# DBPRO: Data Analytics using KNIME open source tool

Eric Dörheit
eric.doerheit@campus.tu-berlin.de

Olga Bode
olga.bode@mailbox.tu-berlin.de

Dorothea Poljak
dorothea.poljak@gmx.de

Supervisor:
Dr. Marcela Charfuelan

July 12, 2015

**Abstract**

**We present our results of the evaluation of the open source tool *KNIME* which is used for data analytics and data mining. We choose anomaly detection as the subject to evaluate *KNIME* as many methods of data analytics such as clustering, classification and statistical techniques are applicable to anomaly detection [1]. As a data set for the analysis we use the data provided for the *DEBS Grand Challenge 2012* [2].**

## Contents

# 1 Introduction

For the evaluation of *KNIME* we first investigated the tool by following several white-papers provided by *KNIME*. Our report is structured as follows: We start with describing the functionalities and the usage of *KNIME*. In Section 1.2 we explain the data set on which we detect anomalies. Section 1.3 provides an overview of anomaly detection based on [1] and in Section 2 we use *KNIME* to apply anomaly detection. Finally, we summarize our work and impression of *KNIME* in Section 3.

## 1.1 The Open Source Tool KNIME

In this section we give an overview on *KNIME* open source tool. *KNIME* is an open source tool for data analysis. It includes different functions, methods, algorithms and analytical functionalities including machine learning techniques and data mining capabilities. It offers a graphical user interface to use its functionalities. In order to enable scientists and users from other fields than computer science to apply data analytics methods to their data, there are no programming skills needed to use *KNIME*.

The basic idea of *KNIME* is to combine nodes with different functionalities to workflows which are represented as graphs. There are nodes for data access from different sources such as databases, files and web services, data transformation and pre-processing (ETL), data analysis and visualization. *KNIME* is implemented in Java and benefits from the modularity of the underlying *Eclipse* platform. It is possible to execute Java, Perl and Python code fragments inside of workflows. Hence, one of the advantages of *KNIME* is the integration of different programming languages and other tools known in the field of data analytics such as *Weka* and *R*.

### 1.1.1 Workflow

The data handling process is represented as a directed graph in *KNIME*. The data is structured in tables and passed along the graph. Each node solves different tasks. These nodes can either be configured statically or with flow variables. Parallel tasks are executed in different threads so that multiple algorithms can be executed concurrently. The workbench of *KNIME* allows to create and edit workflows. In particular it offers a project overview, the workflow editor view and a node repository.

### 1.1.2 Nodes

*KNIME* nodes are single processing units of the workflow using a data set as input and output for the other nodes. Furthermore, each node can have a number of In-ports and Out-ports which can transport additional information such as a learned model or a database connection object. The use of the nodes ranges from modeling (*Linear Regression Learner*) to the data manipulation (*Append* node), from visualization nodes like *Scatter Plot* to input-/ output-nodes for read and write operations. The next advantage of *KNIME* is the ability to monitor the processing state as there are three possible node states:

- Red: Node is inactive and not yet configured
- Yellow: Node is configured but not executed
- Green: Execution was successful

*KNIME* provides a large variety of nodes and there is an API so that custom nodes can be developed and used in *KNIME*. We give a short overview of a few nodes we used often.
For importing the files containing the data set we use *File Reader* node in combination with the *File List* node, for filtering columns we use the *Filter* node, for grouping and aggregating we make use of *Group By* node, for appending two tables together the *Append* node is useful etc. To benefit from custom code snippets without needing to develop a whole new node the *Java Snippet* node is helpful as it enables to process each row of an input data set. *KNIME* also provides connections to databases using a collection of database nodes and allows to execute SQL statements it: *Database Writer* is used to transfer the results to the database and *Database Reader*, respectively, from database. We describe more advanced nodes such as nodes available for tasks such as clustering in Section 2.
The features described above make this tool a reasonable to use for data analytics tasks.

