Erschienen in: International Journal of Computational Intelligence Systems ; 6 (2013), sup1. - S. 34-45
https://dx.doi.org/10.1080/18756891.2013.818186

Fuzzy Logic in KNIME
– Modules for Approximate Reasoning –

Michael R. Berthold [1] , Bernd Wiswedel [2], and Thomas R. Gabriel [2]

[1] Department of Computer and Information Science, University of Konstanz,
Universitätsstr. 10, 78484 Konstanz, Germany
E-mail: Michael.Berthold@Uni-Konstanz.DE

[2] KNIME.com AG,
Technoparkstrasse 1,
8005 Zurich, Switzerland
E-mail: Bernd.Wiswedel@KNIME.com, Thomas.Gabriel@KNIME.com

Abstract

In this paper we describe the open source data analytics platform KNIME, focusing particularly on extensions and modules supporting fuzzy sets and fuzzy learning algorithms such as fuzzy clustering algorithms, rule induction methods, and interactive clustering tools. In addition we outline a number of experimental extensions, which are not yet part of the open source release and present two illustrative examples from real world applications to demonstrate the power of the KNIME extensions.

Keywords: KNIME, Fuzzy C-Means, Fuzzy Rules, Neighborgrams.

1. Introduction

KNIME is a modular, open[a] platform for data integration, processing, analysis, and exploration [2]. The visual representation of the analysis steps enables the entire knowledge discovery process to be intuitively modeled and documented in a user-friendly and comprehensive fashion.

KNIME is increasingly used by researchers in various areas of data mining and machine learning to give a larger audience access to their algorithms. Due to the modular nature of KNIME, it is straightforward to add other data types such as sequences, molecules, documents, or images. However, the KNIME desktop release offers standard types for fuzzy intervals and numbers, enabling the implementation of fuzzy learning algorithms as well.

A previous paper[2] has already described KNIME's architecture and internals. A follow-up publication focused on improvements in version 2.0[3]. However, for readers not yet familiar with KNIME, we provide a short overview of KNIME's key concepts in the following section before we describe the integration of fuzzy concepts and learning algorithms in the remainder of this paper. To the best of our knowledge none of the other popular open source data analysis or workflow environments[14,9,17] include fuzzy types and learning algorithms. Many specialized open source fuzzy toolboxes exist but most are either purely in academic use or can not be used stand alone. Commercial tools, such as Matlab, often also offer fuzzy extensions. In this paper

[a]KNIME is downloadable from http://www.knime.org

we focus on a complete, integrative platform which is available open source.

We will first describe KNIME itself and provide some details concerning the underlying workflow engine. Afterwards we discuss the fuzzy extensions, in particular the underlying fuzzy types before discussing the integrated algorithms. Before showing two real world applications of those modules we briefly describe ongoing work.

2. KNIME

KNIME is used to build workflows. These workflows consist of *nodes* that process data; the data are transported via connections between the nodes. A *workflow* usually starts with nodes that read in data from some data sources, which are usually text files or databases that can be queried by special nodes. Imported data is stored in an internal table-based format consisting of columns with a certain data type (integer, string, image, molecule, etc.) and an arbitrary number of rows conforming to the column specifications. These data tables are sent along the connections to other nodes, which first pre-process the data, e.g. handle missing values, filter columns or rows, partition the table into training and test data, etc. and then for the most part build predictive models with machine learning algorithms like decision trees, Naive Bayes classifiers or support vector machines. For inspecting the results of an analysis workflow several view nodes are available, which display the data or the trained models in various ways. Fig. 1 shows a small workflow and its nodes.

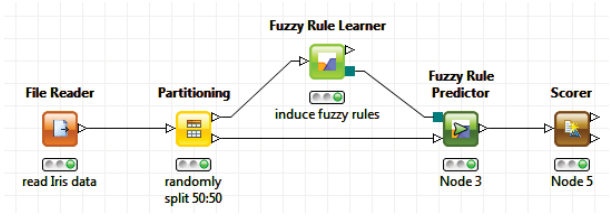

Fig. 1. A small KNIME workflow, which builds and evaluates a fuzzy rule set on the Iris data.

In contrast to pipelining tools such as Taverna[b], KNIME nodes first process the entire input table before the results are forwarded to successor nodes. The advantages are that each node stores its results permanently and thus workflow execution can easily be stopped at any node and resumed later on. Intermediate results can be inspected at any time and new nodes can be inserted and may use already created data without preceding nodes having to be re-executed. The data tables are stored together with the workflow structure and the nodes' settings. The small disadvantage of this concept is that preliminary results are not available quite as soon as if real pipelining were used (i.e. sending along and processing single rows as soon as they are created).

One of KNIME's key features is *hiliting*. In its simple form, it allows the user to select and visually mark ("hilite") several rows in a data table. These same rows are also hilited in all the views that show the same data (or at least the hilited rows). This type of hiliting is simply accomplished by using the 1:1 correspondence among the tables' unique row keys. However, there are several nodes that completely change the input table's structure and yet there is still some relation between input and output rows. A nice example is the *MoSS* node that searches for frequent fragments in a set of molecules. The node's input are the molecules, the output the discovered frequent fragments. Each of the fragments occurs in several molecules. Hiliting some of the fragments in the output table causes all molecules that contain these fragments to be hilited in the input table. Fig. 2 demonstrates this situation in a small workflow where a confusion matrix is linked back to the original data.

