Breast Cancer Dataset Analysis - KNIME

Camila Perez and *Meritxell* Arbiol

Abstract. In this paper we are going to train several models testing also different preprocessing steps to obtain a higher accuracy in the result, and then be able to select the model that allows us the best possible classification. The dataset used is about breast cancer and the purpose is to determine if the tumor is benign or malignant using several classification models with their respective validations; using only the KNIME tool. First the data have been treated by doing a general exploration and preprocessing steps, so that the dataset is as clean as possible and only with what really implies an improvement for the different models to be able to train them correctly with the aim of finding the best model using cross-validation.

1 Introduction

In this project we are going to analyze a dataset using KNIME software. We'll perform several data mining processes testing different learning models to see what differences exist in the final classification results depending on the model chosen and why. We are also going to apply some dimensionality reduction techniques in order to see if this improves the final results or not.

It is always good to keep in mind that in order to obtain good results, before doing any analyses on our data, we have to make sure that it meets certain quality standards.To ensure quality data we have to explore it in order to know if certain values need to be transformed or removed to have a complete and homogeneous data set.

As we know, the most common steps to be followed in a Data Science project are:

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation
- Deployment

For this project we will focus on 2, 3, 4 and 5 steps. We are going to perform all of them exclusively through the KNIME tool.

2 KNIME

2.1 What is KNIME?

Knime is an analytic platform free and open source for data analytics, reporting and integration platform for creating data science. It was built on the Eclipse platform and programmed mainly in Java. It allows us to develop models in a visual environment. It is a graphical tool which has a series of nodes and arrows. Each node encapsulates different types of algorithms and each arrow represents the data flowing between nodes that are displayed in a graphical and interactive way.

2.2 What is KNIME for?

As we have mentioned in the point above, thanks to this tool we can create workflows in an easy and intuitive way. A strength of this tool is that no programming knowledge is required, since everything is done through a drag-and-drop style graphical interface.

When we say that using KNIME is very intuitive, it is because it is all visual. We have a series of node repositories that are grouped according to their functionality. This grouping makes it easier to find the workflow we want to represent.

In case of being a new user of this tool, there are many examples already made so that it is easy to understand what each of the existing components is for and how to use them to achieve the desired function.

3 Data

3.1 Data Exploration

Our dataset is about breast cancer and can be found in the UCI Machine Learning repository maintained by the University of California, Irvine.

It should be noted that it is the most common cancer disease in the world. For this reason, we believe that it can be interesting and important to work with this data.

An alternative to reduce this mortality rate is to be able to diagnose in a timely manner in time, since a tumor is not always cancerous; the big challenge presented by these data is to predict when the tumor is malignant and when it is benign for each of the cases, taking into account the attributes available to us.

The breast cancer dataset contains 569 samples, and 32 attributes. The first column shows the unique ID numbers of the samples, the second attribute is the target, and corresponds to the diagnosis (M=malignant tumor, B=benign tumor), respectively. The following 29 columns correspond to features calculated from a digitized image of a sample taken from a breast mass.

This data set is composed by 10 features which were calculated from each real cell core: radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter, area, smoothness (local variation in radius lengths), compactness (perimeter*2 / area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry, fractal dimension ("coastline approximation" - 1). For each image other attributes were created; mean, standard error and worst (mean of the three largest values).[1]

These attributes describe the characteristics of the cell nucleus present in the image, which can be used to build a predictive model to determine whether a tumor is benign or malignant.

In addition to the information provided by the repository and kaggle, Knime allows us to discover the structure and properties of the dataset in detail.

After extracting the csv file from the source, we use Knime's CSV Reader to import the dataset into the platform. We added the statistic node in order to calculate statistical moments such as minimum, maximum, median, mean, standard deviation, variance among others. And also we perform interactive univariate data exploration using the Data Explorer node, which is pretty similar to the information that we obtained from the statistics node, but with more details and with the possibility of displaying properties of the input data in an interactive view. We can observe that all the attributes are numerical values, except for one, which is the nominal attribute 'diagnosis' (also target). Using the bar chart node we can also visualize the distribution of our target, with the dataset containing 357 benign (62.7%) and 212 malignant (37.3%) tumor samples; so we can indicate that the dataset is very unbalanced.

