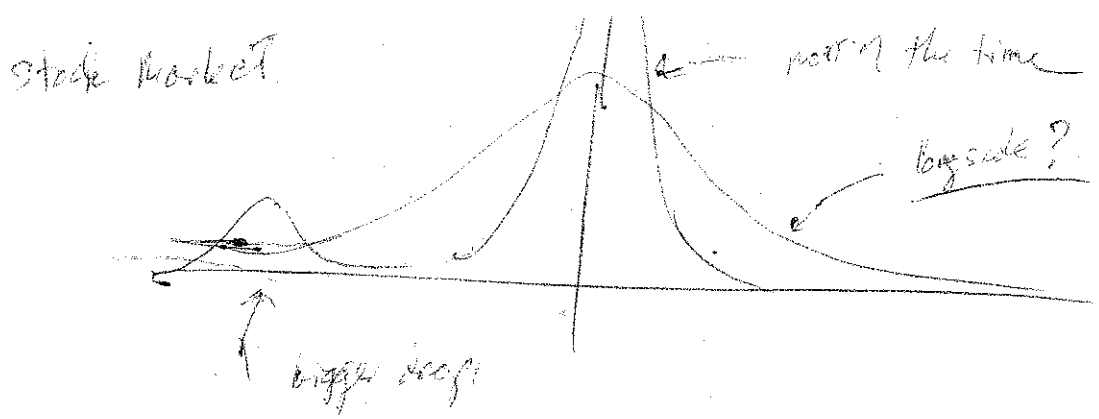
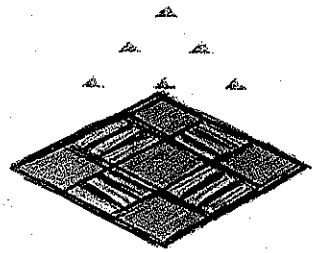


Chance Discovery: The Discovery and Management of Chance Events

HUBEY

Papers from the 2002 AAAI Fall Symposium
Technical Report FS-02-01



opensource.mit.edu
user innovation.mit.edu

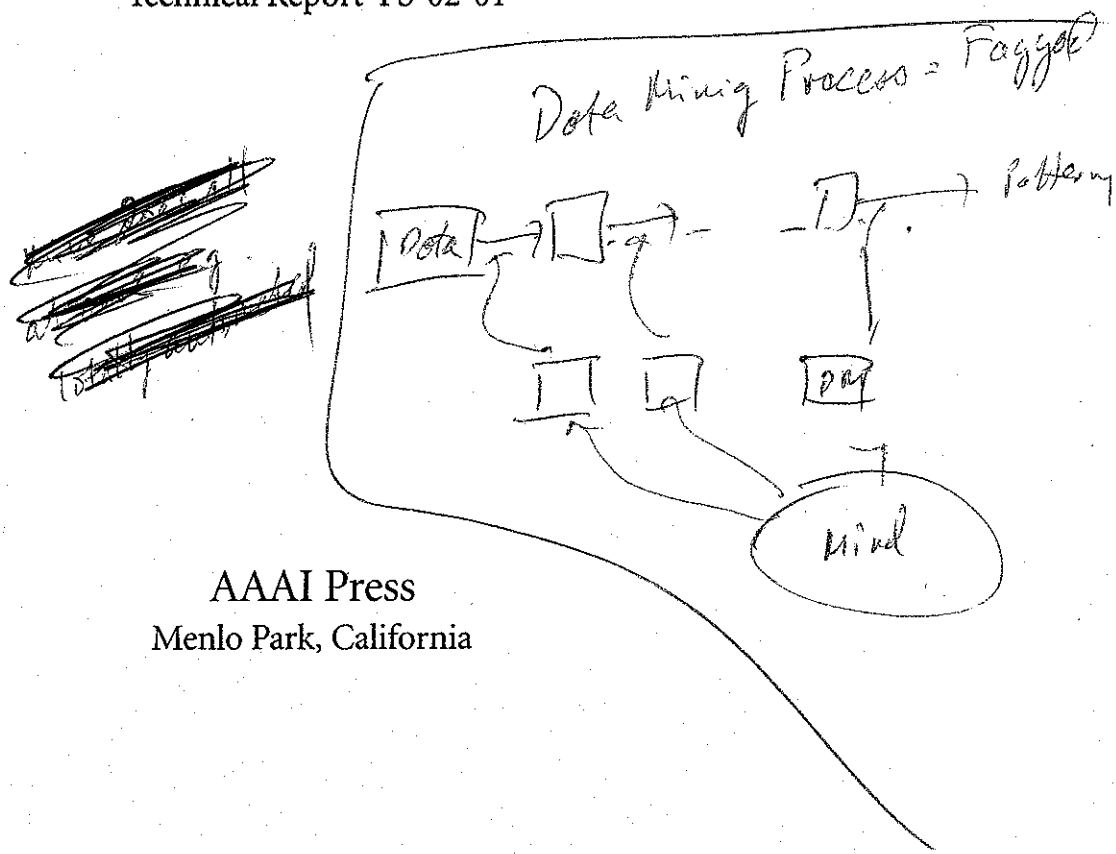
AAAI Press
American Association for Artificial Intelligence