## 1.2 DEBS 2012 Grand Challenge

In this section we introduce the data on which we apply anomaly detection.

The goal of the *DEBS Grand Challenge* is to solve problems event processing problems faced in domains such as the high-tech manufacturing industry. In this area problems of event-based data management and analysis arise. The *DEBS 2012 Grand Challenge* data originates from 1,000 peaces

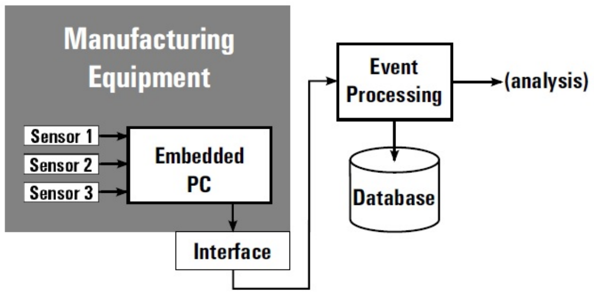

Figure 1: System architecture for the generation of data of the DEBS 2012 Grand Challenge[2, p. 1]

of geographically distributed high-tech manufacturing equipment. The raw data is gathered by sensors within manufacturing equipment using an embedded PC and storing in a flat file. Afterwards, the system integrates this file into a database to perform the analysis of data applying a set of developed queries (Figure 1). The goal of data monitoring is to record, analyze and then detect possible violations of predefined *Key Performance Indicators* (KPIs) to bridge the gap between the actual occurrence of the event causing a violation and the detection of such a violation [2].

The *DEBS Grand Challenge* data gathered by sensors embedded within a high-tech manufacturing equipment contains about 77 million entries collected during 18 days (roughly 8,5 million entries per day). There are binary and analogue sensors. The binary sensors generate data with 0 or 1 and the analogue between 0 and $2^{15}$. They produce data with a frequency between 100 Hz and 1000 Hz. The sensor data is collected and aggregated into a single event which contains time stamp (*ts*) and unique identifier (*index*) fields generated by PC. The rest of the event fields are the values recorded by sensors embedded within the equipment. The events are at least 100 nanoseconds apart [2].

The data set we use for our approach has a volume of 5 GB including 12 days of monitoring and the data of 32 sensors which change their value frequently (the other sensor data filtered out). We extracted *year, month, day, hour, minute, second* and *millisecond* from the *ts* column and created extra columns for each of time measurements applying *Data Manipulation* nodes in *KNIME* (later it facilitates the aggregation operations on the data).

## 1.3 Anomaly Detection

Anomalies in data are patterns which do not conform to the expected behavior and anomaly detections deals with finding this patterns [1]. There are many techniques that can be applied to detect anomalies. Subsequently we describe classification based, nearest neighbor based and clustering based techniques as well as statistical anomaly detection techniques.

### 1.3.1 Based on Classification

There are different classification technique to develop the classifiers, such as *decision tree* [5], *naive Bayes*, *association rules*, *neural networks* etc. [1]. Each technique is a two-step process. In the first step the algorithm used to build classifier (learning model) from a labeled training data set. In the second step learning model classify a test set into normal or anomalous classes. There are two broad categories of classification based anomaly detection techiques:

1. supervised learning - one-class anomaly detection technique
2. unsupervised learning - multi-class anomaly detection technique

One-class classification provides that training set has only one class and each test instance which does not fall into learnt model is considered as possible anomaly. In comparison to one-class classification multi-class classification assume that training set belongs to a multiple normal classes. Such technique learn a classifier to differentiate between each of the normal class and the rest of the classes. In this case each test instance which does not classify as normal is declared to be anomalous [1].

The focus of our approach lies on building the learning model applying one-class classification algorithm using decision tree technique in *KNIME* (Section 2.3.1).