In addition, from the data obtained we see that there is a coarse difference in the ranges of the attributes; there are very large numbers as in area_worst (max 4254), and very small ones as in concave points_mean (mean 0.059). This could indicate the need to normalize or standardize our data. We also use the boxplot node to visualize the outliers, given the large number of predictors, we group them into the 3 classes of measures: Mean, Standard Deviation and Worst. The visualization of the data distribution has already shown enough presence of outliers, and if there are few they can be eliminated, if there are more, other alternatives will have to be found to deal with them.

Knime provides some tools to analyze correlation present in our dataset; for this we add the correlation ranking node and the linear correlation node. The former calculates correlations between all predictors, including our target nominal attribute; the linear correlation node calculates only correlations between numeric-numeric and nominal-nominal pairs, but not if they are numeric-nominal.
Based on the correlation matrices and correlation tables, we see that many variables are highly correlated, and those with the strongest correlations (over 98%) between them are:

- Radius_mean with perimeter_mean/area_mean
- Radius_worst with perimeter_worst/area_wost

We have analyzed our dataset, and when we are in the modeling phase we will see if this information could be useful in the improvement of the models.

3.2 Data Preparation

Data Preparation could also be called Data Cleaning. Here we make sure that there are no duplicates, and if there are, we eliminate them.

We also look for null or defective values. If there are, we look at whether there are many or not, since if there are few we can eliminate the row, but if there are many we lose a lot of data by eliminating, and perhaps some type of imputation could be made.

In the analysis performed above we realized the need to normalize our data due to the large amplitude difference between some predictors, so we added the normalizer node in Knime, to bring the data to a certain typical range between 0-1, without the need to assume any underlying distribution.This will allow the future model to create a more accurate prediction based on a more symmetrical distribution.

Another thing we have to take into account in this step is to analyze if all the variables are necessary for the analysis, and if not we eliminate those we do not want to. In this dataset we have a total of 32 variables, so it is possible that some of them do not provide us with anything during the analysis to make the classification later. Here we can check if there exist some high similarities between variables. When we have a high similarity between two or more variables, it is not necessary to take them all into account for the analysis, since working with correlated variables is not good since the performance of the model will be impacted by a problem called "Multicollinearity". So we calculate the correlation matrix between each pair of variables, to take the results into account later for each of our prediction models, and in this way we can make several tests and see if the results in the classification improve by using only those uncorrelated variables. Therefore we use column filter node to remove highly correlated variables previously analyzed (perimeter_mean, area_mean, perimeter_worst and area_wost); and also to take out the next steps the poorly correlated features with our target to achieve good performance (fractal_dimension_mean, texture_se, smoothness_se, symmetry_se,fractal_dimension_se).

Regarding outliers detection, in the case of the dataset we are dealing with, we originally have 569 rows, of which 56 are considered outliers with an interquartile range multiplier of k = 3. We can consider that the number of outliers does not reach 10% of the total of our dataset, so we could consider eliminating them since they are few. Before eliminating anything, we are going to test for each of our models to know if eliminating the outliers improves the result or not.

4 Types of Analysis

Now we can create statistical models that allow us to get a good prediction. Before we must incorporate a Partition node, this node divides the future models into two partitions, training and test data. Another thing also needed to do is to normalize the data we are going to use for modeling.

In this section we explain how we create the models and train the algorithms according to achieve a good classification result by doing some small modifications on the data used for the training:

- Do not remove any row from the data
- Removing attributes highly correlated
- Removing outliers
- Removing attributes highly correlated and outliers

All the results achieved of these tests will be presented and detailed in the Results section.

4.1 Decision Tree

The decision tree is one of the nonparametric supervised learning algorithms used for classification and regression, although they are more commonly used in classification problems.

