Documentation Of Data Exploration, Clustering And Classification Using The KNIME Platform

Jounaid Ruhomaun, March 2021

Background

The following report documents the development and testing of multiple different workflows that complete three data-science/machine-learning tasks, using the KNIME analytics platform. The first task experiments with dimensionality reduction though the use of *Principle Component Analysis*, with testing and validation around the *K-Means Clustering* algorithm as well different normalisation techniques for the provided wine dataset (with visualisations provided within the appendix section). The second task takes an in-depth look into the effectiveness of two classification methods/algorithms for said wine dataset, the *K-Nearest Neighbour* and *Decision Tree* models, whilst also commenting on the benefits of the *K-Fold Cross Validation* technique used. The final task provides an acceptable solution to the binary classification and prediction challenge surrounding the Large Hadron Collider dataset, which utilises *Random Forest Decision Tree* models as well as minority class oversampling through the use of the *SMOTE* technique.

Task 1 - Data Exploration And Clustering

The following section details the dimensionality reduction of the wine dataset using *Principal Component Analysis (PCA),* as well as experimentation of the *K-Means* clustering algorithm on said dataset.

K-Means Clustering

K-Means clustering is a form of unsupervised machine learning algorithm that attempts to infer groupings within an unlabelled dataset. The defined parameter, *K*, details the number of 'centroids' (centre points of clusters) to be assigned to the dataset, for which each datapoint is subsequently allocated to the nearest centroids cluster through reducing the inter cluster sum of squares value[1].

The algorithm starts by assigning *K* random centroids to the dataset, then subsequently performs iterative calculations in order to optimise the centroid locations (generally calculated using the datapoints Euclidean distances from the centroids to assign clusters, with the new centroid in each iteration being assigned to the mean location of all datapoints in said cluster). The stopping condition for the algorithm can be determined by either having an initial defined number of iterations, or by measuring the change in the centroids locations (with convergence triggering the halt)[2].

Entropy Scorer

In order to analyse and validate the clusters, the *Entropy Scorer* node is used. The entropy performance measure essentially defines how well separated the clusters are from each other, with lower values indicating better separateness. The quality performance measure is based of the FCM quality algorithm from the *Fuzzy Clustering in Parallel Universes*[3] publication, with higher values indicating better quality.

WSS And BSS

Similarly to the *Entropy Scorer* node, the *KMeansWSS* node implemented within the workflow provides WSS (within cluster sum of squares) and BSS (between cluster sum of squares) validity measures[4]. The WSS measure determines the sum of the distances between each datapoint and its corresponding centroid (used during the execution of the algorithm), with BSS being the sum of the distances between each centroid and the mean of all datapoints multiple by the total number of said datapoints. Therefore, WSS can be thought as a measure of compactness, with BSS (similar to entropy) being a measure of cluster separateness. A lower WSS score and higher BSS score is desirable.

Workflow Overview

Figure 1 below shows the overall workflow for both Tasks 1.1 and 1.2. The wine data set is input using a *CSV Reader* node, for which meta nodes named *Cluster + Plots* are then used to encapsulate the PCA, clustering and graphing of said data. This is done to avoid repetition whilst also keeping the workflow clean and understandable.

