Quantifiable Data Mining Using Principal Component Analysis

Flip Korn, Alexandros Labrinidis, Yannis Kotidis,
Christos Faloutsos*, Alex Kaplunovich, Dejan Perković

{flip,labrinid,kotidis,christos,kaplunov,dejanp}@cs.umd.edu

Department of Computer Science
University of Maryland at College Park

Abstract

Association Rule Mining algorithms operate on a data matrix (*e.g.*, customers $\times$ products) to derive rules [2, 22]. We propose a single-pass algorithm for mining *linear rules* in such a matrix based on *Principal Component Analysis*. PCA detects correlated columns of the matrix, which correspond to, *e.g.*, products that sell together.

The first contribution of this work is that we propose to quantify the "goodness" of a set of discovered rules. We define the "guessing error": the root-mean-square error of the reconstructed values of the cells of the given matrix, when we pretend that they are unknown. The second contribution is a novel method to guess missing/hidden values from the linear rules that our method derives. For example, if somebody bought \$10 of milk and \$3 of bread, our rules can "guess" the amount spent on, say, butter. Thus, we can perform a variety of important tasks such as forecasting, 'what-if' scenarios, outlier detection, and visualization. Moreover, we show that we can compute the principal components with a single pass over the dataset.

Experiments on real datasets (*e.g.*, NBA statistics) demonstrate that the proposed method consistently achieves a "guessing error" of up to 5 times lower than the straightforward competitor.

1 Introduction

Data Mining has recently been receiving increasing interest [9], of which the quintessential problem is association rule mining [2]. Given a data matrix, with, *e.g.*, customers for rows and products for columns, we want to find rules. Existing algorithms find rules of the form $\{bread, milk\} \Rightarrow butter$, meaning that customers who buy "bread" and "milk" also tend to buy "butter". What distinguishes database work from AI/Machine Learning and statistics work is its emphasis on large datasets. The initial association rule mining paper by Agrawal et al. [2], as well as all the follow-up database work [4], proposed algorithms to minimize the time to extract these rules through clever record-keeping to avoid additional passes over the dataset, through parallelism, *etc.*

*This research was partially funded by the Institute for Systems Research (ISR), by the National Science Foundation under Grants No. EEC-94-02384, IRI-9205273 and IRI-9625428.

What is novel about the present work is that it attempts to assess *how good* the derived rules are, an issue that has not been emphasized in in the database literature. We propose the "guessing error" as a measure of the "goodness" of a given set of rules for a given dataset. The idea is to pretend that a cell of the matrix is "hidden" from us, and to try to guess the missing value using the derived rules; the root-mean-square guessing error (averaged over all the cells of the given matrix) indicates how good a set of rules is. The second major innovation of this work is the use of principal component analysis (PCA) to derive *linear rules*, of the form "*customers typically spend 1:2:5 dollars on bread:milk:butter*". Linear rules can be easily used for extrapolations, and can help determine hidden, missing and corrupted values. We provide novel algorithms for estimating missing values, even if multiple values are simultaneously missing/hidden.

PCA (via linear rules) can support the following applications, thanks to their ability to reconstruct missing/hidden values:

- Data cleaning: reconstructing lost data and repairing noisy or damaged data;
- Forecasting: '*if a customer spends $1 on bread and $2.50 on ham, how much will s/he spend on mayonnaise?*';
- What-if scenarios and decision support: '*We expect doubled demand of Cheerios; how much milk should we stock up on?*';
- Outlier detection: '*Which customers* deviate *from the typical sales pattern?*';
- Visualization: PCA, being identical to the Karhunen-Loeve transform [8], is the optimal dimensionality reduction method, mapping the rows of the data matrix in to 2- or 3-dimensional points, that can be plotted to reveal the structure of the dataset (*e.g.*, clusters, linear correlations, *etc.*);

The paper is organized as follows: Section 2 discusses past work. Section 3 presents an introduction to PCA. Section 4 introduces the proposed method. Section 5 gives the experimental results. Section 6 provides a discussion. Section 7 gives some conclusions and pointers to future work.

2 Related Work

Agrawal et al. distinguish between three data mining problems: identifying classifications, finding sequential patterns, and discovering association rules [1]. We review only material relevant to the latter since it is the focus of this paper. See [7] for an excellent, recent survey of all three problems.

The seminal work of [2] introduced the problem of discovering association rules and presented an efficient algorithm for mining them. Since then, new serial algorithms [4, 15, 19] and parallel algorithms [14, 3, 10] have been proposed. In addition, generalized association rules has been the subject of recent work [21, 11].

The vast majority of association rule discovery techniques are basically Boolean, since they discard the quantities of the items bought and only pay attention to whether something was bought or not. A notable exception is the work of Srikant and Agrawal [22], where they address the problem of mining quantitative association rules. Their approach is to partition each quantitative attribute into a set

Symbol	Definition
N	number of records
M	number of attributes
k	cutoff (number of principal components retained)
h	number of holes
$\mathcal{H}$	set of cells which have holes
RMS_1	root-mean-squared error over each hole
RMS_h	root-mean-squared error over h holes
$\times$	matrix multiplication
$\mathbf{X}$	the $N \times M$ data matrix
$\mathbf{X}_c$	the centered version of $\mathbf{X}$
$\mathbf{X}^t$	the transpose of $\mathbf{X}$
$x_{i,j}$	value at row i and column j of the matrix $\mathbf{X}$
$\hat{x}_{i,j}$	reconstructed (approximate) value at row i and column j
$\bar{\mathbf{x}}$	the mean cell value of $\mathbf{X}$
$\mathbf{C}$	the $M \times M$ covariance matrix ($\mathbf{X}_c^t \times \mathbf{X}_c$)
$\mathbf{V}$	the $M \times k$ PC matrix

Table 1: Symbols, definitions and notation used in this paper.

of intervals which may overlap, then apply techniques for mining Boolean Association Rules. In this framework, they aim for rules such as

$$< bread : [3 - 5] > and < milk : [1 - 2] > \;\Rightarrow\; < butter : [1.5 - 2] >$$

The above rule says that customers that spend between 3 to 5 dollars on bread and 1 to 2 dollars on milk, tend to spend 1.5 to 2 dollars on butter. See Section 6.3 for a comparison of this method versus our proposed method.

Traditional criteria for selecting association rules are based on the support-confidence framework [2]; recent alternative criteria include the chi-square test [6] and probability-based measures [20]. Related issues include outlier detection and forecasting. See [13] for a textbook treatment of both, and [5] and [12] for recent developments.

3 Principal Component Analysis

The proposed method is based on *principal component analysis*. PCA is a popular and powerful operation in statistical analysis [13]. It is identical to the Karhunen-Loeve transform from pattern recognition [8].