One of the important design decisions was to ensure easy extensibility, so that other users can add functionality, usually in the form of new nodes (and sometimes also data types). This has already been done by several commercial vendors (Tripos, Schrödinger, Chemical Computing Group, ...) but also by other university groups and open source programmers. The usage of Eclipse as the core platform means that contributing nodes in the form of plugins is a very simple procedure. The official KNIME website offers several extension plugins for re-

[b] `http://www.taverna.org.uk`

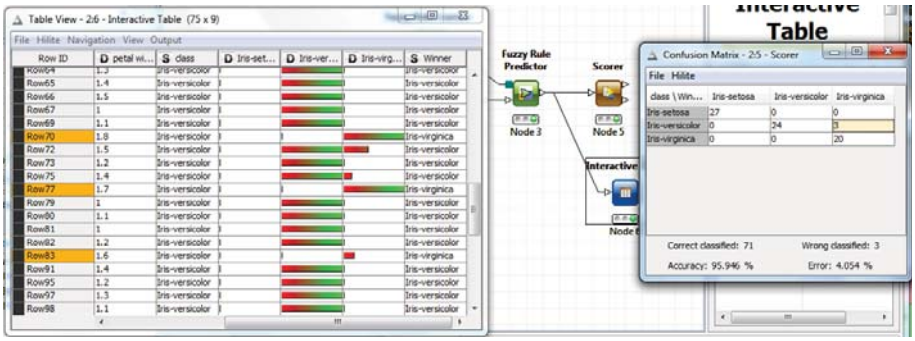

Figure 2: KNIME's hiliting features demonstrated by the linkage between the confusion matrix and the evaluation data.

porting via BIRT[c], statistical analysis with R[d] or extended machine learning capabilities from Weka[e], for example.

Since the initial release in mid 2006 the growing user base has voiced a number of suggestions and requests for improving KNIME's usability and functionality. From the beginning KNIME has supported open standards for exchanging data and models. Early on, support for the Predictive Model Markup Language (PMML) [15] was added and most of the KNIME mining modules natively support PMML, including association analysis, clustering, regressions, neural network, and tree models. With the latest KNIME release, PMML support was enhanced to cover PMML 4.1. See [18] for more details.

Before dicussing how fuzzy types and learning methods can be integrated into KNIME, let us first discuss the KNIME architecture in more detail.

3. KNIME Architecture

The KNIME architecture was designed with three main principles in mind.

- Visual, interactive framework: Data flows should be combined by a simple drag and drop operation from a variety of processing units. Customized applications can be modeled through individual data pipelines.
- Modularity: Processing units and data containers should not depend on each other in order to enable easy distribution of computation and allow for independent development of different algorithms. Data types are encapsulated, that is, no types are predefined, new types can easily be added bringing along type specific renderers and comparators. New types can be declared compatible to existing types.
- Easy expandability: It should be easy to add new processing nodes or views and distribute them through a simple plugin mechanism without the need for complicated install/deinstall procedures.

In order to achieve this, a data analysis process consists of a pipeline of nodes, connected by edges that transport either data or models. Each node processes the arriving data and/or model(s) and produces results on its outputs when requested. Fig. 3 schematically illustrates this process.

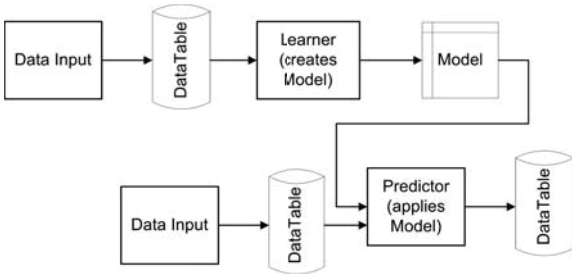

Fig. 3. A schematic for the flow of data and models in a KNIME workflow.

[c] http://www.actuate.com
[d] http://www.r-project.org
[e] http://www.cs.waikato.ac.nz/ml/weka/

The type of processing ranges from basic data operations from filtering or merging to simple statistical functions ranging from computations of mean, standard deviation or linear regression coefficients to the computation of intensive data modeling operators (clustering, decision trees, neural networks, to name just a few). In addition, most of the modeling nodes allow for an interactive exploration of their results through accompanying views. In the following we will briefly describe the underlying schemata of data, node, workflow management and how the interactive views communicate.

3.1. Data Structures

All data flowing between nodes is wrapped within a class called `DataTable`, which holds meta-information concerning the type of its columns in addition to the actual data. The data can be accessed by iterating over instances of `DataRow`. Each row contains a unique identifier (or primary key) and a specific number of `DataCell` objects, which hold the actual data. The reason to avoid access by Row ID or index is scalability, that is, the desire to be able to process large amounts of data and therefore not be forced to keep all of the rows in memory for fast random access. KNIME employs a powerful caching strategy, which moves parts of a data table to the hard drive if it becomes too large. Fig. 4 shows a UML diagram of the main underlying data structure.

3.2. Nodes

Nodes in KNIME are the most general processing units and usually resemble one node in the visual workflow representation. The class `Node` wraps all functionality and makes use of user-defined implementations of a `NodeModel`, possibly a `NodeDialog`, and one or more `NodeView` instances if appropriate. Neither dialog nor view must be implemented if no user settings or views are needed. This schema follows the well-known Model-View-Controller design pattern.