### 1.3.2 Based on Statistical Techniques

An alternative for detecting anomaly is to use statistical methods. Therefore, it is needed to use a statistical model that can approximate data instances without a large distance to the real values. If this model is proven to predict values in most cases, then this model can be used to detect anomalies since data instances which are far away from predicted values can be assumed to be anomalies. The focus of this section is on Regression analyses.

**Regression** Regression is used in statistic to find a linear, polynominal or other correlation of data attributes in a data set. The objective is to find a good approximation of the data and the ability to predict an attribute out of the other attributes of a data instance.

In the multiple Linear Regression the variable $y$ is a linear combination out of the other parameters of the data instance. The Variable $x$ is independent, can take all possible values and there are multiple functions of the independent variable.

For example
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + e$
with the linear combination of the parameter $\beta_0, \beta_1$ and $\beta_2$.

In general regressions use the method of least squares to find out parameter $\beta_i$.

The method searches a hyperplane that approximates the data. The first step is to set the right functional class. Polynomial function is the class we had chosen. In the second step the parameter of the function will be decided so that the sum of the squares variations the observations $y$ is minimized to the values of the function.

### 1.3.3 Based on Clustering

Clustering is a technique that assigns data instances to clusters. There are clustering algorithms which assign each data instance to a cluster and also algorithms which let data instances unassigned. In order to use a clustering as a means to detect anomalies, it is necessary to assume what an anomaly is in the context of clustering. There are three assumptions usable for this purpose [1]:

1. Normal data instances belong to clusters while anomalies do not.
2. Normal data instances can be found closest to a cluster centroid while anomalies are far away from their closest cluster centroid.
3. Normal data instances lie in dense clusters while anomalies are in small or sparse clusters.

Assuming the first statement, algorithms such as DBSCAN, ROCK and SNN clustering are usable as they do not assign each data instance to a cluster. Based on the second assumption algorithms such as Self-Organized Maps (SOM), k-Means clustering as well as Expectation Maximization (EM) can be used due to the fact that they not only assign a data instance to a cluster but also compute its distance to its closest cluster centroid. Given the third assumption the use of various clustering algorithms is possible but it is needed to additionally compute the density/size of each cluster and define thresholds below which the data instances of these clusters are anomalies.

# 2 Anomaly Detection with KNIME

In the following we describe how to detect anomal data instances in the *DEBS 2012 Grand Challenge* data set using *KNIME*. Therefore we start with explaining the handling of the large data set and proceed with the ETL process. Subsequently, we focus on the application of the techniques illustrated in Section 1.3.

## 2.1 Handling of large Data Sets

*KNIME* offers many possibilities to pull data out of different data sources as described in Section 1.1. Since *KNIME* stores data either in-memory or on the local filesystem, it is not possible to directly take advantage of a a distributed filesystem such as *HDFS* within *KNIME*. Hence, we need a way to handle large files.

### 2.1.1 Split into smaller Files

The first solution is simply splitting the large file into smaller ones, e.g. with a command-line tool such as *split* or *awk*, and then making use of the loop and file list nodes in *KNIME* to iterate over the newly created files and execute the analytics on each of the files.

### 2.1.2 Use of Databases

The second option is to use a database to store the data set. Therefore, you can also split a large file into smaller ones, iterate over them and append a database table with the files data. An other way is not to use *KNIME* and to directly import the file into the database. In our case we use a *MySQL* database and the workflow shown in Figure **??**.
To import data from a database into *KNIME* there are multiple nodes available to build SQL queries and execute them. We use a loop to import chunks of data so that we do not run out of memory as this decreases the performance of *KNIME*.

### 2.1.3 Big Data Extensions

An alternative for handling large files is to make use of the Big Data Extensions which *KNIME* offers. They enable to connect to a *HDFS*, where you can upload files to or read files from, and to *Hive* as well as *Impala*, which allows to execute SQL queries on an *Hadoop* system. These extensions make it possible to access data from these sources but do not accelerate *KNIME* in the favor of executing *KNIME* workflows in a cluster.

