5-2020

Formal Concept Analysis Techniques Can Help in Intelligent Control, Deep Learning, etc.

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-20-51
To appear in *Proceedings of the 15th International Conference on Concept Lattices and Their Applications CLA'2020*, Tallinn, Estonia, June 29 - July 1, 2020.

Formal Concept Analysis Techniques Can Help in Intelligent Control, Deep Learning, etc.

Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
vladik@utep.edu

Abstract

In this paper, we show that formal concept analysis is a particular case of a more general problem that includes deriving rules for intelligent control, finding appropriate properties for deep learning algorithms, etc. Because of this, we believe that formal concept analysis techniques can be (and need to be) extended to these application areas as well. To show that such an extension is possible, we explain how these techniques can be applied to intelligent control.

1 How Formal Concept Analysis Fits into the General Scheme of Being

General need for compression. The current world is filled with data. At any given moment of time, numerous sensors produce humongous amount of numerical measurement results, images, videos, etc.

This data is usually useful – otherwise it would not have been produced, so it is desirable to store and process this information.

However, the problem is that our ability to produce information way exceeds our ability to store and process it. As a result, we cannot physically store every bit produced by the sensors, we need to *compress* this data.

Compression is closely related to interpolation and extrapolation. Of course, if all the bits were independent, if each was informative – containing information that cannot be extracted from other bits – there would be no way to drastically compress this information without losing a significant portion of it. And this would make producing this immediately deleted information useless.

Good news is that bits are not independent, there is usually a strong correlation between them, correlation that allows us to drastically decrease the number of stored bits without losing much information. For example, a photo can be

easily compressed from several Megabytes to dozens of Kylobytes – several orders of magnitude – and we can still easily recognize all the features of a person on the webpage where this compressed photo is posted.

This correlation enables us also to effectively interpolate and extrapolate, i.e., to adequately reconstruct missing information. For example, based on many readings of temperature, wind speed, and other meteorological characteristics ate several locations and heights, we can reasonable accurately reconstruct the values of these characteristics at other locations and heights.

Functions of one, two, etc. inputs. In many cases, we are interested in characteristics q that depend only on one (possible, multi-dimensional) input x: $q = f(x)$. For example:

- we may be interested in the temperature $q(x)$ at different locations and different moments of time (i.e., at different points x in space-time),
- we may be interested in the income $q(x)$ of different people x at different moments of time, etc.

However, in many other cases, we are interested in characteristics $q(x, y, \ldots)$ that depend on two (or even more) different inputs x, y, ... For example:

- we may be interested in the degree $q(x, y)$ to which a given person x would like or dislike a certain movie y (or a certain book, or a certain research paper if x is a researcher),
- we may be interested in knowing the degree $q(x, y)$ to which, in a given situation x, different controls y will lead to good results, etc.

In such situations, we need to compress, interpolate, and extrapolate the desired dependence $q(x, y, \ldots)$.

How can we compress, interpolate, and extrapolate multi-input dependencies: general description. For simplicity, let us consider the case when the desired quantity depends only on two inputs: $q = q(x, y)$. In this case, in the beginning,

- we have information about x and information about y, and
- we need to perform some processing of this information.

A natural way to speed up data processing is to perform some operations in parallel – just like for us humans, a natural way for a person to perform a task faster is to have several helpers working at the same time on the same task. So, if there are some computational steps where we can process x separately and process y separately, these steps need to be performed in parallel before everything else. Thus, in general, processing such data consists of the following two major stages:

- first, we perform an appropriate processing on x, resulting in some values $a(x)$, and at the same time, we perform an appropriate processing on y, producing $b(y)$;

- after that, we perform some processing on the results $a(x)$ and $b(y)$ of the first stage, producing $F(a(x), b(y))$, where F denotes the algorithm performed at this second stage.

At the end, we approximate the original dependence $q(x, y)$ with the simpler-to-store and simpler-to-process dependence $F(a(x), b(y)) \approx q(x, y)$.

Let us describe possible situations, from the simplest to the most complicated.