3.1 Intuition behind PCA

In our running example, we have N customers and M products organized in an $N \times M$ matrix $\mathbf{X}$, where the entries are the dollar amount spent by customer i on product j. Table 1 gives a list of symbols used from here on and their definitions. To make our discussion more concrete, we will use rows and "customers" interchangeably, and similarly for columns and "products". Of course, the proposed method is applicable to any $N \times M$ matrix, with a variety of interpretations for the rows and columns, *e.g.*, patients and medical-test-measurements (blood pressure, body weight, *etc.*); documents and terms (typical in Information Retrieval [18]), *etc.*

Each row vector of the matrix can be thought of as an M-dimensional point. Given this set of N points, PCA identifies the axes (orthogonal directions) of greatest variance, after centering the points about the origin. Figure 1 illustrates an example of an axis that PCA finds. Suppose that we have M=2 dimensions; then our customers are 2-d points, as in Fig. 1. The corresponding direction x' that PCA suggests is shown. The meaning is that, if we are allowed only k=1, the best direction to project on is the direction of x'. The direction x' is, in effect, a linear 'rule' that governs the correlation between money spent on the products, based on customer purchasing activity. In this case, the projection of a data point on the x' axis gives the overall "volume" of the purchase. For the setting of Figure 1, the coordinates of the first PC = (0.866, 0.5) imply the linear rule "*bread* : *butter* $\Rightarrow$ \$.50 : \$.866", that is, for the most of our customers (2-d points) the relative spendings bread-to-butter are close to the ratio 0.866:0.5. As we shall see shortly, these *linear rules* can be used for forecasting, 'what-if' scenarios, outlier detection, and visualization. In addition, they are often amenable to interpretation as underlying factors to describe, in this case, purchasing behavior.

customer	bread (\$)	butter (\$)
Billie	.89	.49
Charlie	3.34	1.85
Ella	5.00	3.09
...	...	...
John	1.78	.99
Miles	4.02	2.61

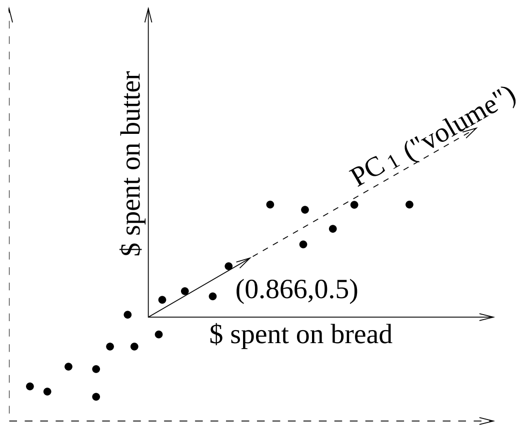


Figure 1: A data matrix in table form and its counterpart in graphical form, after centering (original axis drawn with dotted lines). As the graph illustrates, PCA identifies the vector $(0.866, 0.5)$ as the "best" axis to project along.

Technically, the directions identified by PCA are the *eigenvectors* of the *covariance matrix* $\mathbf{C}$ (see Eq. 2); each eigenvector has an associated *eigenvalue* whose magnitude indicates the variance of the points along that eigenvector. (See Appendix A for formal definitions of eigenvalues and eigenvectors.) The goal of PCA is to reduce the dimensionality of a dataset while retaining as much variation as possible. PCA does this by identifying the direction of maximum variance (given by the largest eigenvalue/vector) and then incrementally identifying the orthogonal direction with maximum variance (the second eigenvalue/vector, *etc.*). In the end, only the eigenvectors associated with the k largest eigenvalues, namely the *principal components*, are kept. The goal here is to preserve most of the important information while discarding the redundancy. In order to choose the cutoff k of PCs to retain, the simplest textbook heuristic [13, p. 94] is to retain enough eigenvectors so that the sum of their eigenvalues cover 85% of the grand total. That is, choose the cutoff k such that

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{M} \lambda_j} \approx 85\% \tag{1}$$

This is the heuristic that we used in this paper.

4 Proposed Method

The proposed method is based on PCA. In subsection 4.1 we present an efficient, *single-pass* algorithm to compute the k principal components. A fast algorithm is extremely important for database applications, where we expect matrices with several thousands or millions of rows. Subsection 4.2 presents one of the two major contributions of this paper: the introduction of a measure for the "goodness" of a given set of rules. Subsection 4.1 presents the second major contribution: how to use the linear rules (*i.e.*, eigenvectors) of PCA, to predict missing/hidden values.

4.1 A Single-Pass Algorithm for PCA

By definition, PCA needs to compute the eigenvectors of the *covariance* matrix $\mathbf{C}$ of the given $N \times M$ matrix $\mathbf{X}$. The covariance matrix $\mathbf{C} = [c_{ij}]$ intuitively is the "column-to-column" similarity matrix, having high c_{ij} values if the columns i and j are correlated. Mathematically, it is defined as

$$\mathbf{C} \equiv \mathbf{X}_c^t \times \mathbf{X}_c \tag{2}$$

where $\mathbf{X}_c$ is derived by the given $\mathbf{X}$ matrix by subtracting the column average from every cell. That is, $\mathbf{X}_c$ is a zero-mean matrix, or "centered", in the sense that its column averages are all zero.

The covariance matrix $\mathbf{C}$ is a square matrix of side M. To compute the PCs, we have to perform the following steps: (a) zero-mean the input matrix to derive $\mathbf{X}_c$; (b) compute $\mathbf{C}$ from Eq. 2; (c) compute the eigenvalues/vectors of $\mathbf{C}$ and pick the first k. We assume that $\mathbf{C}$ can fit in memory: it needs M^2 cells, where M is the number of columns/products, which should typically be on the order of hundreds or thousands, for real applications. Under this assumption, we can compute the column averages and the covariance matrix with a single-pass over the N ($\approx$ millions) of rows of the given $\mathbf{X}$ matrix, using the algorithm of Figure 2(a).

Once we have the covariance matrix $\mathbf{C}$ in memory, we can use any off-the-shelf eigensystem package to determine its eigenvalues and eigenvectors, as shown in Fig. 2(b).