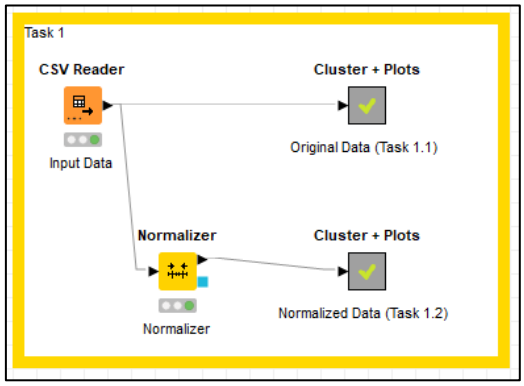


Figure 1

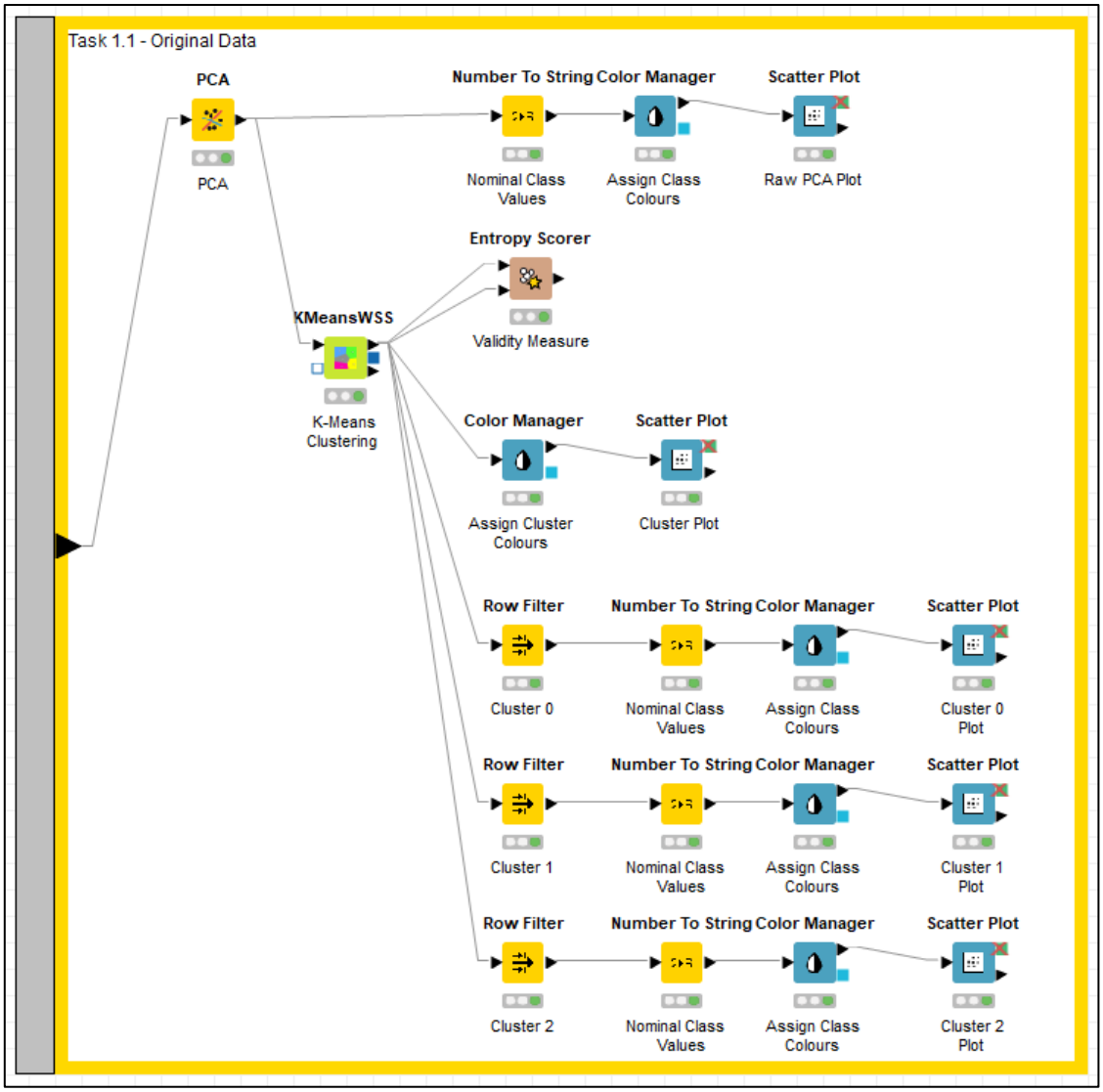


Figure 2

Figure 2 shows the inner workflow for the *Cluster + Plots* meta node used for Task 1.1 (which is identical to said node used for Task 1.2). The input data (excluding the class data) is first reduced to two dimensions using the *PCA* node (which appends the two PCA dimension columns to the end of the dataset) before having the datapoints coloured by class (Using the *Color Manager* node) and plot via the interactive *Scatter Plot* node in the topmost workflow branch. The scatter plot produced can be seen in Appendix 1.1.1 (Plot 1).

The colours green, red and blue (for classes 1, 2 and 3 respectively) are chosen as they are standard base colours that stand out from each other, simplifying graph analysis. It is also important to note that the *Color Manager* node requires the class data to be nominal in order for specific colours to be assigned, therefore a *Number To String* node carries out the conversion of class values before each subsequent plot (these nodes could be encapsulated in a further meta node, however such method would be overly verbose and wouldn't justify the increased encapsulation).