Chance Discovery: The Discovery and Management of Chance Events

Papers from the 2002 AAAI Fall Symposium

Peter McBurney, Yukio Ohsawa, and Simon Parsons, Cochairs

November 15–17, North Falmouth, Massachusetts
Technical Report FS-02-01



AAAI Press
Menlo Park, California

O(1) DETECTION OF CHANCE EVENTS, REAL-TIME PROCESSING OF HIGH-DIMENSIONAL STREAMING DATA, AND DATAMINING

H.M. Hubey, Department of Computer Science
Montclair State University
Upper Montclair, NJ 07043

hubeyh@mail.montclair.edu
http://www.csam.montclair.edu/~hubey

Abstract

A scalable/parallelizable/distributed method is shown that was originally developed for data mining but which is extended to perform web mining, text mining, mining of time-series or temporal data, mining mixed-type data, and mine high-dimensional data. It is implicitly stochastic thus it can incorporate statistical methods such as Bayesian. It is based on the automatic generation of assocation rules. Since it also uses a multiplicative fuzzy logic the neural network tuned to the data is easily comprehensible. It lends itself to visualization, and interactive exploration. Since it uses a unique hashing algorithm (for storage and data manipulation), one that works with associative access, it can be used for other techniques such as nearest-neighbor methods. It is shown here that it is a perfect solution for chance-event detection and processing. It lends itself to hardware acceleration and thus can be used for very large scale projects such as streaming data for "homeland defense". The method is an ideal platform for integration of data warehousing, OLAP and data mining.

1. Introduction

Discovery and management of chance events is intimately related to the real-time or near-real-time detection of significant events (which are probably best characterized as an approximation by using outliers as proxies, mining of massive data sets and particularly mining of streams in real-time). In order to be able to come to some conclusion as to whether these events are significant, one must be able to compute in real time some causal or Bayesian calculations.

There are several key properties of warehouses that make it difficult for standard database algorithms to work correctly. One of the most important is that the warehouse can change size and shape (e.g. the dimensionality). There are several hashing schemes that can expand and shrink; for example, extendible hashing from Fagin et al [1979] that provides fast access for dynamic files; dynamic hashing from Larson [1978] which provides similar capability, and linear hashing from Litwin [1980] which provides yet another method of achieving similar aims.

However today's warehouses are expected to provide much more, for example, real-time decision support and diagnosis systems, chance-event detection, mining of massive data sets and streams for "homeland defense", temporal datamining etc. These require a kind of data repository that can support many different kinds of access and serve many types of processing needs. Some of these require the method to possess these properties among others:

1. The bucketing scheme must be dynamic so that it can grow along with the data being collected. There are hashing schemes that allow files to grow and shrink (for example Fagin[1979], Larson[1978] or Litwin[1980]) but they are not in a form in which any computation can be done easily because the hashing function hashes almost identical primary keys to distant buckets.

2. The repository must be conducive to datamining and pattern recognition. In other words the structure should not hinder associative access but foster it.

3. The repository should be able to handle *out-of-bounds data without creating a fault*. This may require a kind of processing which can be done in real-time e.g. streaming data.

4. It should be possible to constantly fix-up the repository as is done with B-trees or hashed databases instead of having to take down the system to redo the index as in [the old] ISAM.

5. It should be possible for the system to immediately detect outliers since chance-occurrences may depend on outliers.

6. Some kind of inferential reasoning either based on fuzzy-logic or Bayesian analysis should be performable easily and quickly using the rapid and associative access method of the system.

7. The load should be easily factorable into different CPUs (i.e. parallelizable) so that performance increases can be obtained by incremental increases in space resources e.g. time-space tradeoff. This makes the system cost-effective without requiring extremely high clock speeds for CPUs for increased performance

8. Such a system should also scale well for any size data which may require distributed warehousing or distributed databases.