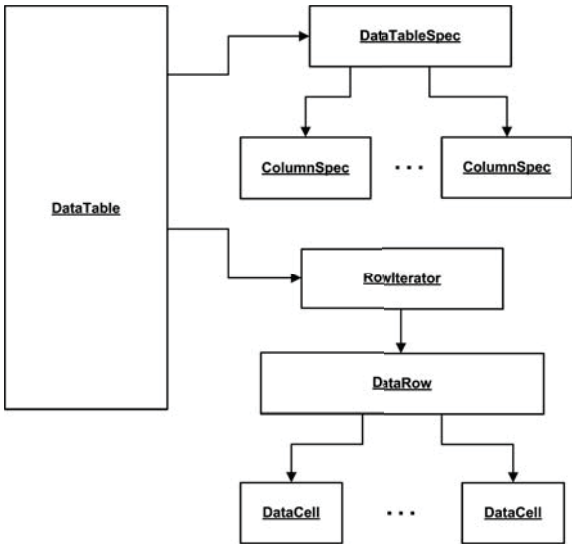

Fig. 4. A UML diagram of the data structure and the main classes it relies on.

In addition, each node has a number of `Inport` and `Outport` instances for the input and output connections, which can either transport data or models. Fig. 5 shows a UML diagram of this structure.

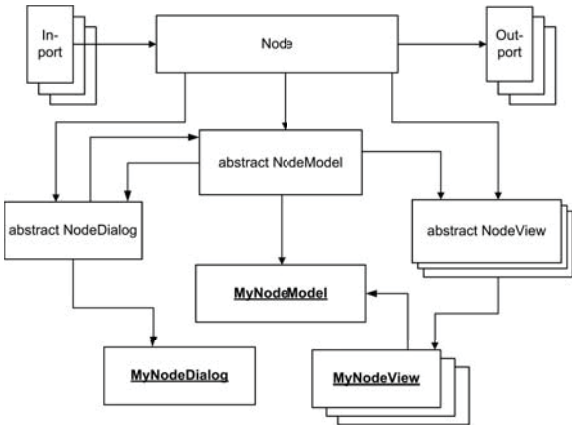

Fig. 5. A UML diagram of the Node and the main classes it relies on.

3.3. Workflow Management

Workflows in KNIME are essentially graphs connecting nodes, or more formally, a directed acyclic graph (DAG). The `WorkflowManager` allows new nodes to be inserted and directed edges (connections) between two nodes to be added. It also keeps track of the status of nodes (configured, executed,

...) and returns, on demand, a pool of executable nodes. This way the surrounding framework can freely distribute the workload among a couple of parallel threads or – optionally – even a distributed cluster of servers. Thanks to the underlying graph structure, the workflow manager is able to determine all nodes required to be executed along the paths leading to the node the user actually wants to execute.

3.4. Views and Interactive Brushing

Each Node can have an arbitrary number of views associated with it. Through receiving events from a `HiLiteHandler` (and sending events to it) it is possible to mark selected points in such a view to enable visual brushing described earlier. Views can range from simple table views to more complex views on the underlying data (e. g. scatterplots, parallel coordinates) or the generated model (e. g. decision trees, rules).

3.5. (Fuzzy) Types in KNIME

KNIME features a modular and extensible type concept. As described earlier, tables in KNIME contain meta information about the types contained in each column. Fig. 6 shows this setup in more detail.

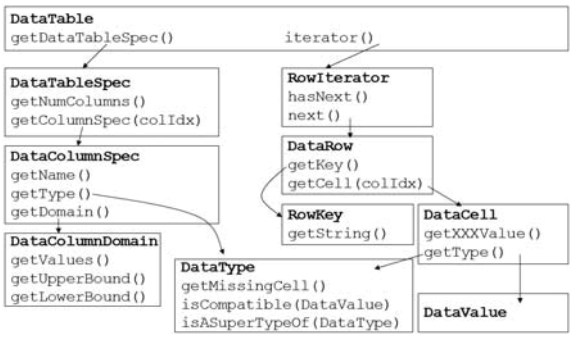

Fig. 6. A schematic showing how data tables can be accessed in KNIME.

This meta information essentially enumerates all possible types (subclasses of `DataValue`) that all cells in that column implement. Particular cell implementations (extending `DataCell`) can implement one or more of these values, `IntCell`, for instance, implements both `IntValue` as well as `DoubleValue`, as an integer can be represented as a double without loosing any information. The reverse is obviously not true, so `DoubleCell` only implements `DoubleValue`. Fig. 7 shows this setup in more detail.

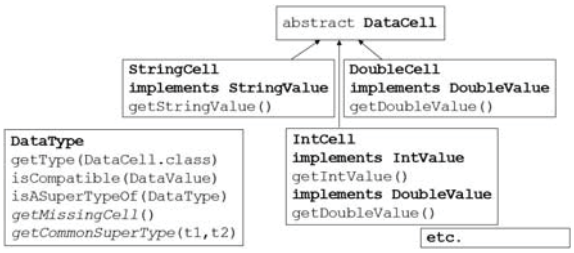

Fig. 7. A schematic showing how data types are organized in KNIME.

Inspecting the KNIME source code reveals, however, that `DoubleCell` does implement additional extensions of `DataValue` namely `FuzzyNumberValue` and `FuzzyIntervalValue` (there are also a few other interfaces implemented such as `ComplexNumberValue` which we will not focus on here). Any double can obviously also represent a singleton fuzzy number[16] or an extreme fuzzy interval with singleton core and support (or a complex number with $0i$) so these extensions allow normal doubles to be treated as fuzzy numbers resp. intervals as well. However, of more interest are obviously the real implementations `FuzzyIntervalCell` and `FuzzyNumberCell` which in this case represent trapezoidal resp. triangular membership functions over a real-valued domain[20].