In conclusion, our algorithm requires a single pass to compute the column averages and the covariance matrix. In more detail, it requires $O(N)$ disk operations to read the matrix $\mathbf{X}$ and $O(NM^2)$ main-memory operations to build the corresponding covariance matrix $\mathbf{C}$. Since typically the number of rows is in the hundreds of thousands (*e.g.*, sales, or customers), and the number of columns in the hundreds (*e.g.*, products, or patient symptoms), our algorithm of Fig. 2 is very efficient. Notice that the algorithms of [3] require more than one pass over the dataset in an attempt to find large itemsets. Also notice that the $O(M^3)$ factor for the eigensystem computation is negligible, compared to the $O(NM^2)$ operations needed to build the covariance matrix, since we assume that $N \gg M$.

```
/* input:  training set X on disk */
/* output:  covariance matrix C */
for j := 1 to M do
    colavgs[j] ← 0;
    for l := 1 to M do
        C[j][l] ← 0;
for i := 1 to N do
    Read ith row of X from disk (X[i][1],...,X[i][M]);
    for j := 1 to M do
        colavgs[j] += X[i][j];
        for l := 1 to M do
            C[j][l] += X[i][j]*X[i][l];
for j := 1 to M do
    colavgs[j] /= N;
for j := 1 to M do
    for l := 1 to M do
        C[j][l] -= N * colavgs[j] * colavgs[l];
```

(a)

```
input: covariance matrix C in main
      memory
output: eigenvectors v_1,...,v_k (i.e.,
      the PCs)
compute eigensystem:
      {v_1,...,v_M} ← eigenvectors(C);
      {λ_1,...,λ_M} ← eigenvalues(C);
      sort v_j according to the
      eigenvalues;
      choose k based on Eq. 1;
      return the k largest
      eigenvectors;
complexity: O(M^3)
```

(b)

Figure 2: Pseudocode for efficient computing of the PCA: (a) single-pass over data matrix and (b) eigensystem computation

4.2 Measuring Goodness: the "Guessing Error"

The association rule mining literature has not defined a criterion to assess the "goodness", or accuracy, of a set of discovered rules. We propose a remedy, through the notion of the "guessing error". The fundamental requirement is that the given set of rules $\mathcal{R}$ allow for estimations of missing values in a given record/row.

The question is: how good is a set of rules $\mathcal{R}$ for a data matrix $\mathbf{X}$? Let's consider a specific row (= customer) $\mathbf{x}_i$ of the matrix, and let's pretend that the j-th attribute is hidden from us (*i.e.*, the amount spend on the j-th product, *e.g.*, bread). Thanks to $\mathcal{R}$ and the rest of the values $x_{i,m}$ ($m \neq j$), we are able to estimate the missing value as $\hat{x}_{ij}$. The *guessing error* for this specific cell (i, j) is $\hat{x}_{ij} - x_{ij}$.

Definition 1 *The guessing error for a set of rules $\mathcal{R}$ on a data matrix $\mathbf{X}$ is defined as the root mean square of the guessing errors of the individual cells, that is*

$$RMS = \sqrt{\frac{1}{NM} \sum_{i}^{N} \sum_{j}^{M} (\hat{x}_{ij} - x_{ij})^2} \tag{3}$$

More specifically, we also define it as the *single-hole guessing error* RMS_1, exactly because we allowed only a single hole at a time. The generalization to the h-hole guessing error RMS_h is straightforward.

We have not yet discussed how the set of rules $\mathcal{R}$ was derived. Using a practice that is common in Machine Learning, we can use a portion $\mathbf{X}_{train}$ of the data set $\mathbf{X}$ to derive the rules $\mathcal{R}$ ("training set"), and some other portion $\mathbf{X}_{test}$ of the data set $\mathbf{X}$ to compute the guessing error ("testing set"). The details of the choice of training and testing sets is orthogonal to our definition, and outside the scope of this work, since they have been extensively examined in the machine learning and classification literature [17]. A reasonable choice is to use 90% of the original data matrix for training and the remaining 10% for testing. Another possibility is the use the entire data matrix for both training and testing. In this paper, we report only the results the former choice because the two choices above gave very similar results.

The ability to measure the goodness of a set of rules $\mathcal{R}$ for a given testing dataset $\mathbf{Y}$ is very important, for developers of data-mining products and for end-users alike:

- For developers, it allows benchmarking and comparison with competing products and designs: a low "guessing error" over a variety of input matrices indicates a good product.
- For end-users that use a given product on a specific dataset, low "guessing error" implies that the derived rules have captured the essence of this dataset, and that they can be used for estimation of truly unknown values with more confidence.

We would like to highlight that the definition of the "guessing error" can be applied to *any type* of rules, as long as they can do estimation of hidden values. In the next subsection we focus on our proposed linear rules, and show how to use them to obtain such estimates.

4.3 Determining Hidden and Unknown Values

Here we illustrate (sketch) the algorithm for determining unknown values of the data matrix. If we can reconstruct holes, then we can find hidden values or forecast future values. This framework is also applicable to 'what-if scenarios' where we can specify some of the values ('*What if the demand for Cheerios doubles?*') and then forecast the effect on other attributes ('*Then the demand for milk will double.*'). In addition, it can be used to discover outliers by hiding a cell value, reconstructing it, and comparing the reconstructed value to the hidden value. The value is an outlier when the value predicted is significantly different from the existing hidden value.

Once the user has specified partial knowledge from a transaction (*e.g.*, the dollar amounts spent by a new customer, for some products, are given), the set of unknowns $\mathcal{H}$ are determined by the k PCs that have been kept. The geometric intuition is the following: the PCs form a k-dimensional hyper-plane in M-space, the "PC-hyperplane", on or close to which the data points lie. The h holes result in an h-dimensional hyper-plane in M-space, the "feasible solution space", on which the solution is constrained. We want to find a point that definitely agrees with our given partial data ("feasible solution space"), and is as close to (or exactly on) the "PC-hyperplane". Figure 3(a) illustrates the case in the simplest possible form: we have M=2 products, k=1 PC, and h=1 hole. Namely, we know (a) that a customer spends the given amount on bread and (b) that most of our previous customers fall on or close to the line defined by the first PC. We want to find the amount spent on butter (the hole). The intersection of "feasible locations" (vertical dashed line) and "expected locations" (solid diagonal line) gives our best prediction for the 2-d point that corresponds to that sale; the value on the "butter" axis, labeled as "guess" is our proposed estimate for the required amount spent on butter.