## 2.2 The ETL Process with KNIME

ETL describes the process of extracting, transforming and loading data. In the extractions phase data is extracted from various data sources. In our case we already have one homogeneous file. Then the transformation phase follows where the data is altered in order to be able to run queries and do analysis. Therefore, we split the timestamp into multiple columns and filter columns which remain constant. Afterwards we store the new tables into a *MySQL* database.

## 2.3 Anomaly Detection with KNIME

In Section 1.3.3 we describe multiple approaches to detect anomalies. We apply these techniques on the data explained in Section 1.2.

### 2.3.1 Classification

We apply classification algorithm to the following task. The authors of [2] proposed to monitor the energy consumption of the manufacturing equipment to record the raw data where its value higher as normally (the possible anomalous can be detected). They recommend to calculate the average value and the relative variation for each of the sensors *mf01*, *mf02* and *mf03* per second and then to analyse: if the relative variation of any energy consumption sensors more then 30% from its average values, the raw sensor data needs to be recorded (the relative variation triggers the recording of the raw values of the sensor) [2].

**Decision tree classifier** From amount of the classification techniques mentioned in Section 1.3 we decided us for *decision tree* - one of the most successful techniques for supervised classification learning [4].

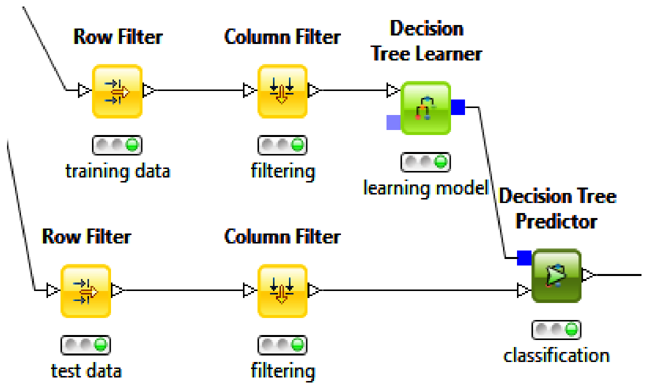

Figure 2: Decision Tree Learner and Decision Tree Predictor in Knime

We distinguish between normal and anomalous classes applying one-class classification anomaly detection technique. The low energy consumption is considered as a normal class (relative variation less than 30%) and the high energy consumption as an anomalous class (relative variation exceeds the threshold of 30%).

First, *Java-Snippet* node signs each data event as high or low at the class column. Second, *Row Filter* splits the data into two parts: training and test set. The training set including a class column consists of the data which gathered during first 7 days of monitoring and the test set without class column consists of the data which monitor the last 5 days. After employing *Column Filter* node both sets contain only the columns with average sensor data per second (*predictors*). Third, *Decision Tree Learner* [3] builds the learning model using training instance as input. After employing a learning model *Decision Tree Predictor* node classify the training instance into one of above defined classes see *Figure 2*. As output we got classified data and specified decision tree.

**Classifier estimation** *KNIME* offers two possibilities for classifier estimation: *ROC Curve* [5] and *Confusion Matrix* [5].

Predicted class is the output of developed learning model as well as probabilities for the class. Using probabilities *ROC Curve* node creates graphical plot that illustrates the trade-off between the true-positive rate - proportion of positive events (low energy consumption) that are correctly identified - and the false-positive rate - proportion of negative events (high energy consumption) that are incorrectly identified as positive. The area

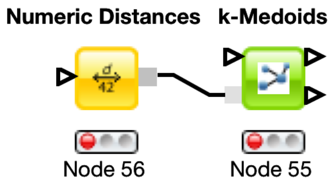

Figure 3: Clustering nodes using a distances

under the curve measures the accuracy of the model: the best classification model has the largest area under the curve. Perfect accurancy has an area of 1.0 [5]. *ROC Curve* also allows to compare a few classifiers by joining the class probability columns from the different predictors into one table.