9. For increased speed and efficiency parts of the system should be implementable in hardware. For example hardware based load-distribution for very-high dimensional data is extremely important.

10. Off-the-shelf items should be useable to implement the system instead of requiring special manufacturing facilities. Such is the case with Beowulf in which local area network switches are used.

The existing dynamic hashing algorithms cannot handle the requirements because they suffer from the scheme that requires or implements a system such that nearly synonymous keys that hash to distant buckets instead of either to the same bucket or adjacent buckets.

What we need in creating a repository is a system that is to be able to *put input vectors that are close/near each other in the same or adjacent buckets so that k-neighbors and similar algorithms can quickly fetch required data.* Indeed we need something like associative access so that we can immediately find all the input vectors that are near the target vector. Such a system is HUPSA [2000,2001,2002] (High-dimensional, Unified, Parallelizable, Scalable, Associative-NN-method].

1.1. Description

Without loss of generality let the data being warehoused be the set of n-dimensional and real-valued vectors $Z = \left\{ z_k \middle| z_k \in \Re^n \right\}$ k=1..n. Then each atomic data is of the form $z_k^{(p)}$ where the input pattern, $p \in \{1 \ldots N\}$. We would like to store these where they can be accessed efficiently and quickly. Standard B-trees of various kinds or hashing may be used. However since this warehouse is expected to change size we should use either extendible hashing, linear hashing, or dynamic hashing or a system that has the advantages of these with none of the disadvantages. A special hashing function that will allow for associative access and thus allow quick access to outliers as well as to the data will be described.

In order to use hashing, we first normalize each component of the vector using the algorithm

$$1) \qquad \xi_k = H\left(\frac{Z_k - z_k}{Z_k - \varsigma_k} - \frac{1}{2}\right)$$

where, H(.) is the Heaviside Unit Step function, $Z_k = max_p\left(z_k^{(p)}\right)$ and $\varsigma_k = min_k\left(z_k^{(p)}\right)$. Therefore the first part of the normalization maps all vectors to the interval [0,1], then the Heaviside Step function assigns all those greater than 0.5 the value 1 otherwise 0. *Hence each input vector is (becomes) a bitstring that can be used for hash-based storage.*

Viewed dynamically each vector is placed at one of the nodes of an n-dimensional hypercube. Each node of the hypercube may point to a bucket that holds the data. In addition each node may also hold the actual occurrence of such vectors in the data record hence it provides efficient access for nearest neighbors type of datamining. It is also the idea data structure for a datamining method [Hubey2000], [Hubey2001]. Now, the storage method of HUPSA will be described. It is obvious that two arrays Max[] and Min[] must be stored so that all new data vectors can be hashed to one of the buckets.

2. Dynamic Database Components

The input data is normalized $z_k \in \{0, 1\}^n$ so that each type/kind of input vector is representable as a node of an n-dimensional binary hypercube. The n-cube is also embeddable in a 2-D mesh (a grid) on which we can define a distance [Hubey2000, 2002, 2002]. Furthermore the 2D grid is representable as an extension of the Karnaugh map (KH-map). The map uses the reflection algorithm to generate all sequences of bitstrings so that adjacent cells have distance one. This construction also creates a simple algorithm for a Hamiltonian traversal of the hypercube [Hubey1999]. First the objects and simple algorithms that occur naturally in connection with bitstrings or binary vectors are described directly below.

2.0 Hypercubes

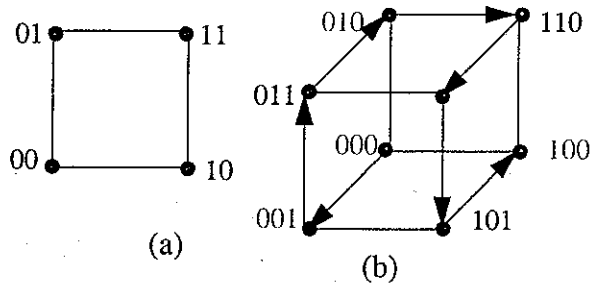


Figure 1: Hypercubes in N-Space. The Hamiltonian walk is superimposed on the hypercube in Fig (1b).

The natural space for bitstrings is the space of hypercubes as well as that of vector spaces. The advantage of the hypercube is that the models can be explained in graph-theoretic terms.