The second workflow branch clusters the data using the *KMeansWSS* node (which appends the calculated cluster column to the end of the dataset) before subsequent comparative plots and validity is carried out. As previously mentioned, the *Entropy Scorer* node is used with the original class and newly generated cluster columns to generate the validity statistics (quality and entropy).

The *Scatter Plot* node labelled 'Cluster Plot' displays the datapoints coloured by the newly generated cluster column. The colours chosen for clusters 0,1 and 2 are purple, orange and light yellow respectively as they are different than the colours used within plot 1, but are still easily differentiable within the plot. The scatter plot produced can be seen in Appendix 1.1.2 (Plot 2).

The bottom three scatter plots display each clusters datapoints, coloured by their class attribute (with the same colours used as in Plot 1). In order to extract the datapoints per cluster, three *Row Filter* nodes are used (one for each *Cluster ID*), before going through the same nominal data conversion (for class labels) and colour assignment as described in the topmost workflow branch. The three scatter plots produced can be seen in Appendix 1.1.3, 1.1.4 and 1.1.5 (for Plots 3a, 3b and 3c respectively).

As previously mentioned, the bottom branch in the overall workflow (shown in Figure 1) carries out the same dimensionality reduction and clustering as with the previously described *Cluster + Plots* meta node for Task 1.1, but with the dataset normalised using the *Normalizer* node. Said node has options for Min-Max, Z-Scored and Decimal Scaling normalisation (for which experimentation and comparison can be seen in Figure 3). The scatter plots produced mirror those generated for Task 1.1, labelled Plots: 4; 5; 6a; 6b and 6c, with each respective scatterplot produced shown in Appendices 1.2.1 to 1.2.5. Note that these plots where generated using Z-Score normalisation.

Experimentation And Results (Original Dataset)

Plot 1 (shown in Appendix 1.1.1) gives a useful visualisation of the datapoints class distribution with the unnormalized dataset, which shows that class 1 (green) is generally somewhat separate to the left of the plot, with classes 3 and 2 heavily overlapping towards the right. It is clear however that class 2 (red) is somewhat more concentrated to the right of the distribution in comparison to the other classes, suggesting normalisation would be effective in making said class distributions more separable.

In comparison with Plot 2 (shown in Appendix 1.1.2) which shows the *K-Mean Clustering* distribution for the unnormalized dataset, it can be seen that as expected the clusters are not entirely representative of the class distribution. Since, as previously mentioned, class 1 has a large proportion of datapoints that are separate from the other classes, it is expected that one cluster would be highly representative of said class, which is verified by the cluster shown in Plot 3c (Appendix 1.1.5). Plot 3a (shown in Appendix 1.1.3), which represents the class distribution of the middle cluster seen in Plot 2 (purple), shows that said cluster also has a high proportion of datapoints from class 1, with a few from classes 2 and 3. The final clusters (orange in plot 2) class distribution can be seen in Plot 3b (Appendix 1.1.4) which as expected is almost entirely representative of classes 2 and 3.

These results suggest that applying *K-Means* clustering to the original unnormalized dataset is somewhat ineffective in classifying said dataset, as only one cluster (Plot 3c) seems to be somewhat representative of an actual class, with the same class being largely represented in the second cluster (Plot 3a), and two classes overlapping in representation within the third cluster (Plot 3b). This is likely due to the datapoints class distribution overlapping heavily (seen in plot 1), which suggests normalisation could make an improvement on said clustering.

Experimentation And Results (Normalisation)

Figure 3 below shows a comparison of validity measures using the original dataset, as well as the three different normalisation methods tested. Min-Max normalisation carries out a linear transformation of the dataset in regards to the minimum and maximum parameters set. Z-Score normalisation carries out a similar process, however each datapoint is normalised based on a gaussian distribution. Decimal Scaling calculates a 'scaling' parameter, *j*, based on the number of times the max value within each column of the dataset is divisible 10 before a decimal value is produced. Each value in said column is then divided by 10^*j* to complete normalisation[5].