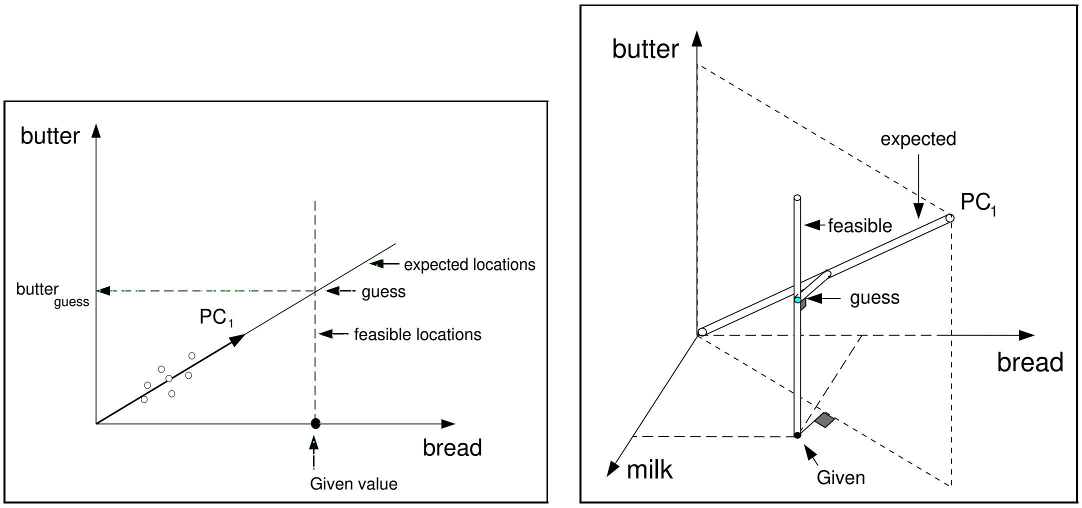
The two hyper-planes correspond to linear equations, which are presented in Appendix A. There are three possibilities regarding their intersection, all of which are illustrated in Fig. 3-4:

CASE 1: (EXACTLY-SPECIFIED) The two hyper-planes intersect at a point. This occurs when

$(k+h) = M$. Here the respective linear equations have an exact solution. Figure 3(a) illustrates an example in $M = 2$ dimensions, for $h = 1$, hole and cutoff $k = 1$ principal component.

CASE 2: (OVER-SPECIFIED) The two hyper-planes do not intersect. This occurs when $(k+h) < M$. The respective equations are overdetermined, and the closest distance between them is chosen for the solution. Figure 3(b) illustrates an example in $M = 3$ dimensions, for $h = 1$ hole and cutoff $k = 2$.

CASE 3: (UNDER-SPECIFIED) The intersection of the two hyper-planes forms a $(min(k,h) - 1)$-dimensional hyper-plane. This occurs when $(k+h) > M$. The respective equations are under-determined. Among the infinite solutions, we propose to keep the one that needs the fewest eigenvectors. Thus, we ignore $(k+h) - M$ PCs to make the system exactly-specified, and then solve it using CASE 1. Figure 4 illustrates an example in $M = 3$ dimensions, for $h = 2$ holes and cutoff $k = 2$.



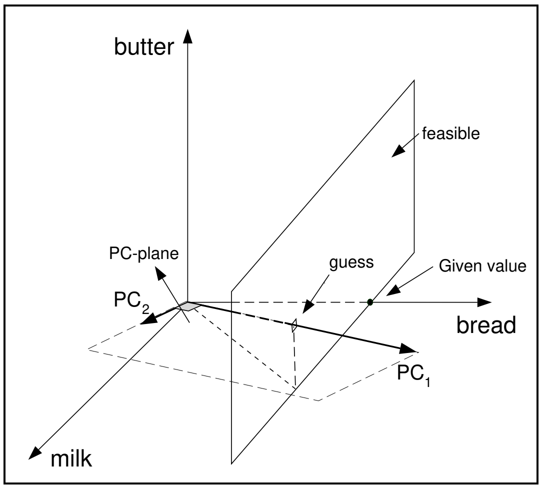
(a) exactly-specified (b) over-specified

Figure 3: Two of the three possible cases: exactly defined, and over-specified

5 Experiments

We ran three sets of experiments. The first was to investigate the prediction accuracy achieved by the proposed PCA method compared to the straightforward competitor; the second was to examine the stability of PCA in estimating more than one simultaneous hole; the last was to see how our method scales up for large datasets.

Methods: We compared PCA with a straightforward technique for predicting values, named 'col-avgs': for a given hole, use the respective column average from the training set. Note that 'col-avgs' is



under-specified

Figure 4: The last possible case: under-specified.

identically the same as if we applied PCA with $k = 0$ eigenvalues. We cannot compare PCA with any association-based methods because, as we argue in Sec. 6.3, association-based methods do not lead to prediction of missing/hidden/corrupted values.

Error Measure: We use the RMS_h "guessing error" described in Sec. 4.2.

Datasets: We ran our experiments on a variety of real datasets, described next. Sec. 6.1 displays scatter-plots of them.

- **'nba'** (459×12) - NBA statistics from the 1991-92 season, including minutes played, field goals, rebounds, and fouls;
- **'baseball'** (1574×17) - batting statistics from major league baseball for four seasons; fields include batting average, at-bats, hits, home runs, and stolen bases; available at `www.usatoday.com/sports/baseball/sbstats.htm`;
- **'abalone'** (4177×7) - physical measurements of an invertebrate animal, including length, diameter, and weights; available at `www.ics.uci.edu/~mlearn/MLSummary.html`.

5.1 Prediction Accuracy

Preliminary to running these experiments, for each dataset we chose 90% of its rows for the training matrix; the remaining 10% were used as the testing matrix. First we computed the PCs of the training matrix, along with the column averages of the training matrix for use as the competitor.

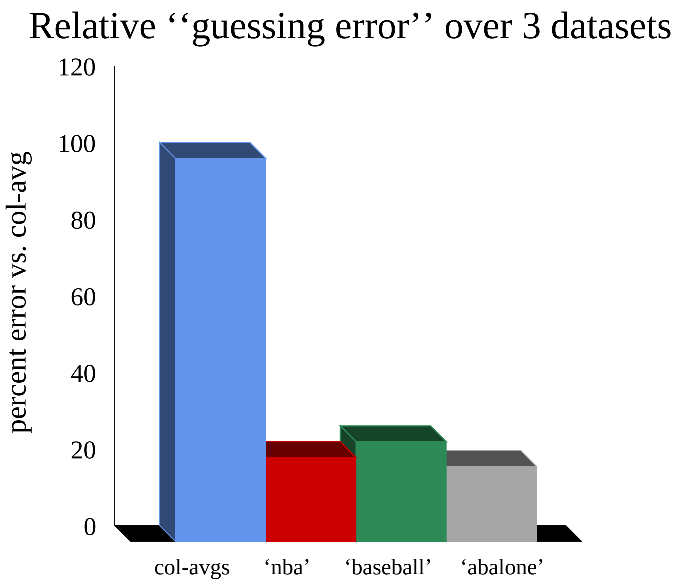


Figure 5: Ratio of guessing error between PCA and 'col-avgs' for **'nba'**, **'baseball'**, and **'abalone'**.