A decision tree is used to create a training model to predict the class of the target variable based on learning simple decision rules inferred from training data. It starts with one node, from which others emerge depending on the options presented, and from each of these, others. After building a classification model based on training data, the model is checked for accuracy and then the model is used to classify new data. Class labels are discrete values.[2]

Some components of a decision tree are:

- **Root node:** represents the entire population or sample and will then be split. It is also a decision node.
- **Decision node:** When a node is divided into other subnodes, the data are separated into two groups depending on a decision.
- **Leaf/terminal node:** The leaf node is no longer split; it is where the final decision is made. In the decision tree each leaf node represents a class.

To build a classification model using a decision tree in knime, we add the following components:

- **Decision Tree Learner Node:** This node incorporates a classification decision tree in main memory. There are two quality measures for the split calculation; the gini index and the gain ratio.
- **Decision Tree Predictor node:** This node uses an existing decision tree to predict the class value for new patterns.
- **Decision Tree View node:** A plot of the provided decision tree using a JavaScript based library.
- **Scorer:** It allows to compare two columns by their attribute value pairs and shows the confusion matrix.
- **X-Partitioner and X-Aggregator:** the first node initializes a cross-validation loop, and at the end of the loop the X-Aggregator collects the results of each iteration. All nodes between these two nodes are executed as many times as iterations are to be performed.

This KNIME structure can be seen in Figure 1.

Based on "entropy" which is the measurement of impurities or randomness in the data points; the algorithm provides two quality measures for the division calculation: the Gini index and the gain ratio. For calculating those measures, we need information gain, which detects the best features that give as much information as possible about a class. [3]

$$Entropy = -\sum P_i * Log_2(P_i)$$

p: probability that it is a function of entropy

Information Gain = Entropy before splitting - Entropy after splitting

Gini index computes the probability of a specific variable that is classified wrongly when it is randomly chosen: [4]

$$Gini\ Index = 1-\sum(P_i)^2$$

Considering that there is not much difference in performance when using the gini index compared to the gain, and that both gini and entropy can be used as a splitting criterion, we decided to use it as it improves the accuracy slightly, taking into account the best features to predict the target.

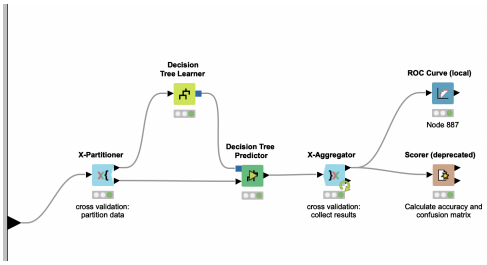

Figure 1. Decision tree - Knime metanode

4.2 Random Forest

This model is closely related to the previous one (Decision Tree). For the Random Forest model we combine several Decision Trees in parallel and in this way the variance we obtain at the end is lower than that obtained with the Decision Tree model. This is because each of the Decision Trees is trained using a different sample of the dataset, so the result obtained does not depend only on one Decision Tree, but on many. Thus, the final output is obtained by taking into account the majority voting classifier.[5]

The Random Forest Algorithm use two techniques called Bootstrap and Aggregation:

- **Bootstrap:** Technique used to randomly select row and feature sampling from the dataset in order to create sample datasets for each Decision Tree model.
- **Aggregation:** Technique used to get the final classification using the majority voting classifier taking into account the results of each Decision Tree model.

These techniques are illustrated in Figure 2.

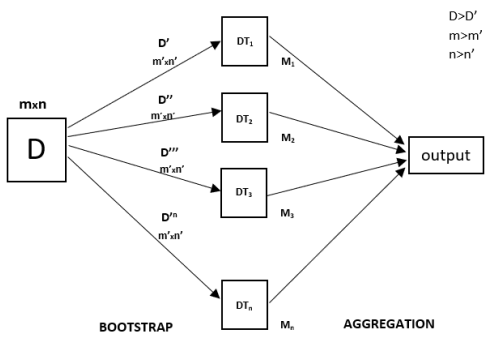

Figure 2. Bootstrap and Aggregation techniques

To apply this model in KNIME we incorporate the Random Forest Learner which consists of a chosen number of trees, in our case the number of trees is 50. Each of the decision models is built within a tree randomly chosen from a sample from our training data using the bootstrapping technique already mentioned before. And thanks to the Aggregation technique the model used for the classification is built. We connect this node to the Random Forest Predictor in order to predict the class per row of our data test based on the learned model. In the scorer node we can see our confusion matrix and the accuracy of the model.