2.1. Reflected Code

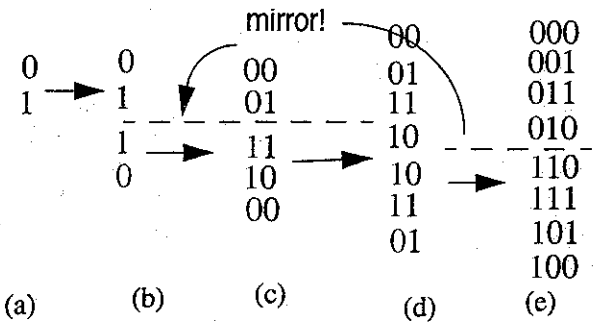


Figure 2: The Reflected Code (Gray Code) (a) write the bits, (b) reflect the bits, (c)prefix 0s to the first half and 1s to the second half, (d) reflect the result again, (e) prefix the first 4 with 0s and the second half with 1s. It should be noted that the last number differs from the first by one bit.

The hypercube graph is intimately connected to the reflected code. And for low-dimensional spaces the hypercube is connected with the Karnaugh map. In higher dimensions we have to use other methods such as that of Quine-McCluskey. The next part of the puzzle is the Karnaugh map, or K-map which can be used to perform Boolean minimization although only for smaller dimensions (e.g. $n \leq 4$).

2.2. K-map

$$F(w, x, y, z) = \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}\bar{y}z + w\bar{x}\bar{y}\bar{z} + w\bar{x}\bar{y}z$$

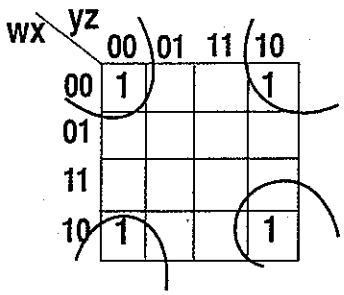


Figure 3: K-map Simplification: A Boolean function such as F(w,x,y,z) can be simplified using the graphical method known as the K-map. It reduces to $F(w, x, y, z) = \bar{x}\bar{z}$. For more detail please consult any book on digital design, for example, Mano[1991], Shiva[1991] or Tomek[1990].

2.3 Data Torus and KH-map

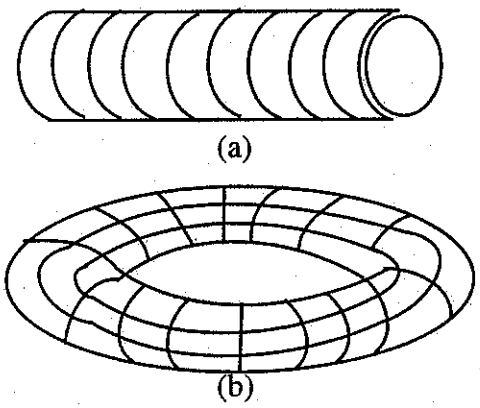


Figure 4: Metric Space for High-Dimensional Data. The KH-map is first wrapped on a cylinder as in (a), and then the cylinder is bent into the shape of a torus so that the corners of the KH-map constitute 'neighbors'. This can be found in Hubey[1994, 1999] in conjunction with mathematical linguistics.

Using the reflected code we can create a larger structure than a K-map which being a visual method is restricted to four variables. It would be possible to create larger K-maps but they would require visualization in high-dimensions. The torus in Fig (4) can have a distance metric associated along the surface and thus seems an ideal space for high-dimensional data [Hubey2000].

2.4. The Bucketing Scheme

In addition to the relationships of the above objects, the normalization/approximation procedure described above is ideally suited for hashing-based associative access for datawarehousing. Indeed hashing based storage is also ideal for relational databases or any other kind of storage. The resulting bitstring may be used similarly to the various hashing and bucketing schemes described in the literature. The great advantage of hash-based storage in this particular scheme in which the data is both normalized and is an approximation of the actual data is that it provides associative access.

This is ideal for classification/estimation methods such as the "K-nearest neighbor" methods which without the scheme shown here do not scale well because the scattered storage requires exhaustive comparisons in order to work. The scheme presented here is associative access hence only those data that are "nearest" will be fetched and used in the comparison of the nearest-neighbor methods. The additional advantage of the scheme is that it is dynamic hashing hence the database can expand and shrink. Indeed, even the dimensionality of the database/warehouse can increase and shrink dynamically.