Figure 5 shows the RMS_1 guessing error for the **'nba'**, **'baseball'**, and **'abalone'** datasets, normalized by the guessing error attained by 'col-avgs'. As a frame of reference, we also present the normalized RMS_1 of 'col-avgs', which is, of course, 100%.

In Fig. 6, we show the RMS_h for the **'nba'** and **'baseball'** datasets, for $h \in \{1, 2, 3, 4, 5\}$ holes. The results for the **'abalone'** dataset were similar, and are omitted for brevity. Note that the guessing error is relatively stable for up to several simultaneous holes.

5.2 Scale-up

Figure 7 demonstrates the scale-up of our algorithm. The vertical axis is the average actual computation time to determine the principal components (in seconds), as measured by the `time` utility of UNIX^{TM}. The horizontal axis is the number of data matrix rows N. Since all of our datasets were relatively small ($N < 5000$), we used a 100,000 $\times$ 100 data matrix created using the Quest Synthetic Data Generation Tool available at `www.almaden.ibm.com/cs/quest/syndata.html`. The methods were implemented in 'C' and 'Splus' under UNIX^{TM}. The experiments ran on a dedicated Sun SPARCstation 5 with 32Mb of main memory, running SunOS 4.1.3. The disk drive was a FUJITSU M2266S-512 model 'CRANEL-M2266SA' with minimum positioning time of 8.3 ms and maximum positioning time of 30ms.

The plot is close to a straight line, as expected. The y-intercept of the line is the time to compute the eigensystem. Notice that it seems negligible.

6 Discussion

Here we show the visualization capabilities that PCA offers by presenting 2-d scatter-plots of the datasets used. The plots also provide a sense of how well-described they are by only a few linear rules. Using the

Figure 6: Guessing error vs. number of holes (1-5) for the **'nba'** and **'baseball'** datasets, 'col-avgs' vs. PCA.

'nba' dataset, we demonstrate how these linear rules can be interpreted, with references to the plots. Finally, we present a qualitative comparison of the linear rules of PCA versus general association rules that were the subject of [22].

6.1 Visualization

The application of PCA for visualization has been well-studied, and is well known in the pattern classification and image processing literature as the Karhunen-Loeve (KL) transform [8]. Recall that PCA identifies the axes of greatest variation. By projecting the points onto the top two or three of these axes (*i.e.*, the eigenvectors associated with the largest eigenvalues), the points can be plotted to give an idea of the density and structure of the dataset. For example, Figure 8 shows a scatter-plot of a dataset of **'nba'** which originally included the statistics of N=459 players for M=12 attributes and has been reduced to 2-dimensional PC space (*i.e.*, two principal components). In (a), the x-axis corresponds to the first (and strongest) principal component; the y-axis corresponds to the second principal component. In (b), the x-axis corresponds to the second PC and the y-axis corresponds to the third PC. Most of the points are very close to the horizontal axis, implying that they all closely follow the first eigenvector and are considerably linear. The plot also shows that many of the attributes are correlated with one another, such as field goals and minutes played. There are two points that are clearly outliers: $(3000, 971)$ and $(2100, -1296)$, corresponding to Michael Jordan and Dennis Rodman, respectively. Figure 9 shows 2-d plots for (a) **'baseball'** and (b) **'abalone'**.

6.2 Interpretation of the Linear Rules

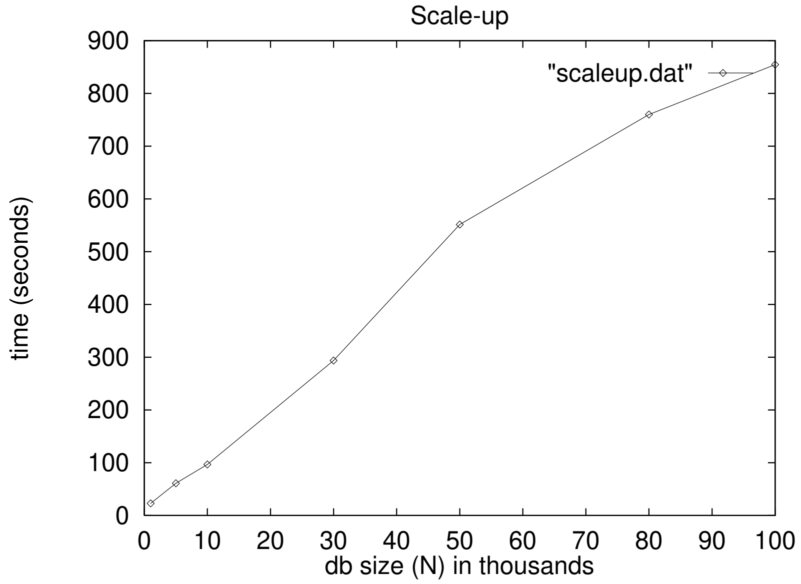


Figure 7: Scale-up: time to compute PCA versus db size N in records.

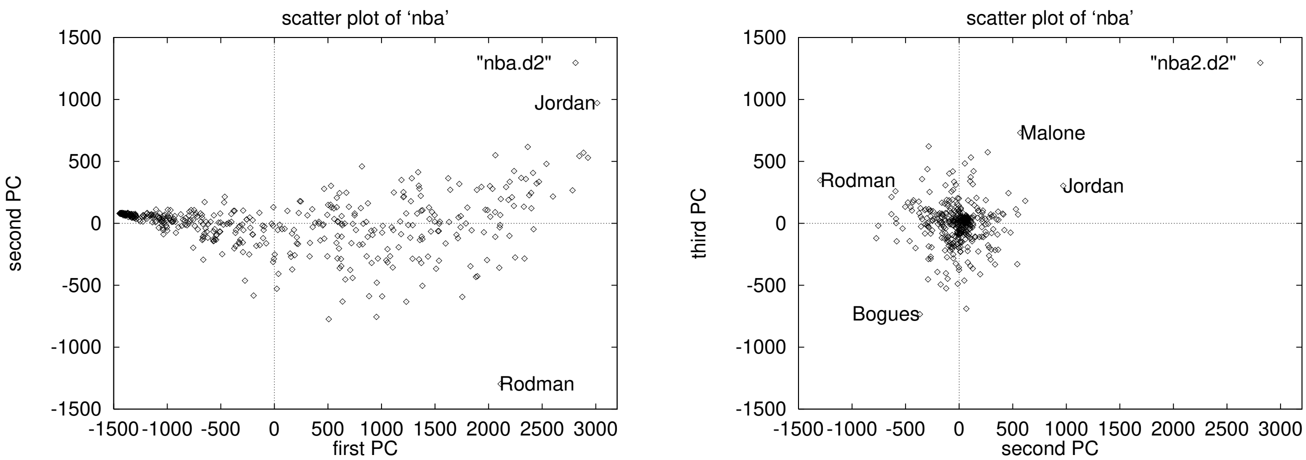
In this section, we illustrate by example how principle components can be interpreted as the underlying factors that govern a dataset. Table 2 presents the first three principal components (PC_1, PC_2, and PC_3) for the **'nba'** dataset (see Fig. 8), after truncating small row values (specifically, cells whose absolute value is less than half the maximum absolute row cell value). For example, PC_1 was truncated to