Fig. 8 shows how this is represented in KNIME for a small fuzzy rule set learned on the Iris data[11]. The meta information about the table on the right is displayed at the top. When a table contains fuzzy intervals/numbers the headers of these columns represent the most common supertype (`FuzzyIntervalCell` in this case) and also some additional properties. An upper and lower bound can be given for some types (as is the case for the first four columns), while the nominal values are listed for others (as can be seen in the fifth column).

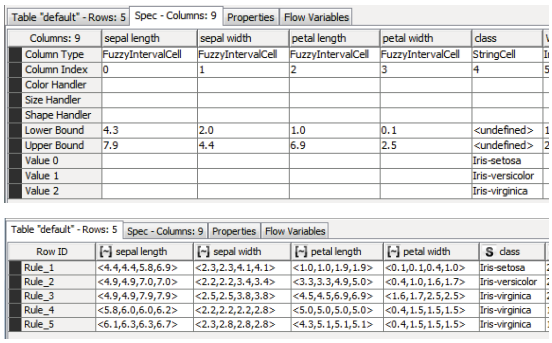

Table "default" - Rows: 5	Spec - Columns: 9	Properties	Flow Variables			
Columns: 9	sepal length	sepal width	petal length	petal width	class	
Column Type	FuzzyIntervalCell	FuzzyIntervalCell	FuzzyIntervalCell	FuzzyIntervalCell	StringCell	
Column Index	0	1	2	3	4	
Color Handler						
Size Handler						
Shape Handler						
Lower Bound	4.3	2.0	1.0	0.1	<undefined>	
Upper Bound	7.9	4.4	6.9	2.5	<undefined>	
Value 0					Iris-setosa	
Value 1					Iris-versicolor	
Value 2					Iris-virginica	

Table "default" - Rows: 5	Spec - Columns: 9	Properties	Flow Variables			
Row ID	[~] sepal length	[~] sepal width	[~] petal length	[~] petal width	S class	
Rule_1	<4.4,4.4,5.8,6.9>	<2.3,2.3,4.1,4.1>	<1.0,1.0,1.9,1.9>	<0.1,0.1,0.4,1.0>	Iris-setosa	
Rule_2	<4.9,4.9,7.0,7.0>	<2.2,2.2,3.4,3.4>	<3.3,3.3,4.9,5.0>	<0.4,1.0,1.6,1.7>	Iris-versicolor	
Rule_3	<4.9,4.9,7.9,7.9>	<2.5,2.5,3.8,3.8>	<4.5,4.5,6.9,6.9>	<1.6,1.7,2.5,2.5>	Iris-virginica	
Rule_4	<5.8,6.0,6.0,6.2>	<2.2,2.2,2.2,2.8>	<5.0,5.0,5.0,5.0>	<0.4,1.5,1.5,1.5>	Iris-virginica	
Rule_5	<6.1,6.3,6.3,6.7>	<2.3,2.8,2.8,2.8>	<4.3,5.1,5.1,5.1>	<0.4,1.5,1.5,1.5>	Iris-virginica	

Fig. 8. An example of fuzzy intervals in KNIME. The underlying meta data is at the top, while the data as it is shown in a table can be seen underneath.

In the following we describe how a number of prominent fuzzy learning methods can be easily embedded into this general framework.

4. Fuzzy C-Means

The well-known fuzzy c-means algorithm[7] is contained in one learning module/node of KNIME. The configuration dialog of the node is shown in Fig. 9, it exposes the usual parameters of the standard implementation in addition to the setup of a noise cluster[10].

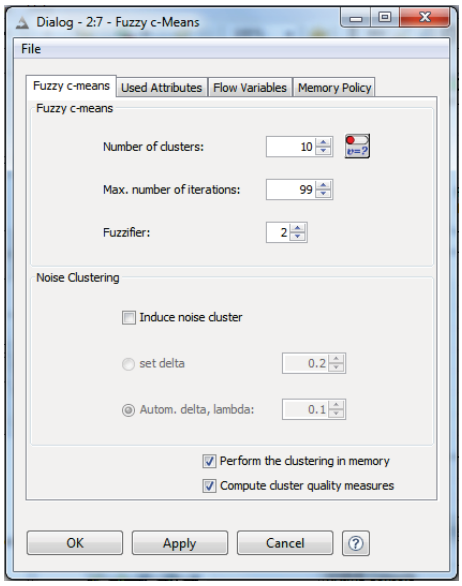

Fig. 9. The dialog of the fuzzy c-means clustering node, displaying the available options.

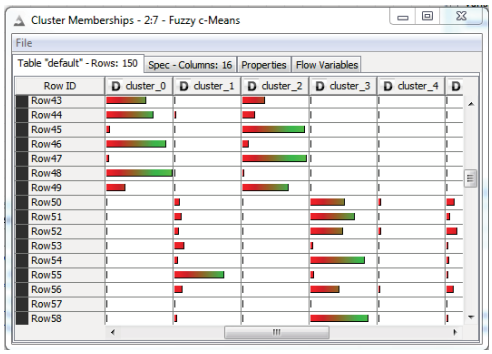

Fig. 10. The output of the fuzzy c-means clustering node, here using the bar renderer to display the degrees of membership.

Note the small button next to the "Number of clusters" field. This indicates that this setting can be easily controlled by a workflow variable[f]. It enables workflows to be set up that loop over different numbers of clusters, running e.g. a cross-validation run and collecting the results over all cluster settings.