Linear case. The simplest case is when $q(x, y)$ can be well approximated as a linear function of the values $a(x) = (a_1(x), \ldots, a_k(x))$, i.e., when

$$q(x, y) \approx b_0(y) + b_1(y) \cdot a_1(x) + \ldots + b_k(y) \cdot a_k(y)$$

for some coefficients $b_i(y)$ depending on y. By adding $a_0(x) = 1$, we can make this formula more uniform

$$q(x, y) = b_0(y) \cdot a_0(x) + b_1(y) \cdot a_1(x) + \ldots + b_k(y) \cdot a_k(y).$$

In matrix notations $q_{xy} \stackrel{\text{def}}{=} q(x, y)$, $a_{xi} \stackrel{\text{def}}{=} a_i(x)$, and $b_{yi} \stackrel{\text{def}}{=} b_i(y)$, this formula takes the form

$$q_{xy} = \sum_{i=0}^{k} a_{xi} \cdot b_{yi}. \tag{1}$$

This is a known idea of *matrix decomposition*, actively used in Principal Component Analysis (see, e.g., [15]), in predicting people's reaction to movies, etc. (see, e.g., [1, 17]).

A natural generalization of linear case, to operations generalizing addition and multiplication. In the above case (1):

- we use multiplication to process individual components a_{xi} and b_{xi} of the results $a(x)$ and $b(y)$ of processing x and y, and
- we use addition to combine these results.

Instead of multiplication and addition, we can use more general combination functions.

For example, we can have expert control rules of the type "if x satisfies the property a_i (e.g., if $x > 0.1$), then the control y should satisfy the property b_i (e.g., $y \in [0, 1]$)". We can then combine these rules into an equivalent formula, according to which y is a reasonable control for the situation x if:

- either the first rule is applicable, i.e., x satisfies the property a_1 and y satisfies the property b_1,
- or the second rule is applicable, i.e., x satisfies the property a_2 and y satisfies the property b_2, etc.

If we:

- denote the truth value of the statement "x satisfies the property a_i" by $a_i(x)$ and
- denote the truth value of the statement "y satisfies the property b_i" by $b_i(y)$,

then the truth value $q(x,y)$ of the statement "y is a reasonable control for x" takes the form

$$q(x,y) = (a_1(x) \,\&\, b_1(y)) \vee (a_2(x) \,\&\, b_2(y)) \vee \ldots$$

This is the usual example of formal concept analysis; see, e.g., [3, 6]

This example can be extended to the case when experts use imprecise ("fuzzy") words from natural language to describe their rules; see, e.g., see, e.g., [2, 8, 10, 13, 14, 16]. In this case, the expert's control rules have a similar form "if x is a_i (e.g., small), then the control y should be b_i (e.g., moderate)". We can similarly translate these rules into an equivalent formula, according to which y is a reasonable control for the situation x if:

- either the first rule is applicable, i.e., x satisfies the property a_1 and y satisfies the property b_1,
- or the second rule is applicable, i.e., x satisfies the property a_2 and y satisfies the property b_2, etc.

We can then ask the expert to estimate, on a scale from 0 to 1, the degrees $a_i(x)$ to which different values x satisfy the imprecise ("fuzzy") property a_i and the degrees $b_i(y)$ to which different values y satisfy the property b_i.

Since it is usually not practically possible to ask the expert to provide estimates for the combined statement "x satisfies the property a_i and y satisfies the property b_i" for all the pairs (x,y) – there are just too many possible pairs – we have to estimate the degrees to which such statements are true based on whatever information is available – namely, the degrees $a_i(x)$ and $b_i(y)$. For this estimation, we can use a general algorithm $f_{\&}(a,b)$ for estimating our degree of confidence in a composite statement $A \,\&\, B$ based on our degrees of confidence a and b in the statement A and B.

This algorithm has to satisfy certain properties: e.g., since $A \,\&\, B$ means the same as $B \,\&\, A$, this operation must be commutative; since $A \,\&\, (B \,\&\, C)$ is equivalent to $(A \,\&\, B) \,\&\, C$, this operation must be associative, etc. Such operations are known as "and"-operations, or, for historical reasons, t-norms.

Similarly, we can use a similarly motivated "or"-operation (also known as t-conorm) $f_{\vee}(a,b)$ to estimate our degree of confidence in $A \vee B$ based on our degrees of confidence a and b in the statements A and B. In these terms, the desired degree of confidence $q(x,y)$ can be described as follows:

$$q(x,y) = f_{\vee}(f_{\&}(a_1(x), b_1(y)), f_{\&}(a_2(x), b_2(y)), \ldots). \tag{2}$$

Case when some transformations are linear, while others are not. So far, we have considered the case when the functions F is linear – or similar to linear, with more general operations instead of addition and multiplication.