$$(.808, \approx 0, \ldots, .406, \approx 0, \ldots, \approx 0)$$

By drawing on common knowledge of basketball and by examining these principal components, we conjecture the following: PC_1 represents "court action", separating the starters from those who sit on the bench, and gives a 0.808:0.406 = 2:1 ratio. This is a linear rule with the obvious interpretation: the average player scores 1 point for every 2 minutes of play. (that is, roughly 1 basket for every 4 minutes played). According to PC_1, Jordan was across-the-board the most active player that season

field	PC_1	PC_2	PC_3
minutes played	.808	$-.4$	
field goals			
goal attempts			
free throws			
throws attempted			
blocked shots			
fouls			
points	.406	.199	
offensive rebounds			
total rebounds		$-.489$	.602
assists			$-.486$
steals			$-.07$

Table 2: Relative values of the PCs from **'nba'**.



(a) side view (b) front view

Figure 8: A scatter plot of **'nba'**: two 2-d orthogonal views.

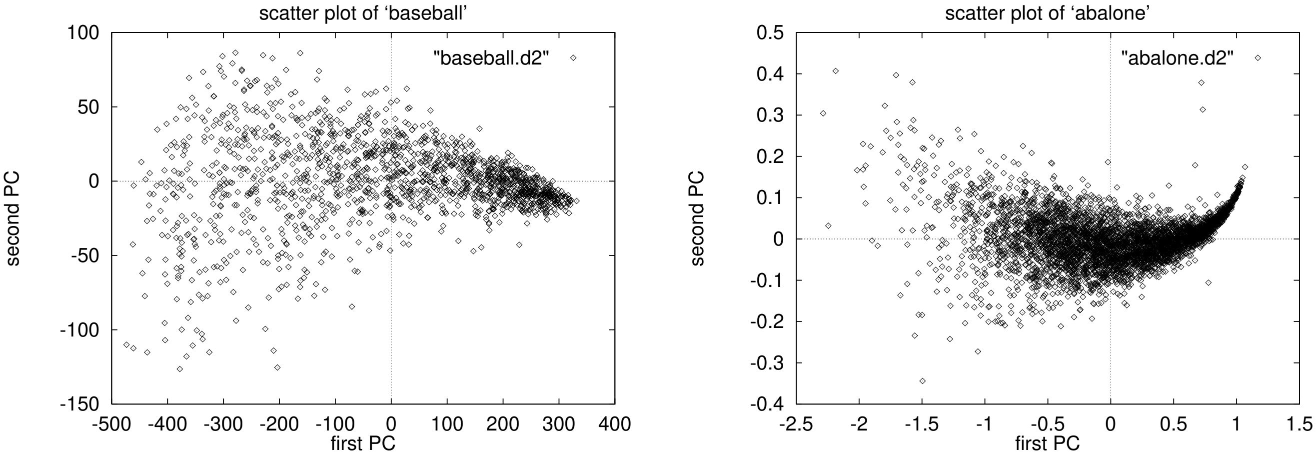
in almost every statistic. PC_2 shows that the number of offense rebounds is negatively correlated with points in a 0.489:0.199 $\approx$ 2.45:1 ratio. Intuitively, this is because a goal attempt makes it difficult for a player to get in a good position for rebounding, and vice versa. Thus, PC_2 roughly represents "field position", separating the guards, who get the most opportunities to shoot, from the forwards, who are more likely to be rebounders. For example, in Fig. 8, we see the extremes among active players: star shooting guard and Michael Jordan at one end with 2404 points and 91 rebounds, and power forward (and excellent rebounder) Dennis Rodman at the other with 800 points and 523 rebounds. PC_3 says that rebounds are negatively correlated with assists and steals. Typically, tall players make better rebounders because they can reach high and short players are better at assists and steals because they can move fast. Thus, PC_3 roughly represents "height", with Mugsy Bogues (5'3") and Karl Malone (6'8") at opposite extremes. (See Figure 8(b)).

6.3 Linear Rules vs. Association Rules

Since we propose a completely different paradigm of rules, namely, linear rules as opposed to association rules, it is important to discuss the qualitative differences between the two. Specifically, we are concerned with the following types of rules:

- Boolean association rules [2]: *e.g.*, $\{bread, milk\} \Rightarrow butter$
- quantitative association rules [22]: *e.g.*, $< bread : [2-5] > \;\Rightarrow\; < butter : [1-2] >$
- linear rules: *e.g.*, ratio of spendings *bread:butter = 2:3*

Boolean association rules have the advantages that they are easy to interpret and relatively easy to implement. On the weak side, a given data matrix $\mathbf{X}$ with, say, amounts spent per customer per product, is converted to a binary matrix by treating non-zero amounts as plain "1"s. This simplifies the data mining algorithms but tends to lose valuable information.



(a) **'baseball'** (b) **'abalone'**

Figure 9: Scatter plots of (a) **'baseball'** and (b) **'abalone'** in 2-d PC space.

Quantitative association rule algorithms perform an important step to retain the above information. Figure 10(a) illustrates how these rules might work for a fictitious dataset with a few customers (points) and $M = 2$ products only, namely, "bread" and "butter". In this dataset, the quantitative association rules will derive rules that corresponds to the dashed rectangles of the figure. For example, the first two lower-left rectangles will yield the rules

$$< bread : [1-3] > \;\Rightarrow\; < butter : [.5-2.5] >$$
$$< bread : [3-5] > \;\Rightarrow\; < butter : [2-3] >$$

Linear rules, for the same setting of Figure 10 and with $k = 1$ PC, will fit the best possible line through the dataset; its unit vector is exactly the first PC of the given data matrix. Thus, the corresponding rule will look like

$$bread : butter = .81 : .58$$

Intuitively, we conjecture that the quantitative association rules should supersede the Boolean association ones, which are not discussed further. Comparing the linear rules with the quantitative association rules, we identify the following strong points for each approach.

The advantage for the quantitative association rules are

- They will be more suitable if the data points form clusters.
- They have been applied to categorical data, although similar extensions of PCA are discussed in [13].