In the image below (Figure 3), we can see our metanode that composes the Random Forest analysis. In the Random Forest Learner node we have our training data previously normalized as input, and in the Random Forest Predictor, it has as input the learned model and the test data (without having been previously normalized) to know how good the model is at classifying.

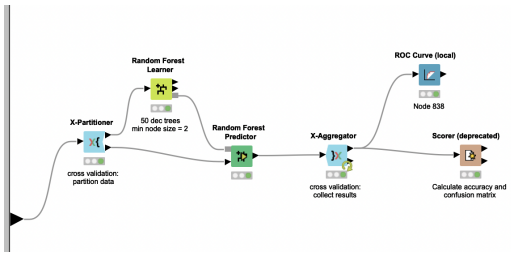

Figure 3. Random Forest - Knime metanode

4.3 Naive Bayes Model

Naïve Bayes Models are populars learning algorithms based on Bayes Theorem:

$$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$$

Naïve Bayes assumes that all predictors are independent and contribute equally to the target.[6]

From Bayes Theorem, in classification problems we can find the class variable(y) with maximum probability, as follows:

$$\text{y=argmax}_y * P(y) \prod P(x_i|y)$$

With the function above we can obtain from the predictors the class of the target.

Types of Naïve Bayes Classifier:

- **Multinomial Naïve Bayes Classifier:** It is used for discrete counts.
- **Bernoulli Naïve Bayes Classifier:** It is useful when multiple features but each one is assumed to be a binary-valued variable.
- **Gaussian Naïve Bayes Classifier:** It assumes that predictors follow a normal distribution.

Among the advantages of naïve bayes algorithm is that it allows to quickly predict the class of the test data set and works well in multi-class predictions. The assumption of independent predictors, on the other hand, is part of the disadvantages, since it is highly improbable that a set of predictors are completely independent.

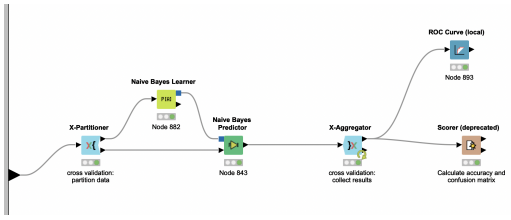

Figure 4. Naive Bayes - Knime metanode

As illustrated in Figure 4 above, to create a Naive Bayes model in Knime from the given training data, we incorporate the Naive Bayes Learner node. This computes

the Gaussian distribution for the numerical attributes, and after being connected to the Naive Bayes Predictor node we achieve to predict the class per row based on the learned model. The probability of the class is obtained from the product of the probability per variable and the probability of the class variable itself. We also added the X-Partitioner and X-Aggregator nodes to perform cross-validation loops.

4.4 Logistic Regression

Logistic Regression is a supervised classification algorithm where the target variable can only take discrete values for a given set of predictors.[7]

This model predicts the probability of a given data to belong to the category 0 or 1, in our case the categories are M or B (malignant or benign tumor). Logistic regression models the data using the sigmoid functions, this can be seen more clearly in Figure 5.