Normalisation Method	Entropy	Quality	WSS	BSS
Unnormalized	0.9420	0.4057	2.63E6	3.17E12
Min-Max (0 to 1)	0.2547	0.8393	48.99	1.25E7
Z-Score	0.2347	0.8519	1272.54	1028.49
Decimal Scaling	0.8809	0.4442	4.68	2.85E6

Figure 3

The clustering can be considered somewhat ineffective when carried out on the original unnormalized dataset, as its almost completely entropic and has a quality of less than 50%, which confirms the visual observations made within the plots (described previously). A large improvement can be seen with both Min-Max and Z-Score normalisation in comparison to the unnormalized dataset in terms of both entropy and quality, with Z-Score being slightly more effective. Given the fact that entropy is less than 30% and quality is greater than 80% with these two normalisation methods, it can be said that *K-Means* clustering is somewhat effective in classifying the wine dataset when normalised with said methods. Despite the Decimal Scaling method causing a slight improvement on the quality and entropy scores in comparison to the original dataset, said improvement is not sufficient enough for this normalisation method to be considered effective when clustering the wine dataset.

The WSS and BSS measures observed provide somewhat interesting results. The WSS score for the original dataset is expectedly high, with all three normalisation methods making vast improvements on said score. Interestingly, the Decimal Scaling method produces the lowest WSS score, with both Decimal Scaling and Min-Max having somewhat large BSS values (with a large difference between WSS and BSS scores). Z-Score normalisation however produces very close WSS and BSS scores, with its BSS score being slightly less than said WSS value. Given the WSS and BSS scores alone, an incorrect conclusion on the effectiveness of Decimal Scaling (and Min-Max normalisation to a lesser extent) could be made, however, since the *Entropy Scorer* node gives comparable quality and entropy measures, it can be assumed that somewhat similar (albeit relatively low) WSS and BSS scores are ideal when applying the *K-Means Clustering* algorithm on a dataset.

Plot 4 (shown in Appendix 1.2.1) which gives a visualisation of the datapoints class distribution with Z-Score normalisation, shows the increase in separability normalisation brings to the dataset (in comparison to Plot 1). There are three clear separate sections, with class 1 being in the top left of the plot, class 2 being in the bottom middle, and class 3 in the top right, with very little overlap between sections (slight overlap with classes 1 and 2). The clusters shown in Plot 5 (Appendix 1.2.2) are very representative of said class distributions, with three clearly separate corresponding clusters. The first cluster (purple in Plot 5) contains all datapoints pertaining to class 1, with only five datapoints from class 2, which is seen in Plot 6a (Appendix 1.2.3). The second cluster (orange in Plot 5) only contains datapoints from class 2 and is highly representative of said classes distribution. The last cluster (yellow in plot 5) is highly representative of class 3 with only three datapoints from class 2.

Conclusion

These results suggest that applying *K-Means* clustering to the original unnormalized dataset is somewhat ineffective in classifying said data, as only one cluster (Plot 3c) seems to be somewhat representative of an actual class, with the same class being largely represented in the second cluster (Plot 3a), and two classes overlapping in representation within the third cluster (Plot 3b). This is likely due to the datapoints class distribution overlapping heavily (seen in plot 1), which suggests normalisation could make an improvement on said clustering.

When normalisation is applied to the dataset, performance of the *K-Means* algorithm greatly improves, with validity measures that can be considered as being effective in clustering said dataset (mainly with Min-Max and Z-Scored normalisation methods). As is visually verified in Plots 5 to 6c, classes 1 and 3 are successfully grouped in highly separate clusters, with class 2 having a largely representative cluster (although said cluster does not completely contain all class 2 datapoints).

Overall, the experimentation in this task has shown how *K-Means Clustering* can be used to group unclassified datasets with a high-level of accuracy to said datasets inferred classifications, although, data transformations such as normalisation sometimes must be carried out in order to produce effective results.

Task 2 – A Comparison Of Classification Models

The following section details the training and validation of the two classification algorithms, *'K Nearest Neighbour (KNN)'* and *'Decision Tree'*, using the wine dataset. Performance testing and evaluation is carried out using *'10-Fold Cross Validation'* on said classification models.

K Nearest Neighbour

The KNN method takes the assumption that a datapoint is likely to be within the same classification as those within close proximity to it (which are referred to as neighbours)[6].