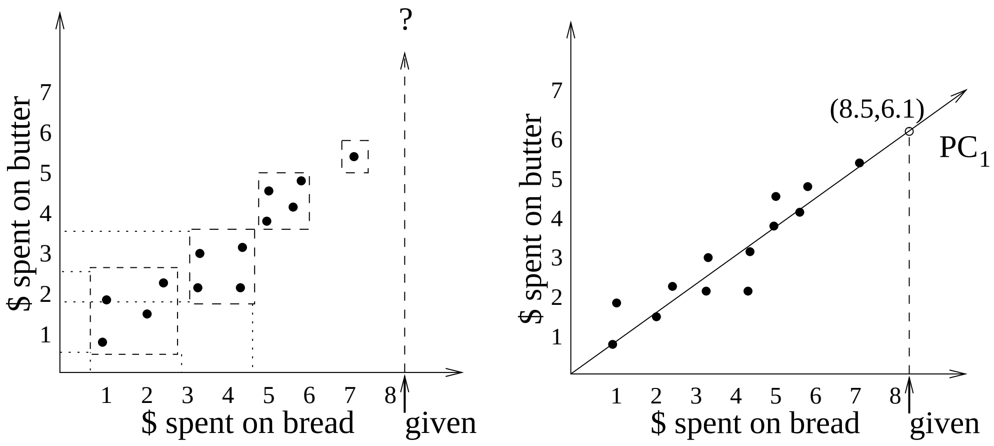
For the linear rules, the advantages are the following:

- They achieve more compact descriptions, if the data points are linearly correlated, as in Figure 10, or as in the real datasets that we saw earlier. In such cases, a single linear rule captures the correlations, while several minimum bounding rectangles are needed by the quantitative association rules to convey the same information;

- They can perform extrapolations and predictions: For example, in Figure 10, suppose that we are given that a customer bought \$8.50 of bread; how much butter is s/he expected to buy? The linear rules will predict \$6.10 on butter, as Figure 10(b) illustrates. The quantitative association rules have no rule that can fire, exactly because the vertical line of "feasible solutions" intersects none of the bounding rectangles. Thus they are unable to make a prediction;

- Their derivation requires a single pass over the dataset;

- They are easily implemented: thanks to highly fine-tuned eigensystem packages, the remaining programming effort is minimal. As an indication, Appendix B lists the code for the Karhunen-Loeve transform (≡PCA), in `mathematica`. Notice that, excluding comments and blank lines, it spans 9 lines! The code is available at available at `ftp://olympos.cs.umd.edu/pub/SRC/kl.m`.



(a) Quantitative (b) Linear Rules

Figure 10: Illustration of Rules on a fictitious dataset of sales on bread and butter: (a) quantitative association rules; (b) linear rules. The "given" entry asks for an estimation for butter, for the given amount spent on bread

7 Conclusions

We have proposed a completely different type of rules as the target of data mining efforts, namely, *linear rules*. These rules have significant advantages over Boolean and quantitative association rules:

- They lead to a natural measure, the "guessing error", which can quantify how good a given set of rules is;

- They can estimate one or more missing/hidden/corrupted values, when a new data record is given; thus, they can also be used in forecasting, for 'what-if' scenarios, and for detecting outliers;
- They are based on the time-tested tool of Principal Component Analysis (PCA), which is the optimal way to perform dimensionality reduction [13];
- They are easy to implement: the most difficult part of our method is the solution of an eigensystem for which reliable packages and/or source code are widely available;
- They are fast and scalable, requiring a *single pass* over the data matrix, and growing linearly on the largest dimension of the matrix, presumably the number N of rows (customers);
- They give visualization for free, thanks to the dimensionality reduction properties of PCA.

We discussed how to interpret linear rules and what their qualitative differences are from the Association Rules. Finally, we presented experiments on several real datasets, which showed that the proposed linear rules can achieve up to 5 times smaller guessing error than its competitor.

Future research could focus on further applications of linear rules and PCA for data mining applications, such as for categorical data, outlier detection, and so on.

Acknowledgments

We would like to thank Björn Thór Jónsson and Kostas Stathatos for their help in interpreting the PCs for the "**nba**" dataset. We would also like to thank Rakesh Agrawal for offering us his synthetic dataset generator, and Mike Franklin for providing an RS/6000 to install and run the generator.

A Linear Algebra Definitions and Proofs

Here we give the formal definition of eigenvalues and eigenvectors:

Definition 2 *For a square $n \times n$ matrix* $\mathbf{S}$*, a unit vector* $\mathbf{u}$ *and a scalar λ that satisfy*

$$\mathbf{S} \times \mathbf{u} = \lambda \times \mathbf{u} \tag{4}$$

are called an <u>eigenvector</u> and its associated <u>eigenvalue</u>, respectively, of the matrix $\mathbf{S}$.

Next, we give the algorithm to determine hidden and unknown values. We start with some preliminary definitions.

Definition 3 *The <u>complement</u> of a set $\mathcal{A}$ is denoted $\mathcal{A}^C$.*

Definition 4 *An h-hole row vector $\mathbf{b}_{\mathcal{H}}$ is defined as a vector with holes (denoted with "?"s) at indices given in $\mathcal{H}$.*

An example of a 5×1 2-hole row vector is the following:

$$\mathbf{b}_{\{2,4\}} = [b_1, ?, b_3, ?, b_5] \tag{5}$$

Definition 5 *An $(M - h) \times M$ <u>elimination matrix</u> $\mathbf{E}_{\mathcal{H}}$ is defined as an $M \times M$ identity matrix with $h = |\mathcal{H}|$ rows removed, where the row indices are given in the set $\mathcal{H}$.*

An example of a 3×5 elimination matrix is the following:

$$\mathbf{E}_{\{2,4\}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

An elimination matrix is very useful in helping us pick and choose entries from vectors. For example, we can eliminate the "?"s from $\mathbf{b}_{\{2,4\}}$ as follows:

$$\mathbf{E}_{\{2,4\}} \times \mathbf{b}^t_{\{2,4\}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} b_1 \\ ? \\ b_3 \\ ? \\ b_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_3 \\ b_5 \end{bmatrix} \tag{7}$$

Following is pseudocode for filling in the holes, which requires the use of several auxiliary vectors: $\mathbf{b}', \mathbf{x}_{concept}, \mathbf{d}$. For brevity, we omit the physical intuition behind them.

```
/* input:  b_H, a 1 × M row vector with holes */
/* output:  b̂, a M × 1 row vector with holes filled */
```