$g(z)=\frac{1}{1+e^{z}}$

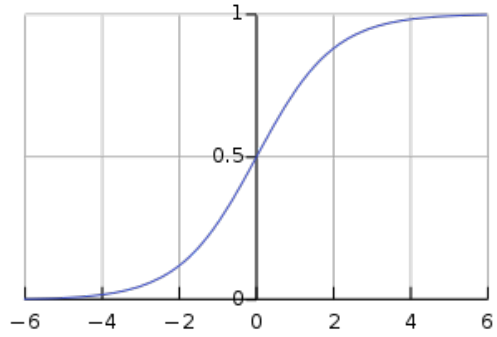

Figure 5. Sigmoid Function

Logistic regression can be classified depending on the number of categories: Binomial (our target has 2 possible values), multinomial (our target can have 3 or more categories) and ordinal (target variables with order categories). In the case of our dataset, it is a binomial regression, since we are only going to classify in two categories.

To apply this model in KNIME we incorporate the Logistic Regression Learner node to perform a multinomial logistic regression selecting our target column (Diagnosis) using only the training data. We connect this node to the Logistic Regression Predictor node to predict the class per row of our data test based on the learned model. In the scorer node we can see our confusion matrix and the accuracy of the model.

In the Figure 6, we can see our metanode that composes the Logistic Regression analysis. In the Logistic Regression Learner node we have our training data previously normalized as input, and in the Logistic Regression Predictor node, it has as input the learned model and the test data (without having been previously normalized) to know how good the model is at classifying.

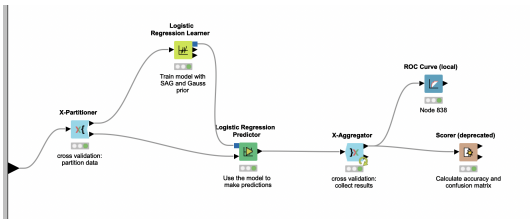

Figure 6. Logistic Regression - Knime metanode

5 Results obtained in each analysis and comparison between them

We compare accuracy and AUC score for every model. Taking into account that accuracy gives us the percentage of correct predictions for the test data, and AUC is the probability that a positive instance randomly chosen ranks higher than an arbitrary chosen negative instance; the result of both metrics allow us to have a good approach to know which model is better. Subsequently, we can use the pruning method to reduce the tree size and increase the prediction accuracy.

5.1 Decision Tree

Taking into account accuracy measures the best classification performance of the model is achieved by removing outliers from the data set with an accuracy of 94,15%, and AUC value of 94.77%. However, the highest AUC value (96,39%) is obtained by removing features highly correlated. This may happen because the accuracy is calculated on the predicted classes while the ROC AUC is calculated on the predicted scores, so the difference is related to the threshold. All numerical results are shown in Table 1.

Table 1. Results obtained with the Decision Tree model

Acc.	Keeping correlation and outliers	94,03%
	Removing outliers	94,15%
	Removing features highly correlated	93,32%
	Removing outliers + high correlations	93,73%
AUC	Keeping correlation and outliers	95,60%
	Removing outliers	94,77%
	Removing features highly correlated	96,39%
	Removing outliers + high correlations	93,16%

5.2 Random Forest

As we can see in the numerical results of Table 2, the best classification result is achieved when neither the features with high correlation nor the outliers are removed from the data set, with an accuracy of 95.982%, and we have 99.01% of the ROC area under the curve. As we can see, when we remove the outliers, the accuracy value decreases slightly. On the other hand, we see that the AUC value increases a little when we remove outliers and highly correlated characteristics, which seems curious to us but can happen.

Table 2. Results obtained with the Random Forest model

Acc.	Keeping correlation and outliers	95,78%
	Removing outliers	95,52%
	Removing features highly correlated	95,26%
	Removing outliers + high correlations	95.52%
AUC	Keeping correlation and outliers	99,01%
	Removing outliers	98,71%
	Removing features highly correlated	98,81%
	Removing outliers + high correlations	99,25%

5.3 Naive Bayes Model

The accuracy and the AUC values for the classification using Naive Bayes Model are shown in the Table 3. Taking into account accuracy measures the best classification performance of the model is achieved by removing outliers from the data set with an accuracy of 94,15%, and AUC value of 98,62%. However, the highest AUC value (98,70%) is obtained Without taking correlation nor outliers. As mentioned above this may happen because the accuracy is calculated on the predicted classes while the ROC AUC is calculated on the predicted scores, so the difference is related to the threshold.