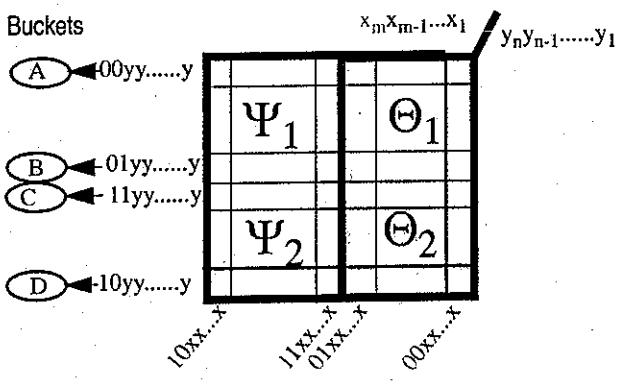


Figure 5: The Naming Scheme of the Variables of the DataWarehouse: The labels show the first bit of each column and row. All the vectors (bitstrings) in a give row hash to the same bucket. For example, bitstrings with the row*column vectors 0yy...y*0xx...x through 0y...y*1x...x map to the bucket A, where * represents concatenation.

In Fig (5) the naming and bucketing scheme is shown. The columns and rows are taken directly from the approximation scheme. The notation 10yy...y indicates that except for the value of the first two (e.g. MSB) bits (which are 10) the others are unknown and we don't need to know their values for the exposition. It does not mean that the bit-values are all the same as seems to be implied by yy...y. In Fig (5) the table represents all the possible combinations of the bitstring $x_m x_{m-1} \cdots x_1 * y_n y_{n-1} \cdots y_1$ where * is the concatenation operator.

Each row in Fig (5) may hash a different bucket. It would be possible for the pointers in adjacent rows to hash to the same bucket to achieve efficient space utilization as done, for example, in extendible hashing [Fagin,1978], dynamic hashing [Larson,1978], and linear hashing [Litwin,1980].

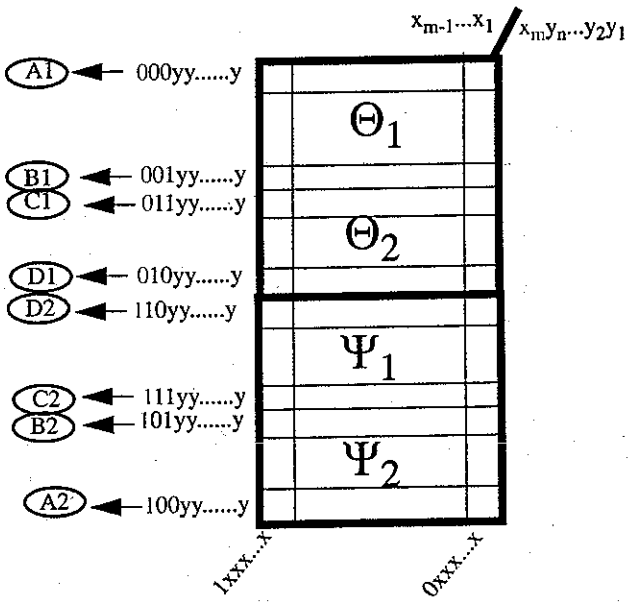


Figure 6 Bucket Splitting All the buckets split at once because the configuration of the system changes. With some more extra work, space utilization can be increased by deferring splitting by using overflow buckets. However, if the data is randomly distributed, if one bucket is ready to split, others are probably full too. This should be compared to Figure (5) which shows the form of the mapping before the splitting of buckets. The string numbering follows the reflected or Gray code.

Bucket merging is basically the inverse/reverse process and is straightforward resembling other bucket-merging algorithms.

2.5 Input Vector Dimension Change in Real-time

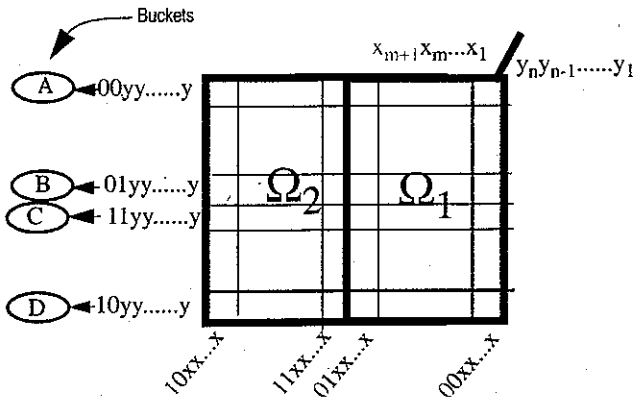


Figure 7 Dynamically Adding a New Variable

It is best to view the addition of the new variable as something akin to a split since the variable gets appended to the string/vector y. Hence new buckets are created for all the data involving the new variable (wlog) x_m. All the previous buckets then automatically become affiliated with the non-occurrence of the new data. However, if there is no need to create new buckets (to save space) or at least to defer splitting, then the new variable may be appended to the end of the horizontal (column vectors) as the variable x_{m+1} as shown above.