Fig. 10 shows the output of the clustering node for the well known Iris data set, where the degree of membership is displayed for each pattern. Various rendering options are available for each column, here a bar char was chosen.

5. Fuzzy Rule Induction

For fuzzy rule induction the FRL algorithm [1,12] was used as a basis. The algorithm constructs fuzzy classification rules and can use nominal as well as numerical attributes. For the latter, it automatically extracts fuzzy intervals for selected attributes. One of the convenient features of this algorithm is that it only uses a subset of the available attributes for each rule, resulting in so-called *free fuzzy rules*. The KNIME implementation follows the published algorithm closely, allowing various algorithmic options to be set as well as different fuzzy norms. After execution, the output is a model description in a KNIME internal format and a table holding the rules as fuzzy interval constraints on each attribute plus some additional statistics (number of covered patterns, spread, volume etc.). These KNIME repre-

[f]Actually all parameters of a node can be controlled by workflow variables but this button makes it easier for typical variables, which are often controlled from the outside.

sentations can be used to further process the rule set but also for display purposes.

Fig. 11 shows an MDS projection of the 4-dimensional rules on to two dimensions. The color indicates the class of each rule, the size the number of covered patterns. More details can be found in [5].

6. Visual Fuzzy Clustering

Another interesting aspect of KNIME is its visualization capabilities. As mentioned above, views in KNIME support cross-view selection mechanisms (called *hiliting*) but views can also be more interactive. One such example is the set of nodes for visual

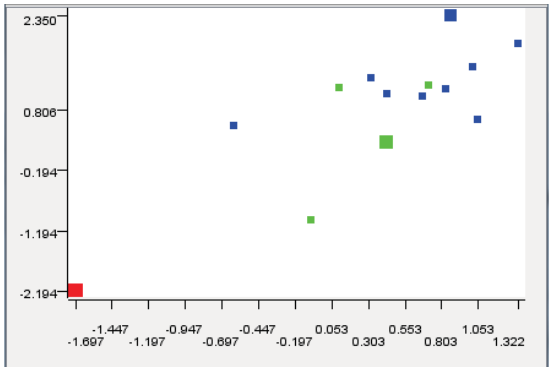

Fig. 11. The fuzzy rules induced from the Iris data projected on to a two dimensional space. Size represents coverage, color the class of the rule.

fuzzy clustering. The nodes can actually perform such clustering in multiple descriptor spaces (*parallel universes*) in parallel [6].

For the purpose of this paper, however, a side aspect of this work is more interesting. The methods described in [6] allow fuzzy clusters to be identified and revised interactively in these parallel universes. Fig. 12 shows a screenshot of the interactive view of this KNIME node again for the Iris data. Each row shows a so-called *Neighborgram* for the data points of interest (usually a user specified class). A single neighborgram represents an object's neighborhood, which is defined by a similarity measure. It contains a fixed number of nearest neighbors to the centroid object, whereby good, i.e. discriminative, neighborgrams will have objects of the centroid's class in the close vicinity.

Fig. 12. The view of the visual fuzzy clustering node. Clusters are presented and fine tuned iteratively by the user.

KNIME also contains nodes for automatic clustering using the data structures. The neighborgrams are then constructed for all objects of interest, e.g. belonging to the minority class, in all available universes. The learning algorithm derives cluster candidates from each neighborgram and ranks these based on their qualities (e.g. coverage or another quality measure). The model construction is carried out in a sequential covering-like manner, i.e. starting with all neighborgrams and their cluster candidates, taking the numerically best one, adding it as a cluster and proceeding with the remaining neighborgrams while ignoring the already covered objects. This simple procedure already produces a set of clusters, which potentially originate from diverse universes. Extensions to this algorithm reward clusters, which group in different universes simultaneously, and thus respect overlaps. Another interesting usage scenario of the neighborgram data structure is the possibility to display them and thus involve the user in the learning process. Especially the ability to visually compare the different neighborhoods of the same object has proven to be useful in molecular applications and for the categorization of 3D objects.

7. Ongoing Work

Current development also includes a number of prototypes for other fuzzy-based analysis and/or visualization methods. It is worth mentioning two more

visualization-based methods.

Performing multi-dimensional scaling (or most other projection methods from a higher dimensional space on to two dimensions) usually loses information pertaining to the uncertainty of the underlying fuzzy points / fuzzy sets. This can be seen in Fig. 11 above. It is not possible to infer from the picture whether the fuzzy sets overlap or how close their core/support regions are. An approach presented in[13] addresses this limitation by also showing estimates for the spread towards neighboring points in the projection. Fig. 13 shows an example for this type of visualization.

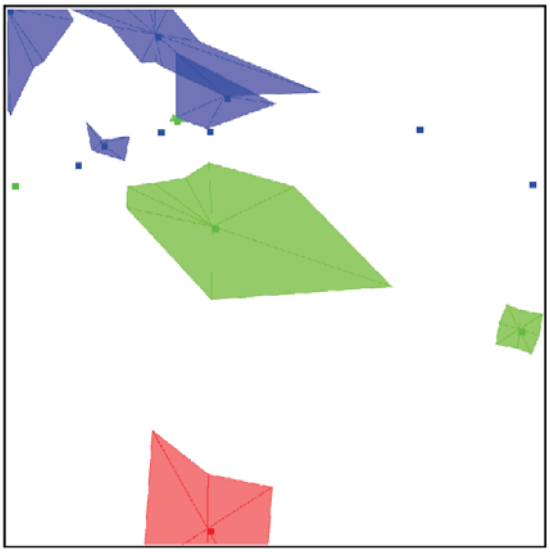

Fig. 13. A prototypical view on projected fuzzy points also displaying estimates for overlap/vicinity of neighboring points.