Table 3. Results obtained with the Naive Bayes model

Acc.	Keeping correlation and outliers	93,67%
	Removing outliers	94,15%
	Removing features highly correlated	91,92%
	Removing outliers + high correlations	93,16%
AUC	Keeping correlation and outliers	98,70%
	Removing outliers	98,62%
	Removing features highly correlated	97,96%
	Removing outliers + high correlations	98,04%

5.4 Logistic Regression

We see that the best results are achieved when we do not eliminate outliers or highly correlated features, obtaining a higher accuracy in the classification and in the AUC value. More detailed results can be found in Table 4.

Table 4. Results obtained with the Logistic Regression model

Acc.	Keeping correlation and outliers	93,85%
	Removing outliers	93,37%
	Removing features highly correlated	91,74%
	Removing outliers + high correlations	92,79%
AUC	Keeping correlation and outliers	98,83%
	Removing outliers	98,68%
	Removing features highly correlated	98,25%
	Removing outliers + high correlations	98,68%

6 Conclusions

As can be seen in the above results we decided to do several preprocessing tests to see how this affects the final result for the various models. Thus, for the preprocessing we have analyzed the different correlations that exist between the variables that we use as predictors to train the model and predict the target. We have tested each of the models by eliminating the most correlated variables, perimeter_mean, area_mean, perimeter_worst and area_worst, keeping radius_mean and radius_worst. We have seen that none of the models obtain better results if we previously eliminate these variables. In fact, if we review the results we see that it is by eliminating these 4 variables that we obtain a lower classification accuracy.

Another step that we wanted to do in the preprocessing and see if it would improve any of our models is to take into account the outliers and eliminate them. We see that by eliminating the outliers, the Decision Tree and Naive Bayes models improve slightly with respect to the other results.

There are two models that we have not yet mentioned, these are the Random Forest model and the Logistic Regression model. It seems that for these two models, eliminating outliers or removing highly correlated variables does not improve the results. The highest accuracies are obtained by training both models with the data only normalized, without further treatment.

We see that the best classification result is the Random Forest model, with a classification accuracy of 95.7%. This is followed by the result obtained with the Decision Tree model and the Naive Bayes model, both with an accuracy of 94.15%. The model with the lowest results is the Logistic Regression model, with an accuracy of 93.8%.

We see that the model that gives us the best results (Random Forest) is the most complex, which does not always imply that it has to be the one that gives us the best results, but in this case it has been so. On the other hand, we see that Logistic Regression is one of the simplest models available, and is the one that has given us the lowest accuracy, although we still have to take into account that the accuracy obtained is higher than 90%, so it is still a very good result.

References

[1] Y. Kalshtein, *Wisconsin breast cancer (diagnostic) dataset analysis* (2017), https://rstudio-pubs-static.s3.amazonaws.com/344010_1f4d6691092d4544bfbddb092e7223d2.html

[2] N.S. Chauhan, *Decision tree algorithm, explained* (2022), https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

[3] N. Tyagi, *What is information gain and gini index* (2021), https://analyticssteps.com/blogs/what-gini-index-and-information-gain

[4] N. Tyagi, *Understanding the gini index* (2020), https://medium.com/analytics-steps/understanding-the-gini-index-information

[5] S. Ray, *6 easy steps to learn naive bayes algorithm with codes in python and r* (2017), https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[6] A. Dutta, *Random forest regression in python* (2022), https://www.geeksforgeeks.org/random-forest-regression-in-python/?ref=lbp/

[7] A. Dutta, *Understanding logistic regression* (2021), https://www.geeksforgeeks.org/understanding-logistic-regression/

[1–7]