The algorithm is first initialised with the chosen number of neighbours, referred to as *K*, for which the distance between each query datapoint and every other point within the dataset is subsequently calculated. The distances are then ordered with the *K* smallest distanced datapoints being selected. The modal classification label is then calculated for the selected distances. Figure 4 shows a two-dimensional visualising of the algorithm being used to classify a query datapoint[7].

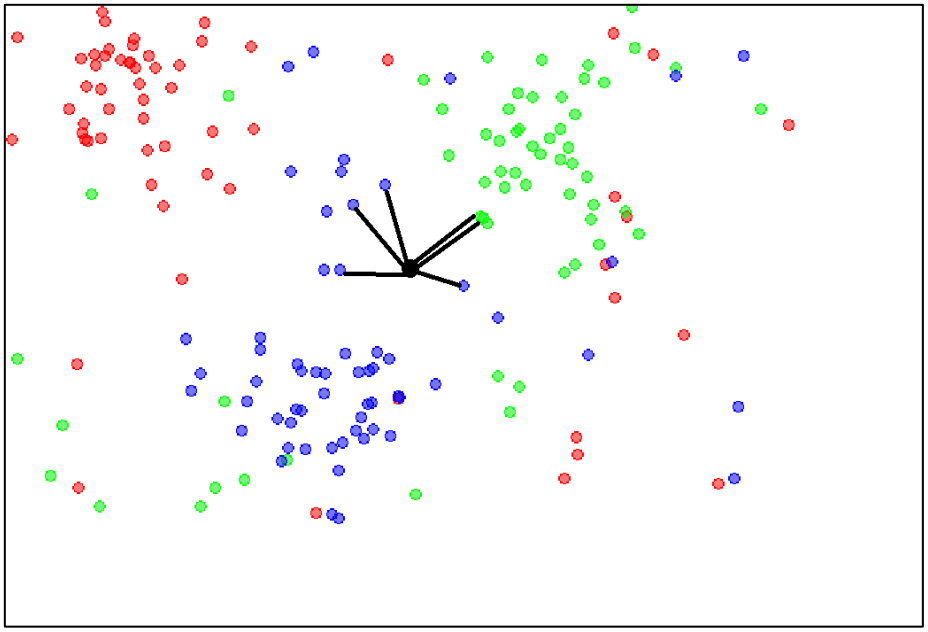


Figure 4

Decision Tree

This classification method takes the approach of generating a binary tree for the data set, where each node is assigned a condition based on an attribute from the dataset[8]. The leaf nodes are used to define the classification label assigned to a datapoint which has traversed the tree.

The decision tree generation algorithm begins by choosing an attribute that would cause largest 'information gain', meaning the largest difference between overall classification labels. This is then repeated iteratively at each child node until all the leaves are *'pure'*, meaning all datapoints at each leaf node are of the same class[9]. In order to prevent overfitting, a restriction on the minimum number of datapoints per leaf node can be set, which can result in some leaf nodes being *'impure'*.

An example decision tree generated from the KNIME workflow (for the wine data set) can be seen in Figure 5.