Another way of visualizing points in medium high dimensional spaces are parallel coordinates. In [4] an extension for this type of visualization was presented, which extends the mechanism to also show fuzzy points or rules. Fig. 14 shows two of the rules learned for the Iris data set.

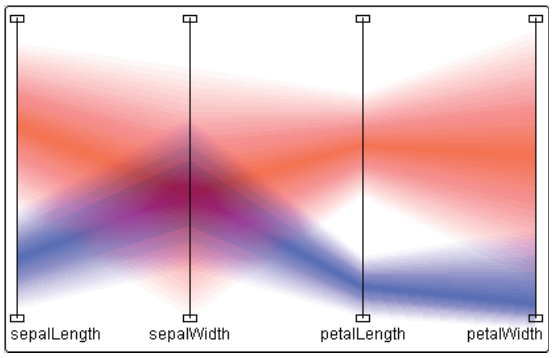

Fig. 14. A visualization of fuzzy rules in parallel coordinates.

8. Other Extensions

In addition to native, built-in nodes, KNIME also allows existing tools to be wrapped easily. An external tool node allows command line tools to be launched, whereas integrations for Matlab, R, and other data analysis or visualization tools allow existing fuzzy learning methods such as ANFIS to be integrated as well.

However, a number of existing wrappers around libraries such as LibSVM or Christian Borgelt's Association Rule and Itemset Mining library demonstrates that it is also feasible to integrate existing tool sets more tightly.

9. Applications

The fuzzy extensions for KNIME discussed here are not only of academic interest but enable users to use these tools easily in practice. In the following we will show two examples. The first one demonstrates the usefulness of the visual fuzzy clustering approach for the exploration of high throughput screening data and the second one focuses on a more complex molecular space modeling task around fuzzy c-means and how the resulting fuzzy partitioning of the space can be visually explored using the KNIME network processing modules.

9.1. Screening Data Analysis

An interesting example for the use of the visual fuzzy clustering methods presented above was reported in [19]. The Neighborgram-based clustering method was applied to a well-known data set from the National Cancer Institute, the DTP AIDS Antiviral Screen data set. The screen utilized a biological assay to measure protection of human CEM cells from HIV-1 infection. All compounds in the data set were tested for their protection of the CEM cell; those that did not provide at least 50% protection were labeled as confirmed inactive (CI). All others were tested in a second screening. Compounds that provided protection in this screening, too, were labeled as confirmed active (CA), the remaining ones as moderately active (CM). Those screening results and chemical structural data on compounds that are not protected by a confidentiality agreement can be accessed online[g]. 41,316 compounds are available, of which we have used 36,452. A total of 325 belong to class CA, 877 are of class CM and the remaining 34,843 are of class CI. Note the class distribution for this data set is very unbalanced. There are about 100 times as many inactive compounds (CI) as there are active ones (CA), which is very common for this type of screening data analysis: although it is a relatively large data set, it has an unbalanced class distribution with the main focus on a minority class, the active compounds. The focus of analysis is on identifying internal structures in the set of active compounds that appeared to protect CEM cells from the HIV-1 infection.

In order to generate Neighborgrams for this dataset, a distance measure needs to be defined. We initially computed Fingerprint descriptors[8], which represent each compound through a 990-dimensional bit string. Each bit represents a (hashed) specific chemical substructure of interest. The used distance metric was a Tanimoto distance, which computes the number of bits that are different between two vectors normalized over the number of bits that are turned on in the union of the two vectors. This type of distance function is often used in cases like this, where the used bit vectors are only sparsely occupied with 1s.

Experiments with this (and other similar) data sets demonstrate well how interactive clustering in combination with Neighborgrams helps to inject domain knowledge in the clustering process and how Neighborgrams help to inspect promising cluster candidates quickly and visually. Fig. 15 shows one example of a cluster discovered during the exploration, grouping together parts of the chemical family of Azido Pyrimidines, probably one of the best-known classes of active compounds for HIV.

This application demonstrates perfectly how fuzzy clustering techniques are critical for real world applications. Attempting to partition this type of screening data into crisp clusters would be absolutely futile due to the underlying fairly noisy data. Instead, by suggesting clusters to the user and having him/her fine tune the (fuzzy) boundaries and then continuing the interactive clustering procedure allows the user to inject background knowledge into the clustering process on the fly.

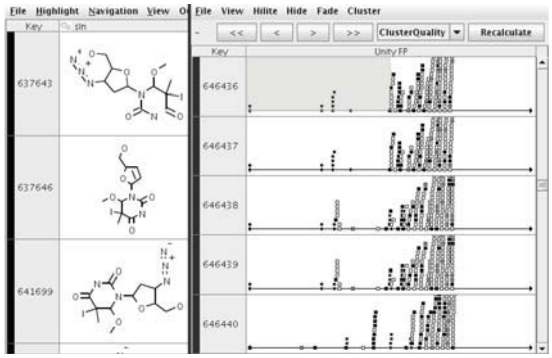

Fig. 15. A fuzzy cluster of the NIH-Aids data centered around compound #646436. This cluster nicely covers part of one of the most well-known classes of active compounds: Azido Pyrimidines.