7 Appendix

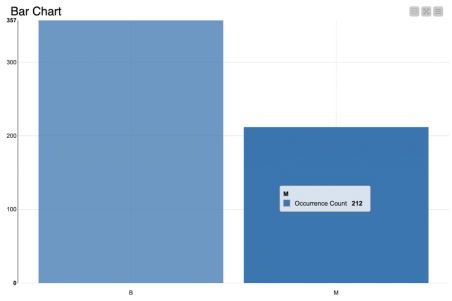

Figure 7. Diagnosis target - Knime

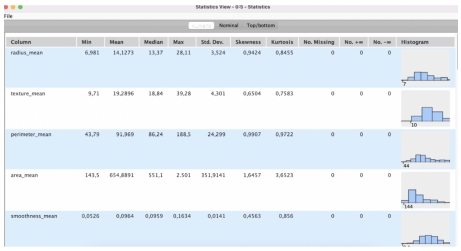

Figure 8. Statistics View - Knime

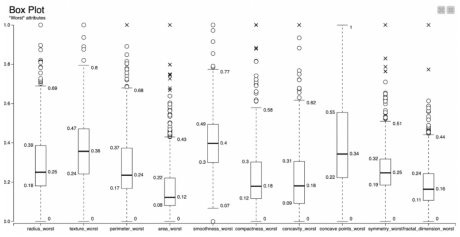

Figure 9. Attributes “worst measure” - Boxplot - Knime

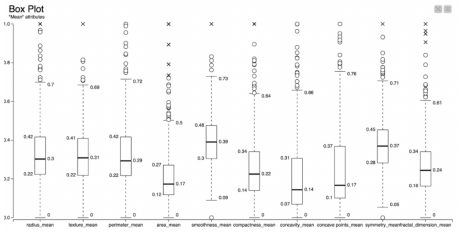

Figure 10. Attributes “mean measure” - Boxplot - Knime

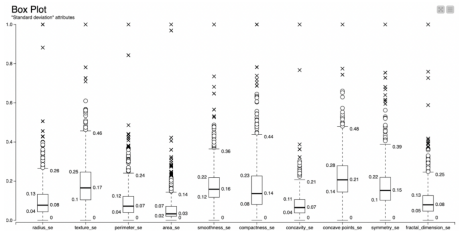

Figure 11. Attributes “standard measure” - Boxplot - Knime

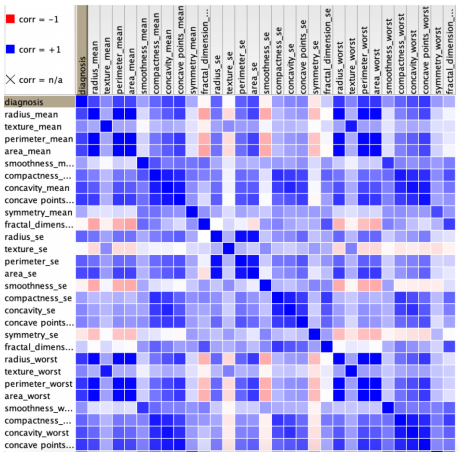

Figure 12. Correlation Matrix - Rank Correlation - Knime

Row ID	First column ...	Second colum...	Correlation value	p value
Row1	radius_mean	perimeter_mean	99,79%	0.0
Row391	radius_worst	perimeter_worst	99,37%	0.0
Row2	radius_mean	area_mean	98,74%	0.0
Row57	perimeter_mean	area_mean	98,65%	0.0
Row392	radius_worst	area_worst	98,4%	0.0
Row407	perimeter_worst	area_worst	97,76%	0.0
Row246	radius_se	perimeter_se	97,28%	0.0
Row76	perimeter_mean	perimeter_worst	97,04%	0.0
Row19	radius_mean	radius_worst	96,95%	0.0
Row74	perimeter_mean	radius_worst	96,95%	0.0
Row21	radius_mean	perimeter_worst	96,51%	0.0
Row100	area_mean	radius_worst	96,27%	0.0
Row103	area_mean	area_worst	95,92%	0.0
Row102	area_mean	perimeter_worst	95,91%	0.0
Row247	radius_se	area_se	95,18%	0.0

Figure 13. Correlation Measure-Linear Correlation - Knime