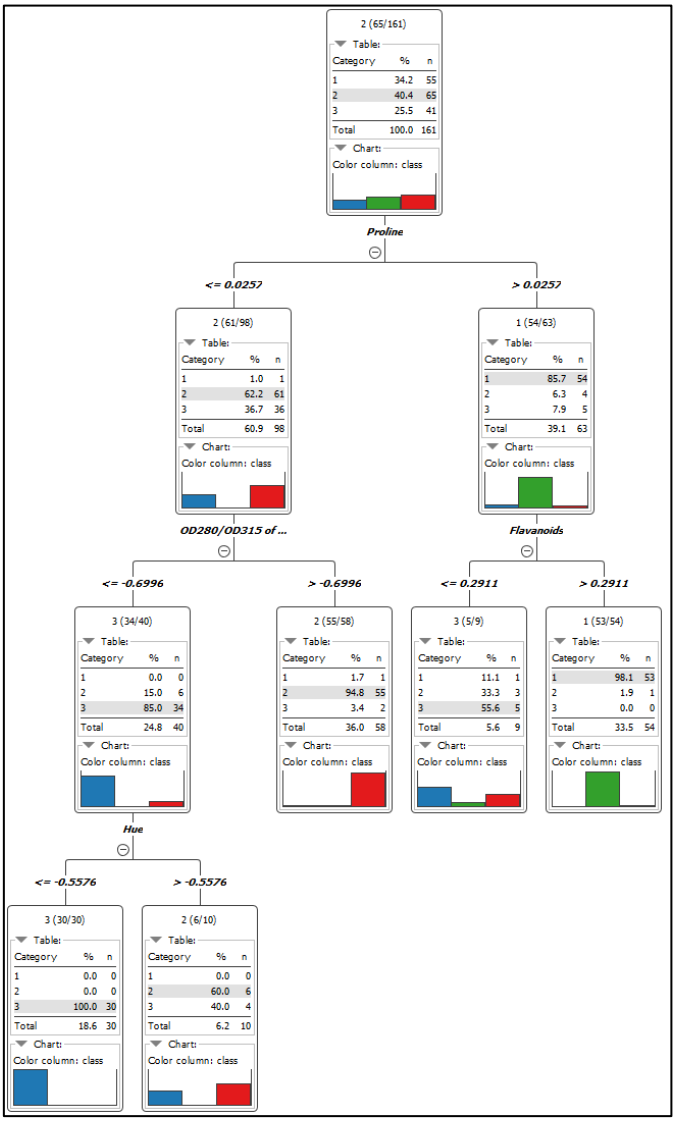


Figure 5

K-Fold Cross Validation

A common issue in regard to the training and testing of classification models is the partition split of the dataset. A larger training partition results in greater estimates and accuracy whereas a larger testing partition results in greater confidence in the model's predictions. There is also the issue surrounding the fact that different testing partitions would vary in accuracy. Therefore, a careful 'balancing act' of these partitions must be played in order to create an optimal and accurate model.

In order to get around these problems, the *'K-Fold Cross Validation'* method can be utilised. The dataset is split into K equal sections referred to as 'folds', where for K iterations the K^{th} fold is used as the testing partition (with the rest used in training)[10]. Each iterations errors are averaged resulting in a final overall generalisation error. If said generalisation error is satisfactory, the dataset can be recombined to produce the subsequent classification model. A

value of 10 for *K* is generally thought as being optimal, resulting in the most realistic accuracy[11], therefore such value is used during this tasks experimentation.