Since the dimensionality of the KH-map increases exponentially in the number of variables for large dimensional data sparse-matrix-like methods will have to be employed for high-dimensional data. For large sequences of adjacent bitstrings where there is no data, there is no need to create a map e.g a matrix. Instead a one-dimensional version of the reflected code (see Fig (2)) or as fewbits of the bitstring as necessary can be used. In other words the matrix will use only the bitstring $y_ny_{n-1}...y_1$ large enough so that the maximum number of data points in any row is not larger than the size of the bucket. If some of the adjacent rows can fill up a single bucket, as earlier, they may point to only a single bucket.

3. Datamining

The datawarehousing structure shown above is also ideal for datamining and machine learning. The main components of the scheme consist of;

A) Dual (synergistic) approximation scheme consisting of:

A.1) Normalization/Approximation as shown in Section (1.1)

A.2) Boolean Minimization & Automatic association rule generation

B) A comprehensible multiplicative neural network

The details can be found in Hubey [2000,2001,2002]. A quick outline will be given here. The first part of the approximation is the mapping of the input vectors to $\{0,1\}^n$. The second part is the minimization which produces (in the unsupervised mode) nonlinear clusters. The number of occurrences of each type of event (which is represented as a bitstring) is accumulated in the KH-map, $K[x,y]$. Then a threshold Θ is selected. The second part of the approximation is the mapping

2) $$K[x,y] = H(K[x,y]-\Theta)$$

Therefore Boolean minimization (e.g. a two-level minimization) using an algorithm such as the Quine-McCluskey algorithm creates nonlinear clusters. The result of this stage is two-fold

I) We obtain assocation rules at the level of approximation given by the threshold Θ. Therefore the we obtain a set of Association Rules are of the type $\{R_j(\Theta)\}$. Therefore we really have a set of association rules at each threshold level. These can be combined or used description of the data at multiple scales.

II) Since the result is derived via unsupervised clustering, we can create a neural network whose middle layers are the unsupervised clusters of (I). Therefore we know the number of middle layers that should be used for the neural network since the neural network has been customized for this set of data. Secondly by such clustering (which is distinctly different from PCA which creates new variables as a linear combination of the original input vectors) we also create a nonlinear dimension reduction since the minterms are of the type $x_ix_j...x_k$. This kind of dimensional reduction until the invention of this method could only be used in physics, mainly fluid dynamics[Olson1973, White1979].

III.1) We can generalize the minterms so that they are products of powers e.g. of the type as given by $x_i^{w_i}x_j^{w_j}...x_k^{w_k}$, and then using logarithms we can train the already customized network to fine-tune the weights i.e. taking logs, the products become sums. These products can

be treated as generalized conjunctions using the complementation $C(x) = 1/x$ [Hubey,1999] and [Hubey ,2002].

III.2) For the third layer we can use generalized disjunctions and thereby create highly nonlinear categories or perform function approximation with the resulting network.

III.3) In both (III.1) and (III.2) the use of fuzzy logic guarantees that the resulting network, is "comprehensible" just like the association rules [Craven,1995].

In other words, a customized and tuned two-layer minimization neural network is created specifically for the data and which presumably captures the relationships inherent in the data. This idea can be extended to quickly capture chance-events, and to perform further processing.