9.2. Molecular Space Modeling

Trying to get a first impression of a large molecular database is often a challenge because the contained compounds do often not belong to one group alone but share properties with more than one chemical group – obviously this naturally lends itself to

[g] http://dtp.nci.nih.gov/docs/aids/aidsdata.html

a modeling of the space using fuzzy techniques. The workflow depicted in Fig. 17 generates such an overview using a number of more complex sub workflows:

- on the left, the information is read from two files, one containing the structures, the other one containing additional information about each compound;
- the next metanode contains a subworkflow creating additional chemical descriptors;
- the internals of the next metanode are displayed in Fig. 18, it determines optimal settings for the parameters of the fuzzy C-means algorithm: the number of clusters and the fuzzifier;
- the fuzzy C-means node than uses these settings for the final clustering of the entire dataset;
- the last three metanodes create an overview of the clusters, sample the data down so that structures can be displayed meaningfully later, and create the actual network which is then displayed by the final node.

The sub workflow shown in Fig. 18 is particularly interesting here because it illustrates the use of the KNIME looping concept. The loop start node on the left takes its input from a table with several settings for the number of clusters and the fuzzifier of the fuzzy-C-means applied to one portion of the overall data. The cluster assigner on the other partition of our data is then evaluated for its quality (essentially measuring clustering quality indices based on within and across cluster distances). The loop end node collects the summary information of each run and the sorter then picks the best iteration and returns it as variables to be fed into the fuzzy C-means in the overall workflow. Setups similar to this can be used to do model selection also across different model classes, the KNIME example server holds a couple of examples for this as well.

The metanode on the bottom (”Create Cluster Overview”) extracts the most common substructure of all molecules belonging to a cluster, the resulting table is shown in Fig. 16. For a chemist, those representations quickly reveal the main composition of the underlying database. However, such crisp reductions do not reveal more insights.

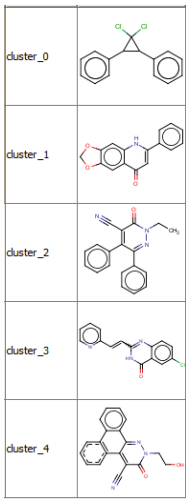

Fig. 16. The most common substructures of the five clusters.

We do not go into much detail about the interna of the subsequent metanodes as they mainly focus on building a network of cluster centers and molecules (as nodes) and introduce edges between those weighted by the corresponding degree of membership. One resulting view is shown in Fig. 19. One can quickly see the main constituents of the clusters, illustrated by a couple of representative molecular structures with a high degree of membership only to that one cluster (we filtered out the majority of the compounds strongly belonging to one class for sake of readability). Compounds that are more ambiguous are positioned inbetween two – or in some cases also three – clusters. These are also molecular structures that can be assigned to a specific chemical group less clearly. The true power of such visualizations lies in their interactivity, of course, just like in many of the other examples. The KNIME network visualization extension allows to highlight points as well and zoom in to focus on details of the network.

10. Conclusions

We have described fuzzy extensions in KNIME and illustrated how classic fuzzy learning methods are easily integrated. We also illustrated how some of the techniques described here can be used in real world application such as the visual clustering of

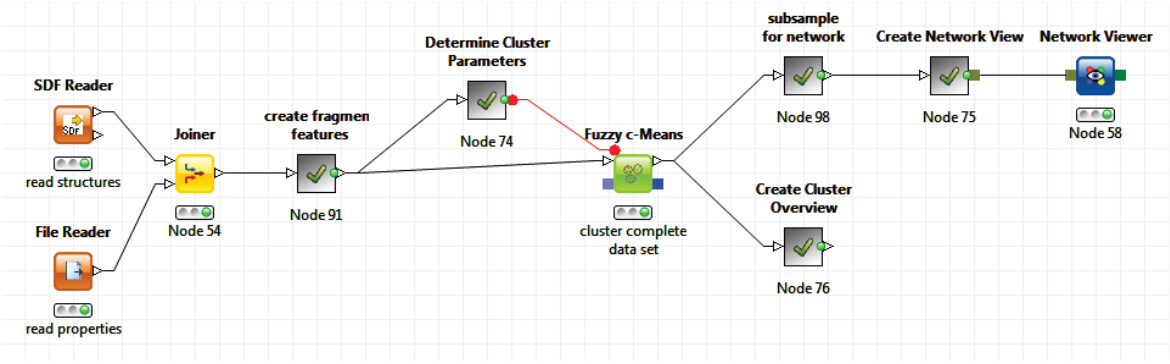

Figure 17: A workflow for the model and visualization of a molecular space.

high throughput screening data and the modeling of molecular spaces.

KNIME offers a solid basis for more fuzzy learning and visualization methods and we look forward to collaborating with the fuzzy community to extend this area of tool coverage in KNIME further.

Acknowledgments

We thank the other members of the KNIME Team and the very active KNIME Community!

References

1. M.R. Berthold. "Mixed Fuzzy Rule Formation," In *International Journal of Approximate Reasoning (IJAR)*, **32**, 67–84, Elsevier, 2003.
2. M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. "KNIME: The Konstanz Information Miner," In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007.
3. M.R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel. "KNIME: The Konstanz Information Miner. Version 2.0 and Beyond," In *SIGKDD Explorations*. ACM Press, **11**(1), 2009.
4. M.R. Berthold and L.O. Hall. "Visualizing Fuzzy Points in Parallel Coordinates," In *IEEE Transactions on Fuzzy Systems*, **11**(3), 369–374, 2003.
5. M.R. Berthold and R. Holve. "Visualizing High Dimensional Fuzzy Rules," In *Proceedings of NAFIPS*, 64–68, IEEE Press, 2000.
6. M.R. Berthold, B. Wiswedel, and D.E. Patterson. "Interactive Exploration of Fuzzy Clusters Using Neighborgrams," In *Fuzzy Sets and Systems*, **149**(1), 21–37, Elsevier, 2005.
7. J.C. Bezdek. "Pattern Recognition with Fuzzy Objective Function Algorithms," Plenum Press, New York,

Figure 18: The subworkflow iterating over several settings of the fuzzy c-means algorithm to identify the optimal number of clusters and the value of the fuzzifier.