Before to develop the confusion matrix in *KNIME*, first we merged into one table the assigned classes of the test set before and after applying the learning model (*Joiner* node). *Scorer* node gets this table as input and as output generates the confusion matrix which gives an overview how well the classifier can recognise events of different classes. Clearly, the accuracy rate of nearly 99% is a very good result, but it may not be acceptable, if the classifier correctly predict only low class. In our case after analysing the confusion matrix we got following results: 0.2% of all low classified events and 9.5% of all high classified events the model predicts incorrectly.

Consequently, applying described above classification technique in *KNIME* and in progress developed learning model based on the average sensor values it is possible to detect anomalous as high energy consumption in data set described in Section 2.

### 2.3.2 Clustering

In *KNIME* there are several nodes which implement different clustering algorithms. For some algorithms (Figure 4) you need a first node which trains a model and a second one which assigns the data instances to clusters. Other algorithms such as DBSCAN need a distance model, for instance the euclidian distance, as displayed in Figure 3. There are also nodes which directly assign data instances to clusters such as the *Fuzzy c-Means* node. This node can be configured so that is clusters all data instances as well as to declare some data instances to be noise.
For the anomaly detection through clustering we use the attributes *mf01*, *mf02*, *mf03*, *pc13*, *pc14*, *pc15* of the data set (see Section 1.2) and we load a chunk with the size of 1,000,000 rows out of the *MySQL* database on

which we apply clustering techniques. We show how to find anomalies with clustering for each assumption described in Section 1.2.

**Anomalies do not belong to clusters** We use the *Fuzzy c-Means* node and configure it to allow noise. All data instances which are declared as noise are anomalies. We use a *Java Snippet* node to append the column *anomaly_1* of type *Boolean* which is *true* if the data instance is assigned to noise to the input data.

**Anomalies are away from cluster centroids** Assuming that anomalies can be found far away from cluster centroids we need to assign all data instances to a cluster and to determine the distance to the cluster centroid. Therefore we use the *k-Medoids* node. All data instances are clustered through this algorithms and the names of the clusters are the RowIDs of the cluster centroid. Since we know the centroids we then compute the euclidian distance of each data instance to its centroid with a *Java Snippet* node. Afterwards we use another *Java Snippet* node which appends a column *anomaly_2* of type *Boolean* which is *true* if the distance of the data instance to its cluster's centroid is beyond a threshold.

**Anomalies are in small/sparse clusters** For this assumption we also need to cluster all data instances so that we can reuse the workflow of assumption two. We now count the number of data instances of each cluster and declare the smallest cluster as the cluster of anomalies. Therefore, we also use a *Java Snippet* node and all data instances which are anomalies have the value *true* in a third added column *anomaly_3*.

To visualize the results we create a diagram which contains all sensor values in relation to the time and show when anomalies are detected as you can see in Figure 5 (You can see the sensor values of *mf01* and *mf03* in blue and red and detected anomalies in green relating to the time on the x-axis).
After applying all techniques we save the new data set containing the columns to the database so that we can reuse our results.

### 2.3.3 Statistical Anomaly Detection Techniques

In *KNIME* the regression analysis is realized through different Regression nodes. We are using the polynomial regression learner node and the regression prediction node these nodes approximate best our scatter plot.