4. Real-time Capture of Chance Events

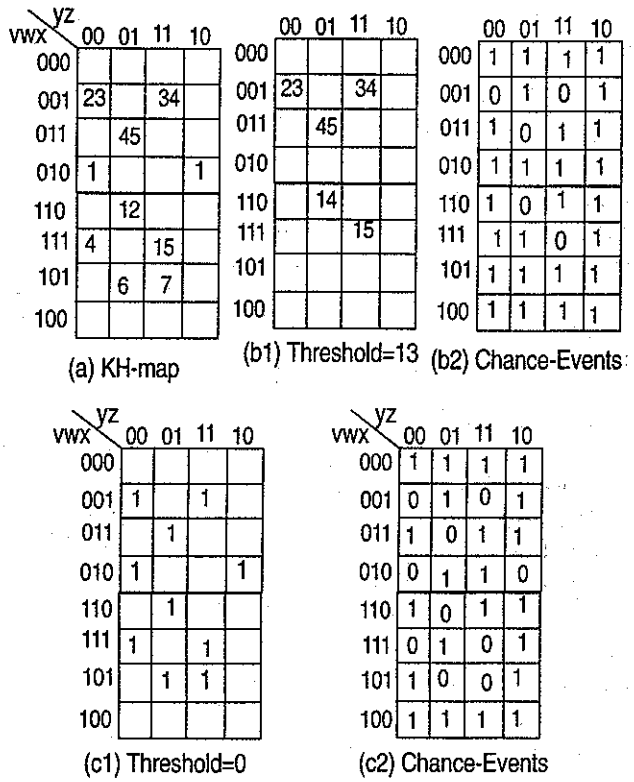


Figure 8: Detection of Chance-Events in Real-Time

The threshold can be set at different levels giving rise to different Boolean minimizations, and thus to different neural networks which with specific kinds of fuzzy logic can be considered to be fuzzy-decoders.

In capturing chance events or outliers we are interested not in the high-frequency occurrence events but exactly the opposite. In function approximation we would call these "outliers". The method discussed above can be tuned so that no special methods are needed to handle outliers [Hubey2002].

However, in this case we would like to focus our attention on the outliers or the low-frequency occurrences. Hence the only we need to do is complement the values in the KH-map. The processing is exactly as explained in section 3 except that everything will hold true for low-frequency or zero-occurrences. Therefore we can achieve real-time detection of such chance events either simply by checking the KH-table or making the calculations in the neural network that is especially tuned to detect such chance-events.

Therefore to modify the datamining system in Hubey[2000] to detect chance-events we have two options which will be illustrated via an example. First, what is a chance-event? If it is an event that has never occurred, then it corresponds to zeros in the KH-map (e.g. Fig (8a)). However if it is a rare-event, then we can handle it the same way as a never-occurred event simply by considering all KH-map values below a threshold to be such events. Practically speaking we would simply select a threshold and set to zero all the cells in the KH-map those entries that are below the threshold. Since we want to train a neural network or have some association rules that determine what constitutes a chance-event, we would then complement all the bits in Fig (8c1) to obtain Fig (8c2). Then using a minimization algorithm such as the Quine-McCluskey algorithm we would simplify the Boolean function shown in Fig (8c2). This two-stage minimized circuit can be treated as a fuzzy-decoder, and is the basis of the datamining algorithm in Hubey[2000, 2001, 2002].

5. Hardware-Assisted Detection, Load-Sharing and Distributed Processing

There are several requirements for a system that can respond to chance-events. We could gain performance increases by using a server farm. Each server can be especially tailored to respond to very specific [types of] chance-events. In this case each server would then be considered an expert system. Or it may be considered an agent. Putting distinct agents or expert systems on different machines, possibly employing heterogeneous operating systems and hardware provides flexibility and thus possibly affords greater performance or flexibility. Extremely high-speeds are necessary for high-load streaming data and this data will have to be sent to the correct agent (machine) for processing.

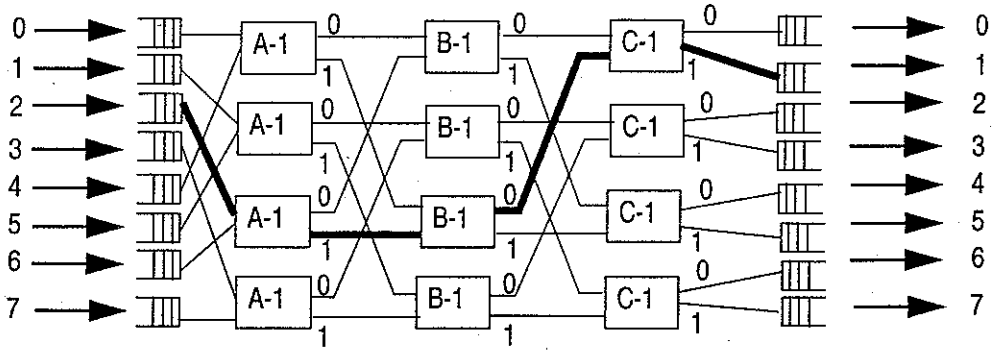


Figure 9: A Simple Banyan Switch: The switch only uses 3-bit per address and hence allows for only 8 input ports and 8 output ports. Input at port 2 being routed to port 1 (001 in binary). Since the high-throughput streaming data is being converted to its bitstring representation (e.g. the hashing function) the same bitstring can be used both to send a message to the correct server that a chance-event has occurred or to a back-end database machine that can store it. The same scheme can be used to create a distributed database for high-performance access. It should be noted that the exact same scheme can be used for search engine farms that have to deal with billions of web pages and search on words (which can be represented as strings e.g. categorical data). These switches can be used to create completely connected computer mesh networks. In the case of a database machine, the actual message (e.g. the data) can follow the routing based on the hash-address bitstring. This is typically how the packet routing is done by these switches in ATM (Asynchronous Transfer Mode). The system has log(n) stages with n/2 microswitches at each stage.