1. $\mathbf{b}' \leftarrow \mathbf{E}_{\mathcal{H}} \times \mathbf{b}^t_{\mathcal{H}}$;
2. $\mathbf{V}' \leftarrow \mathbf{E}_{\mathcal{H}} \times \mathbf{V}$;
3. solve $\mathbf{V}' \times \mathbf{x}_{concept} = \mathbf{b}'$ for $\mathbf{x}_{concept}$
4. $\mathbf{d} \leftarrow \mathbf{V} \times \mathbf{x}_{concept}$;
5. $\hat{\mathbf{b}} \leftarrow \mathbf{b} \times [\mathbf{E}_{\mathcal{H}^c}]^t + \mathbf{d} \times [\mathbf{E}_{\mathcal{H}}]^t$;

In step 3, the equation $\mathbf{V}' \times \mathbf{x}_{concept} = \mathbf{b}'$, where there are $(M-h)$ equations and k unknowns, can either be exactly-specified, over-specified, or under-specified:

CASE 1: (EXACTLY-SPECIFIED) This occurs when $(k+h) = M$.
Here the respective linear equations have an exact solution. In this case,

$$\mathbf{x}_{concept} = (\mathbf{V}')^{-1} \times \mathbf{b}' \tag{8}$$

CASE 2: (OVER-SPECIFIED) This occurs when $(k+h) < M$.
Here the system has no solution, so we find a least-squares solution for $\mathbf{x}_{concept}$ based on the Moore-Penrose pseudo-inverse of $\mathbf{V}'$. This uses the singular value decomposition (see [16]) of $\mathbf{V}'$:

$$\mathbf{V}' = \mathbf{R} \times diag(\mu_j) \times \mathbf{S}^t \tag{9}$$

Since $\mathbf{V}'$ is singular, no inverse exists, but we can find a pseudo-inverse:

$$[\mathbf{V}']^{-1} = \mathbf{S} \times diag(1/\mu_j) \times \mathbf{R}^t \tag{10}$$

and, thus,

$$\mathbf{x}_{concept} = [\mathbf{V}']^{-1} \times \mathbf{b}' \tag{11}$$

CASE 3: (UNDER-SPECIFIED) This occurs when $(k+h) > M$.
Here there are an infinite number of solutions. We propose to reduce the solution by 'throwing away' $(k+h) - M$ PCs to make the system exact, and then solve using CASE 1.

B Karhunen-Loeve Code

```
(* given a matrix mat_ with $n$ vectors of $m$ attributes,
   it creates a matrix with $n$ vectors and their
   first $k$ most 'important' attributes
   (ie., the K-L expansions of these $n$ vectors)
 *)
KLexpansion[ mat_, k_:2] := mat . Transpose[ KL[mat, k] ];

(* given a matrix with $n$ vectors of $m$ dimensions,
   computes the first $k$ singular vectors,
   ie., the axes of the first $k$ Karhunen-Loeve expansion
   *)
KL[ mat_ , k_:2 ]:= Module[
```

```
    {n,m, avgvec, newmat,i,
    val, vec },

    {n,m} = Dimensions[mat];
    avgvec = Apply[ Plus, mat] / n //N;

    (* translate vectors, so the mean is zero *)
    newmat = Table[ mat[[i]] - avgvec , {i,1,n} ];

    {val, vec} = Eigensystem[ Transpose[newmat] . newmat ];

    vec[[ Range[1,k] ]]
]
```

References

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Database Mining: A Performance Perspective". *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining Association Rules between Sets of Items in Large Databases". In *Proc. of the 1993 ACM SIGMOD Conference*, pages 207–216, Washington D.C., USA, May 1993.

[3] Rakesh Agrawal and John C. Shafer. "Parallel Mining of Association Rules". *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, December 1996.

[4] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules". In *Proc. of the* 20^{th} *VLDB Conference*, pages 487–499, Santiago, Chile, September 1994.

[5] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. "A Linear Method for Deviation Detection in Large Databases". In *Proc. of* 2^{nd} *Int'l Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, USA, August 1996.

[6] Sergey Brin, Rajeev Motwani, and Craig Silverstein. "Beyond Market Baskets: Generalizing Association Rules to Correlations". In *Proc. of the 1997 ACM SIGMOD Conference (to appear)*, Tucson, Arizona, USA, May 1997.

[7] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. "Data Mining: An Overview from a Database Perspective". *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, December 1996.

[8] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[9] Usama Fayyad and Ramasamy Uthurusamy. Data mining and knowledge discovery in databases. *Communications of the ACM: Data Mining and Knowledge Discovery (special issue)*, 39(11), November 1996.

[10] Eui-Hong Han, George Karypis, and Vipin Kumar. "Scalable Parallel Data Mining for Association Rules". In *Proc. of the 1997 ACM SIGMOD Conference (to appear)*, Tucson, Arizona, USA, May 1997.

[11] Jiawei Han and Yongjian Fu. "Discovery of Multiple-Level Association Rules from Large Databases". In *Proc. of the* 21^{st} *VLDB Conference*, pages 420–431, Zurich, Switzerland, September 1995.

[12] Wen-Chi Hou. "Extraction and Applications of Statistical Relationships in Relational Databases.". *IEEE Transactions on Knowledge and Data Engineering*, 8(6):939–945, December 1996.

[13] I.T. Jolliffe. *Principal Component Analysis.* Springer Verlag, 1986.

[14] Andreas Mueller. "Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison". Technical Report CS-TR-3515, Department of Computer Science, University of Maryland at College Park, August 1995.

[15] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. "An Effective Hash Based Algorithm for Mining Association Rules". In *Proc. of the 1995 ACM SIGMOD Conference*, pages 175–186, San Jose, California, USA, May 1995.

[16] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992. 2nd Edition.

[17] John Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Mateo, CA, 1993.

[18] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[19] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. "An Efficient Algorithm for Mining Association Rules in Large Databases". In *Proc. of the* 21^{st} *VLDB Conference*, pages 432–444, Zurich, Switzerland, September 1995.

[20] Abraham Silberschatz and Alexander Tuzhilin. "What Makes Patterns Interesting in Knowledge Discovery". *IEEE Transactions on Knowledge and Data Engineering*, 8(6):970–974, December 1996.

[21] Ramakrishnan Srikant and Rakesh Agrawal. "Mining Generalized Association Rules". In *Proc. of the* 21^{st} *VLDB Conference*, pages 407–419, Zurich, Switzerland, September 1995.

[22] Ramakrishnan Srikant and Rakesh Agrawal. "Mining Quantitative Association Rules in Large Relational Tables". In *Proc. of the 1996 ACM SIGMOD Conference*, pages 1–12, Montreal, Quebec, Canada, June 1996.