-votes-84.data

```
republican,n,y,n,y,y,y,n,n,n,y,?,y,y,y,n,y
republican,n,y,n,y,y,y,n,n,n,n,n,y,y,y,n,?
democrat,?,y,y,?,y,y,n,n,n,n,y,n,y,y,n,n
democrat,n,y,y,n,?,y,n,n,n,n,y,n,y,n,n,y
democrat,y,y,y,n,y,y,n,n,n,n,y,?,y,y,y,y
democrat,n,y,y,n,y,y,n,n,n,n,n,n,y,y,y,y
democrat,n,y,n,y,y,y,n,n,n,n,n,n,?,y,y,y
republican,n,y,n,y,y,y,n,n,n,n,n,n,y,y,?,y
republican,n,y,n,y,y,y,n,n,n,n,n,y,y,y,n,y
democrat,y,y,y,n,n,n,y,y,y,n,n,n,n,n,?,?
republican,n,y,n,y,y,n,n,n,n,n,?,?,y,y,n,n
republican,n,y,n,y,y,y,n,n,n,n,y,?,y,y,?,?
democrat,n,y,y,n,n,n,y,y,y,n,n,n,y,n,?,?
democrat,y,y,y,n,n,y,y,y,?,y,y,?,n,n,y,?
republican,n,y,n,y,y,y,n,n,n,n,n,y,?,?,n,?
republican,n,y,n,y,y,y,n,n,n,y,n,y,y,?,n,?
democrat,y,n,y,n,n,y,n,y,?,y,y,y,?,n,n,y
democrat,y,?,y,n,n,n,y,y,y,n,n,n,y,n,y,y
republican,n,y,n,y,y,y,n,n,n,n,n,?,y,y,n,n
democrat,y,y,y,n,n,n,y,y,y,n,y,n,n,n,y,y
...
```

However, manual inspection reveals some exceptional items containing the occasionally occurring '?' symbol for the vote; the most outstanding element with many of these seldom values is:

house-votes-84.data – *not* found exception

```
republican,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
```

The reason for this exception not being found is the nature of the dissimilarity function used. For almost any order of the input data, after processing a few data elements the pattern to cover them is changed to "*e*c*a*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*"; once two different values "...,n,..." and "...,y,..." are seen in a column, the pattern takes the form "...,*,...". From then on, there is no change in pattern when a "?" appears in the same column as the pattern covers it.

Can we capture these types of exceptions with a more powerful dissimilarity function? We can use a function that maintains for each column some maximum number of values (to enable the membership test with constant space and time consumption), beyond which we use a wildcard symbol for the column. Each column has a partial dissimilarity value, which when summed over all columns gives the total dissimilarity value. Indeed, when using such a dissimilarity function for tabular data, the algorithm could discover the cited exception.

Conclusions and Future Work

This paper is an attempt to give a formal framework to the problem of deviation detection in order to stimulate further work and discussion in the KDD community on this important data mining operation. We also presented a linear algorithm for the sequential exception problem. Our solution simulates a mechanism familiar to human beings: after seeing a series of similar data, an element disturbing the series is considered an exception. A dissimilarity function captures how dissimilar is a new data item from the data items seen so far. This algorithm is suitable for large data bases found in data mining.

Experimental evaluation shows that the effectiveness of our algorithm in isolating deviations depends on the dissimilarity function used. Obviously, if we know the nature of exception we are looking for, we can design a very effective dissimilarity function. But in that case, we do not need a general algorithm and the most effective solution would be to write a special program to do the task. On the other hand, based on experiments with several dissimilarity functions on several datasets, our feeling is that it will be difficult to have a universal dissimilarity function that works well for all datasets.

Nevertheless, it seems possible and helpful to have some predefined dissimilarity functions that can handle typical classes of exceptions found in real-life datasets. The deviation detection algorithm then should work in conjunction with automatic selection of the appropriate function that promises the best results. This could be done by triggering the selection based on the properties of the input data, as well as by conditional selection depending on results of previous runs. Working out the specifics and building such a system is a fruitful direction for future research.

References

Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the VLDB Conference.*

Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering* 5(6):914–925.

Aha, D. W.; Kibler, D.; and Albert, M. K. 1991. Instance-based learning algorithms. *Machine Learning* 6(1):37–66.

Angluin, D., and Laird, P. 1988. Learning from noisy examples. *Machine Learning* 2(4):343–370.

Arning, A. 1995. *Fehlersuche in großen Datenmengen unter Verwendung der in den Daten vorhandenen Redundanz.* PhD dissertation, Universität Osnabrück, Fachbereich Sprach– und Literaturwissenschaft.

Chamberlin, D. 1996. *Using the New DB2: IBM's Object-Relational Database System.* Morgan Kaufmann.

Fisher, D. H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2(2):139–172.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: a guide to the theory of NP-completeness.* W. H. Freeman.

Hanson, S. J., and Bauer, M. 1989. Conceptual clustering, categorization, and polymorphy. *Machine Learning* 3(4):343–372.

Hoaglin, D.; Mosteller, F.; and Tukey, J. 1983. *Understanding Robust and Exploratory Data Analysis.* New York: John Wiley.

Johnson, R. 1992. *Applied Multivariate Statistical Analysis.* Prentice Hall.

Li, M., and Vitanyi, P. 1991. *Kolmogorov Complexity.* Springer Verlag.

Michalski, R. S., and Stepp, R. E. 1983. Learning from observation: conceptual clustering. In Michalski et al. (1983). 331–363.

Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., eds. 1983. *Machine Learning: An Artificial Intelligence Approach*, volume I. Los Altos, California: Morgan Kaufmann.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106.

Rissanen, J. 1989. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publ. Co.

Rumelhart, D. E., and Zipser, D. 1985. Feature discovery by competitive learning. *Cognitive Science* 9:75–112.

Shavlik, J. W., and Dietterich, T. G., eds. 1990. *Readings in Machine Learning*, Series in Machine Learning. Morgan Kaufmann.

Vality Technology Inc. 1995. *Integrity product Overview.*

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM* 27(11):1134–1142.