Since this will be a distributed processing network, we would need a special machine with high-speed front and back ends. The machine would accept all incoming data and rapidly normalizing them would send them out to the correct servers (agents).

This part can be automated quite easily using well understood technologies from digital communication. A modified Banyan switch (Batcher-Banyan switch) can easily handle high loads. A Banyan switch is a multistage switch that consists of microswitches at each stage. The message is routed to an output port whose address is given as a part of the message. In this case, the message is the specific type of input data (chance-event) and it is also the address of the machine to handle this specific type of input data. For n inputs and n outputs, each stage uses one bit of the address to route the message. The first stage uses the first bit, the second stage then uses the second bit and so on

6. Conclusion and Summary

It has been shown that the datamining method [Hubey[2000]] can be extended to datawarehousing and thus create a unified environment for both storing and mining data. It has been shown that the same methods used in datamining as in the above work can be used to also detect rare and chance events using comprehensible neural networks. The semantics of chance-events have to be left for another time since knowing what to do without a massive database requires domain knowledge. Research is continuing to expand and extend the method of normalization, associative access and minimization from clustering, datamining, and datawarehousing (and detection of chance events) to broader fields. One method of extension to be considered will be to find the fuzzy equivalence of Bayesian methods. Other areas of research will be in the area of hardware-assisted load-sharing in parallel processing, and hardware-assisted detection of chance-events. The extension of this will be eventually to mining of massive data-streams and "homeland defense".

References

M. Craven and J. Shavlik, []1995] "Extracting comprehensible concept representations from trained neural networks", IJCAI Workshop on Comprehensibility in Machine Learning, (Montreal, Quebec, Canada).

Fagin, R., J. Nievergelt, N. Pippenger, and H.R. Strong,[1979] "Extendible Hashing- a fast access method for dynamic files.", ACM Transactions on Database Systems 4, no. 3 (September 1979): 315-344.

Hubey, H.M. [1994] *Mathematical and Computational Linguistics,* Mir Domu Tvoemu, Moscow, Russia.

Hubey, H.M. [1999] *Mathematical and Computational Linguistics,* Lincom Europa.

Hubey, H.M. [1999] *The Diagonal Infinity*, World Scientific, Singapore.

Hubey, H.M.[2000] "The Curse of Dimensionality", submitted to Journal of Datamining and Knowledge Discovery.

Hubey, H.M., I. Sigura, K. Kaneko, and P. Zhang, [2001] "Dimension Reduction in Datamining", Proceedings of the ICCIT2001 International Conference, October 12-13, 2001, Montclair, NJ.

Hubey, H.M. [2002] "Feature Selection for SVMs via Boolean Minimization" submitted to KDD2002 conference, Alberta, Canada.

Larson, P. [1978] "Dynamic Hashing." BIT 18:184-201.

Larson, P.[1985] "Linear Hashing with Overflow-handling by Linear Probing." ACM Transactions on Database Systems 10, no. 1 (March 1985):75-89.

Litwin, W.[1980] "Linear Hashing: A New Tool for File and Table Addressing." Proceedings of the 6th Conference on Very Large Databases (Montreal, Canada, Oct 1-3, 1980) New York: ACM/IEEE:212-223.

Litwin, W. [1978] "Virtual Hashing: A Dynamically Changing Hashing." Proceedings of the 4th Conference on Very Large Databases (Berlin 1978) New York:ACM/ IEEE: 517-523.

Mano[1991] *Digital Design*, Prentice-Hall, Englewood Cliffs, NJ.

Olson, R [1973] *Essentials of Engineering Fluid Mechanics*, Intext Educational Publishers, NY.

Shiva, S. [1991] *Computer Design & Architecture*, Harper Collins, NY.

Tomek, I. [1990] *Computer Architecture and Organization*, Computer Science Press, NY

White, F. [1979] *Fluid Mechanics*, McGraw-Hill, New York