In some cases, a linear transformation is followed by a non-linear one. For example, in a traditional 3-layer neural network (see, e.g., [5]), the result q of processing the inputs $x_1, \ldots, x_n$ has the form

$$q = \sum_{k=1}^{K} W_k \cdot s_k \left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{ki} \right) - W_0, \tag{3}$$

for some non-linear functions $s_k(z)$.

In other words, we first compute linear combinations $a_k(x) = \sum\limits_{i=1}^{n} w_{ki} \cdot x_i - w_{ki}$, and then perform a non-linear transformation $q = \sum\limits_{k=1}^{K} s_k(a_k) - W_0$.

General case, when everything is possibly non-linear. In general, we may have non-linear transformations $a(x)$ and $b(y)$, followed by a nonlinear transformation $F(a, b)$.

A typical example of such a representation is deep learning (see, e.g., [7]), where the dimension of the signal decreases as we go from the multi-D input through processing layers, and thus, the original multi-dimensional signal is compressed – and, in general, compressed non-linearly. Interestingly, in many experiments, the intermediate results have intuitive meaning – so that the same intermediate values can be used for other problems y.

2 What Is the Remaining Problem and How Formal Concept Analysis Techniques Can Help

General problem. If we have rules, and these rules are perfect, there is no problem. However, this is rarely the case. In most practical situations, we have some information about $q(x, y)$, and we need to come up with the appropriate decomposition into $a(x)$, $b(y)$, and $F(a, b)$.

- In the linear case of matrix decomposition, we have examples of people's attitude to different movies, and we need to come up with the most adequate values a_{xi} and b_{yi}.
- In the case of formal concept analysis, we have a table (often only partially filled) of truth values $q(x, y)$, and we need to find appropriate predicates $a_i(x)$ and $b_i(y)$.
- In the case of intelligent control, we have degrees $q(x, y)$, and we need to come up with appropriate rules – e.g., with the most appropriate functions $a_i(x)$ and $b_i(y)$.

- In the case of deep learning, while there are spectacular successes – like beating a world champion in Go, there are also spectacular failures, when the system classifies a clearly cat picture as a dog and vice versa. This means that even in this case, the problem of finding the appropriate values $a_i(x)$ and $b_i(y)$ is far from being solved.

How can formal concept analysis technique help. Of course, most of the above problems are NP-hard (see, e.g., [11, 12]), so we cannot expect to find a feasible algorithm that always finds a solution. However, many efficient techniques have been developed in formal concept analysis, and it is desirable to extend them to other cases as well.

Such an extension is clearly possible – e.g., the paper [4] provided an efficient greedy algorithm for deriving fuzzy values when $f_{\&}(a,b) = \max(a+b-1,0)$ and $f_{\vee}(a,b) = \min(a+b,1)$, and the paper [9] showed that the same algorithm can work for other "and"- and "or"-operations as well. The unpublished result from [9] is briefly described in the Appendix.

Conclusion. Our conclusion is simple and straightforward: let us thing big, let us extend what we have to other cases.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

The author is greatly thankful to Radim Belohlavek, Marketa Krmelova, and Martin Trnecka for their help and encouragement.

References

[1] G. Acosta, M. Hernandez, N. Villanueva-Rosales, E. Smith, and V. Kreinovich, "Why matrix factorization works well in recommender systems: a systems-based explanation", *Journal of Uncertain Systems*, 2019, Vol. 13, No. 3, pp. 164–167.

[2] R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.

[3] R. Belohlavek and V. Vychodil, "Discovery of optimal factors in binary data via a novel method of matrix decomposition", *Journal of Computer and System Sciences*, 2010, Vol. 76, No. 1, pp. 3–20.

[4] R. Belohlavek and V. Vychodil, "Factor analysis of incidence data via novel decomposition of matrices", *Proceedings of the 7th International Conference on Formal Concept Analysis ICFCA'2009, Darmsdart, Germany, May*

21–24, 2009, Springer Lecture Notes in Artifical Intelligence, Vol. 5548, pp. 83–97.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[6] B. Ganter and R. Wille, *Formal Concept Analysis. Mathematical Foundations*, Springer, Berlin, 1999.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Leaning*, MIT Press, Cambridge, Massachusetts, 2016.

[8] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[9] M. Krmelova, R. Belohlavek, and V. Kreinovich, *Fuzzy Formal Concept Analysis Can Help Extract Rules from Experts*, unpublished paper, 2013.

[10] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.

[11] D. S. Nau, *Specificity Covering*, Duke University, Department of Computer Science, Technical Report CS-1976-7, 1976.