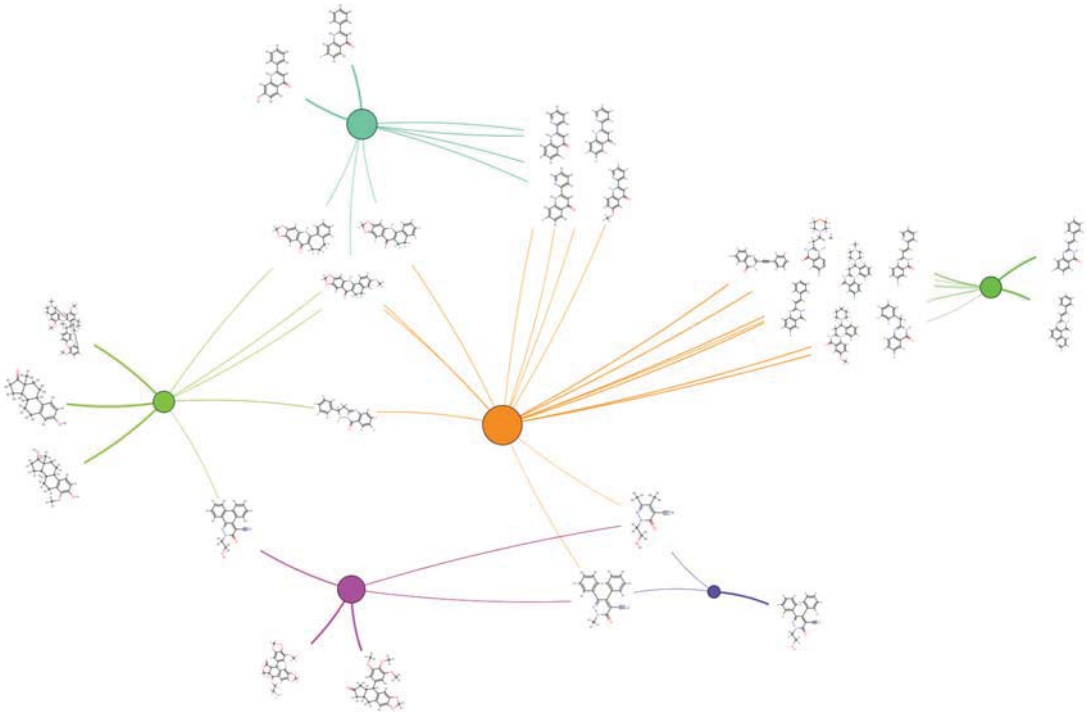

Figure 19: The final network as displayed by KNIME. One can nicely see how a couple of molecular structures fall clearly within one cluster. Others, however, belong to more than one cluster and have therefore substantial connections to more than one cluster node.

1981.

8. R.D. Clark. "Relative and absolute diversity analysis of combinatorial libraries Combinatorial Library Design and Evaluation," Marcel Dekker, New York, 337-362, 2001.
9. T. Curk, J. Demsar, Q. Xu, G. Leban, U. Petrovic, I. Bratko, G. Shaulsky, and B. Zupan. "Microarray data mining with visual programming," *Bioinformatics*. **21**(3), 396–408, 2005.
10. R.N. Davé. "Characterization and detection of noise in clustering," In *Pattern Recognition Letters*, **12**, 657–664, 1991.
11. R.A. Fisher. "The use of multiple measurements in taxonomic problems," In *Annals of Eugenics*, **7**(2), 179–188, 1936.
12. Th.R. Gabriel and M.R. Berthold. "Influence of fuzzy norms and other heuristics on 'Mixed Fuzzy Rule Formation'," In *International Journal of Approximate Reasoning (IJAR)*, **35**, 195–202, Elsevier, 2004.
13. Th.R. Gabriel, K. Thiel, and M.R. Berthold. "Rule Visualization based on Multi-Dimensional Scaling," In *IEEE International Conference on Fuzzy Systems*, 2006.
14. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, **11**(1), 2009.
15. A. Guazzelli, W. Lin, and T. Jena. "PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics."
16. M. Hanss. "Applied Fuzzy Arithmetic, An Introduction with Engineering Applications," Springer, 2005.
17. I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. "YALE: Rapid Prototyping for Complex Data Mining Tasks," In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006.
18. D. Morent, K. Stathatos, W.-C. Lin, and M.R. Berthold. "Comprehensive PMML Preprocessing in KNIME," In *Proceedings of the PMML Workshop*, KDD, 2011.
19. B. Wiswedel, D.E. Patterson, and M.R. Berthold. "Interactive Exploration of Fuzzy Clusters," In: J.V. de Oliveira, and W. Pedrycz (eds) *Advances in Fuzzy Clustering and its Applications*. John Wiley and Sons, 123-136, 2007.
20. L.A. Zadeh. "Fuzzy sets," In *Information and Control*, **8**(3), 338-353, 1965.