Workflow, Experimentation And Results

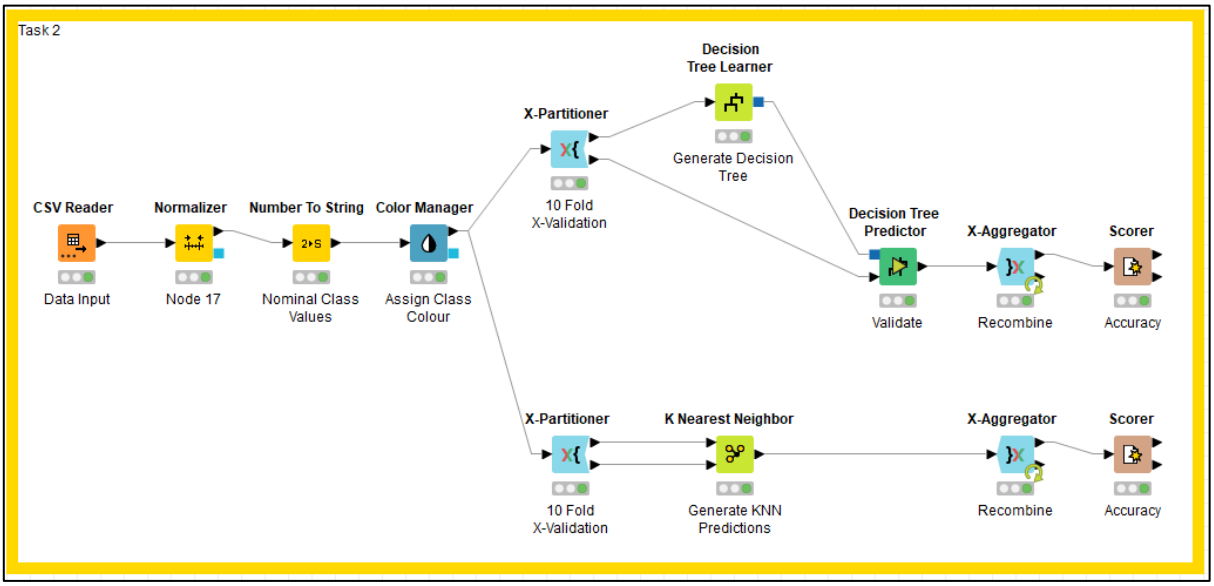


Figure 6

Figure 6 shows the KNIME workflow for the two classification models (on the wine dataset) described previously. The data is first normalised using z-score normalisation (to ensure fair model comparison) before having the class attributes coloured and converted to nominal values. The top workflow branch trains and tests the decision tree model using the *Decision Tree Leaner/Predictor* nodes. The bottom workflow branch uses the *K Nearest Neighbour* node that generates the predicted classifications (using said model) for the dataset, appended within a new column. Both models are placed between *X-Partitioner/Aggregator* nodes which carry out the 10-fold cross validation iterations. Finally, each model's validity measures is then generated using a *Scorer* node.

Min Records Per Node	Class 1 Precision	Class 2 Precision	Class 3 Precision	Overall Accuracy
1	0.887	0.897	0.917	0.899
2	0.948	0.867	0.933	0.910
3	0.982	0.905	0.939	0.938
4	0.932	0.845	0.854	0.876
5	0.930	0.853	0.811	0.865
6	0.945	0.843	0.774	0.854
7	0.950	0.921	0.818	0.899
8	0.965	0.840	0.870	0.888
9	0.898	0.857	0.816	0.860
10	0.964	0.836	0.820	0.871
Average	0.94	0.866	0.855	0.886

Figure 7

Figure 7 shows the individual class precision data as well the overall accuracy on training the decision tree model for different values of *Minimum Records Per Node*. Overall, class 1 has the highest precision with this model, with Classes 2/3 being somewhat similar. Lower *Min Records Per Node* values produce higher accuracy as expected (since leaf nodes are generally more *'pure'*), with *Min Records* = 3 having the highest accuracy at 0.938. However, there is a chance these training runs are overfitted, meaning those scores might not be representative of their actual accuracy. Therefore, a value of 7 for the *Min Records* could be seen as optimal, as it gives a good balance between overfitting and pureness whilst having a reasonably high overall accuracy of 0.899.

K	Class 1 Precision	Class 2 Precision	Class 3 Precision	Overall Accuracy
1	0.922	1.0	0.941	0.955
2	0.937	1.0	0.96	0.966
3	0.952	1.0	0.923	0.961
4	0.967	0.985	0.959	0.972
5	0.937	1.0	0.96	0.966
6	0.951	0.985	0.959	0.966
7	0.967	1.0	0.941	0.972
8	0.967	1.0	0.980	0.983
9	0.952	1.0	0.96	0.972
10	0.967	1.0	0.941	0.972
Average	0.952	0.997	0.952	0.969

Figure 8

Figure 8 shows the individual class precision data as well the overall accuracy with the KNN algorithm for different values of *K*. Class 2 generally has very high precision, with many training runs being 100% precise. Classes 1/3 also have somewhat high precision, both averaging 0.952 precision. When *K* is set to 1, the overall accuracy is comparatively lower than with other values of k¸ which is to be expected since the model is only using 1 datapoint (the closest) for classification. The highest accuracy tested was $K = 8$ with 0.983 accuracy which can be considered optimal for this model.

Conclusion

It is clear that the KNN classification algorithm is more effective than the decision tree classification model for the wine dataset. For all values of *K*, the KNN model generated a higher accuracy than the decision tree model, regardless of the value set for *Min Records Per Node*. What is interesting however is that the two models are more precise with different classes comparatively, with the decision tree model being more precise with class 1, and the KNN algorithm being more precise with class 2. This could suggest that class 2 is somewhat more tightly clustered. Overall, despite the decision tree model being somewhat less effective in comparison to the KNN model, both models can be seen as effective methods in classifying and predicting values for the wine data set.

Task 3 – A Binary Classification Challenge

The follow section details the workflows created in order to generate the *Task3-predictions.csv* file from the provided *test1K.csv* dataset, given classification models trained using the provided *training100Ku.csv* dataset.