[12] D. S. Nau, G. Markowsky, M. A. Woodbury, and D. B. Amos, "A mathematical analysis of human leukocyte antigen serology", *Mathematical Biosciences*, 1978, Vol. 40, pp. 243–270.

[13] H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.

[14] V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.

[15] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman and Hall/CRC, Boca Raton, Florida, 2011.

[16] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.

[17] C. Zhao, S. Sun, L. Han, Q. Peng, "Hydrid matrix factorization for recommender systems in social networks", *International Journal on Neural and Mass-Parallel Computing and Information Systems*, 2016, Vol. 26, No. 6, pp. 559–569.

A How Formal Concept Analysis Can Help Extract Intelligent Control Rules

We have a finite set of pairs (x, y) for which we know $q(x, y)$. Let us denote this set by P. Based on this information, how can we find appropriate functions $a_i(x)$ and $b_i(y)$?

In this appendix, we show that in this general intelligent control case, we can use a greedy algorithm that was originally proposed in [4] for a specific case of "and"- and "or"-operations.

By definition of a greedy algorithm:

- we start by finding the functions $a_1(x)$ and $b_1(y)$,
- then we fix the selected functions $a_1(x)$ and $b_1(y)$ and find the functions $a_2(x)$ and $b_2(y)$, etc., and,
- in general, we fix the already selected functions $a_1(x)$, $b_1(y)$, ..., $a_{k-1}(x)$, $b_{k-1}(y)$, and select a pair of functions $a_k(x)$ and $b_k(x)$.

How do we select these functions $a_k(x)$ and $b_k(y)$?

From the equation (2) and from the fact that $p \le f_\vee(p, q)$ for all p and q, we can conclude that for each k, we should have

$$f_\vee(f_\&(a_1(x), b_1(y)), \ldots, f_\&(a_{k-1}(x), b_{k-1}(y)), f_\&(a_k(x), b_k(y))) \le q(x, y),$$

i.e., equivalently, that

$$f_\vee(q_{k-1}(x, y), f_\&(a_k(x), b_k(y))) \le q(x, y), \tag{4}$$

where we denoted

$$q_{k-1}(x, y) \stackrel{\text{def}}{=} f_\vee(f_\&(a_1(x), b_1(y)), \ldots, f_\&(a_{k-1}(x), b_{k-1}(y)))$$

for $k - 1 \ge 1$ and $q_0(x, y) \stackrel{\text{def}}{=} 0$.

A natural idea is to select the functions $a_k(x)$ and $b_k(y)$ that would cover as many pairs (x, y) as possible, i.e., for which the value

$$N_k(a_k, b_k) \stackrel{\text{def}}{=} \#\{(x, y) \in P : f_\vee(q_{k-1}(x, y), f_\&(a_k(x), b_k(y))) = q(x, y)\}$$

is the largest possible, where $\#S$ denoted the number of elements in the set S.

From this viewpoint, once we selected $b_k(y)$, it is reasonable to select a function $a_k(x)$ which leads to the largest possible coverage, i.e., to select

$$(b_k)^{\downarrow_k}(x) \stackrel{\text{def}}{=} \sup\{a : \forall y \in Y_x\, (f_\vee(q_{k-1}(x, y), f_\&(a, b_k(y))) \le q(x, y))\},$$

where $Y_x \stackrel{\text{def}}{=} \{x : (x, y) \in P\}$. Similarly, if we have selected the function $a_k(x)$, then it is reasonable to select a function $b_k(y)$ which leads to the largest possible coverage, i.e., to select

$$(a_k)^{\uparrow_k}(y) \stackrel{\text{def}}{=} \sup\{b : \forall x \in X_y\, (f_\vee(q_{k-1}(x, y), f_\&(a_k(x), b)) \le q(x, y))\},$$

where $X_y \stackrel{\text{def}}{=} \{y : (x,y) \in P\}$.

Comment. The notations similar to the usual notations from the formal concept analysis are motivated by the fact that in the usual 2-valued logic:

- there are only two truth values 0 and 1;
- when $s_0 \le s$, then $s_0 \vee t \le s$ if and only if $t \le s$; and
- $a \,\&\, b \le s$ if and only if $a \le (b \to s)$.

By using these properties, one can check that in the 2-valued logic, the above formulas can be represented in the following simplified equivalent form (not depending on $q_{k-1}(x,y)$):

$$(a_k)^{\uparrow_k}(y) = \inf_{x \in X_y} (a_k(x) \to a(x,y));$$

$$(b_k)^{\downarrow_k}(x) = \inf_{y \in Y_x} (b_k(y) \to S(x,y)),$$

which are exactly the usual notions $(a_k)^{\uparrow}$ and $(b_k)^{\downarrow}$ in formal concept analysis.