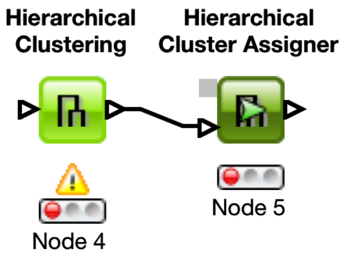

Figure 4: Clustering nodes with cluster assigner

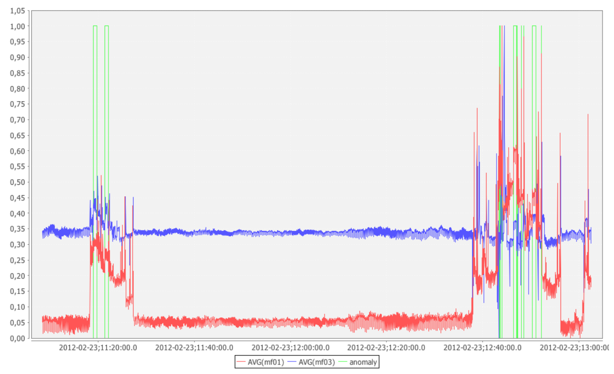

Figure 5: Anomalies detected with clustering

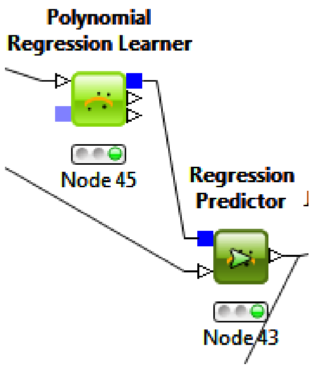

Figure 6: Regression learner and predictor

For the polynomial regression node we choose a target column and a number of independent variables. In this case we are using *mf01, mf02* or *mf03* as the target and for the other independent variables we select *pc13, pc14* and the other energy columns that are not used. At the end we have three models of polynomial regression.

To aim on possible anomalies with our regression model we are assuming that our constructed model describe in general our data and process of the data.

We use for prediction the constructed model (Figure 6). As input we receive the data and the regression node. Our prediction node predicts for each value of our data set with help of our model a prediction. Due to our assumption the values of the predicate target column and it original values should be almost the same, if the predicted values are very different from the real value of our target column, and then we are identifying it as a possible anomaly. This we are doing with the help of a java snippet node, where we are calculating the margin of the predicted value and it original.

For the particular case the difference between both values is to big, it is claim as an anomaly. We found out that for the energy *mf03* and the difference 0.05 for *mf02* 0.12 and for *mf01* 0.1 this value shows when we find a possible anomaly.

With the help of scatter plots node and statistic nodes we discover that our regression model converges well and identify that most of the values have a small margin between the predicated and the original value. It reveals this statistic method is a good way to find anomalies.

## 3 Conclusions

*KNIME* is a useful and easy to utilize tool for advanced data analysis. It facilitates the usage of complex machine learning algorithms as the user does not need to know their exact functioning. In fact *KNIME* provides its users many options to analyze data with the help of the manageable interface and its diversity of nodes. However, in *KNIME* is not possible to process the workflows and their machine learning algorithms in a distributed computer system (cluster) so that *KNIME* is not suitable to process large amounts of data in a reasonable time (Big Data Processing). There are extensions of *KNIME* which enable to receive data sets from Big Data platforms such as *HDFS*, *Impala* etc., but *KNIME* can only be used to orchestrate tasks for these platforms (for instance *SQL* queries for Impala) and to access the data but not to execute *KNIME* workflows on theses platforms.
*KNIME* facilitates the integration of different tools like *R* and *Weka*. The user can adapt *KNIME* to his purposes with the help of different programming languages and the API to create own nodes. In summary *KNIME* is a flexible tool that provides quick and extensive data analysis capabilities.

## References

[1] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[2] Z. Jerzak, T. Heinze, M. Fehr, D. Gröber, R. Hartung, and N. Stojanovic. The debs 2012 grand challenge. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 393–398, New York, NY, USA, 2012. ACM.

[3] Tom M. Mitchell. Machine Learning, chapter *Decision Tree Learning.* McGraw-Hill, 1997.

[4] David L. Poole, Alan K. Mackworth. *Artificial intelligence foundations of computational agents.* New York Cambridge University Press, 2010.

[5] Jiawei Han, Micheline Kamber. *Data mining: concepts and techniques.* chapter *Classification and Prediction.* Morgan Kaufmann Publishers. March 2006. ISBN 1-55860-901-6.