Let us now describe an iterative procedure for finding $b_k(y)$ and $a_k(x)$. In the beginning, the only information that we know about $b_k(y)$ and $a_k(x)$ is that $b_k(y) \ge 0$ and $a_k(x) \ge 0$. Thus, as the starting approximations to the desired functions $b_k(y)$ and $a_k(x)$, we take $b_k^{(0)}(y) = 0$ and $a_k^{(0)}(x) = 0$. For these functions, the quality $N_k\left(a_k^{(0)}, b_k^{(0)}\right)$ is simply equal to the previous value N_{k-1}.

Let us now start improving this selection step by step. In general, let us assume that we have already found approximations $b_k^{(i-1)}(y)$ and $a_k^{(i-1)}(x)$, for which the approximation quality is equal to $N_k\left(a_k^{(i-1)}, b_k^{(i-1)}\right)$.

If some pairs (x,y) are still not covered by this selection, we should try to increase one of the functions $b_k^{(i-1)}(y)$ and $a_k^{(i-1)}(x)$. Let us start with $b_k^{(i-1)}(y)$. The simplest idea is to increase the value $b_k^{(i-1)}(y)$ for one of the value y_0 to the largest possible value

$$b_{y_0}(y_0) \stackrel{\text{def}}{=} \sup\{b : \forall x \in X_{y_0}\ (f_\vee(q_{k-1}(x,y_0), f_\&(a_k^{(i-1)}(x), b) \le q(x,y_0))\},$$

while keeping all other values $b_k^{(i-1)}(y)$ unchanged: $b_{y_0}(y) = b_k^{(i-1)}(y)$ for all $y \ne y_0$.

For each y_0, we form the resulting function $b_{y_0}(y)$, and take $a_{k,y_0} = (b_{y_0})^{\downarrow_k}$ and $b_{k,y_0} = (a_{k,y_0})^{\uparrow_k} = (b_{y_0})^{\downarrow_k\uparrow_k}$. For each y_0, we find the value $N_k(a_{k,y_0}, b_{k,y_0})$ of the objective function, and select $y_{\max}$ for which this value is the largest:

$$N_k(a_{k,y_{\max}}, b_{k,y_{\max}}) = \max_{y_0} N_k(a_{k,y_0}, b_{k,y_0}).$$

The corresponding functions $b_{k,y_{\max}}(y)$ and $a_{k,y_{\max}}(x)$ are then taken as the next iteration $b_k^{(i)}(y)$ and $a_k^{(i)}(x)$: $b_k^{(i)} = b_{k,y_{\max}}(y)$ and $a_k^{(i)} = a_{k,y_{\max}}(y)$. Iterations continue while the value N_k continues to grow, i.e., while

$$N_k\left(a_k^{(i)}, b_k^{(i)}\right) > N_k\left(a_k^{(i-1)}, b_k^{(i-1)}\right).$$

Once it stops growing, i.e., once we have

$$N_k\left(a_k^{(i)}, b_k^{(i)}\right) = N_k\left(a_k^{(i-1)}, b_k^{(i-1)}\right),$$

the iterations stop, and the corresponding functions $b_k(y) \stackrel{\text{def}}{=} b_k^{(i)}(y)$ and $a_k(x) \stackrel{\text{def}}{=} a_k^{(i)}(x)$ are added to the list of pairs

$$(a_1(x), b_1(y)), \ldots, (a_{k-1}(x), b_{k-1}(y)).$$

If after this addition, some pairs $(x, y) \in P$ are still not covered, we similarly find and add the next pair of functions $a_{k+1}(x)$ and $b_{k+1}(y)$, etc., until all the pairs $(x, y) \in P$ are covered.

Our preliminary experiments show that this algorithm leads to reasonable rules.

Comment. A similar greedy algorithm can be used when, instead of the above-described methodology, we use a more logical way to convey the expert's if-then rules, i.e., when we take

$$q(x, y) = f_{\&}(f_{\to}(a_1(x), b_1(y)), f_{\to}(a_2(x), b_2(y)), \ldots),$$

where $f_{\to}(a, b)$ is an implication operation, i.e., an estimate for degree to which the statement $A \to B$ is true given the degrees of confidence a and b in the statements A and B.