The training dataset in question contains high and low-level attribute data produced at CERN's Large Hadron Collider (LHC), with two classes, background and signal (signal being particles of interest). Given the magnitude of data produced at the LHC, having classification models to predict if a particle is of interest is crucial in allowing efficient analysis during experiments.

There are three workflows provided for this particular solution, the first including experimentation to discover which classification models perform best on the dataset, the second being the actual predictions workflow from the *test1K.csv* dataset, and the third providing validation measures for said predictions workflow. Various nodes and techniques are used throughout the workflows such as *SMOTE* and Z-Score normalisation.

Model Selection

The first task is to experiment and verify the accuracy of various different classification models with the complete dataset (including high-level attributes) for which the most accurate model is to be selected for the overall prediction workflow (with high-level attribute prediction).

There is also the issue surrounding the imbalance of classes within the dataset, with the background class taking up a large proportion of datapoints. In order to compensate for this, the *SMOTE* (*Synthetic Minority Oversampling Technique*) node is also tested, which artificially increases the proportion of the minority class (in this case signal) through oversampling[12].

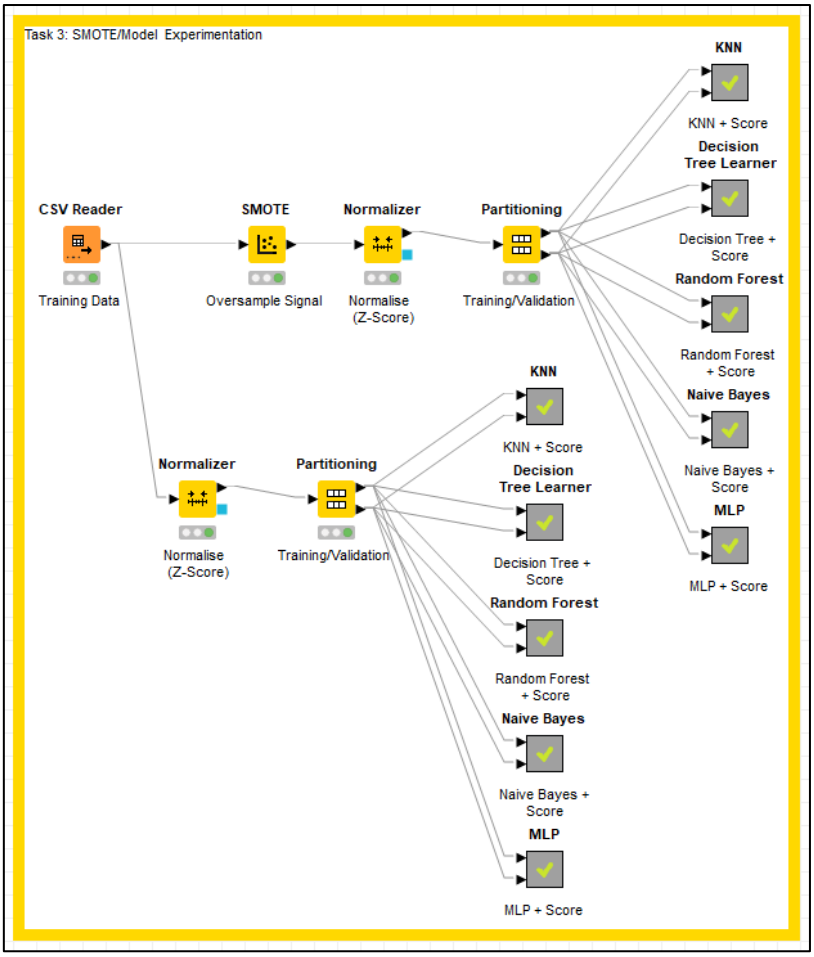


Figure 7

Figure 7 shows the workflow used for experimentation of different classification models as well as the *SMOTE* technique. The full training data (including high-level attributes) is input using a *CSV Reader* node, before having the signal class oversampled with a *SMOTE* node. The dataset is then normalised (using Z-Score *Normalizer* node) and partitioned (using the *Partitioning* node) with a 70/30 training/validation ratio. Despite the previously described benefits of using the *K-Fold Cross Validation* method as apposed to basic partitioning, the fact that in this case an extremely large amount of data is being processed (resulting in very high processing time) made such method unviable within the time constraints (as this workflow is simply for experimentation). The same partitioned data is then passed into five meta nodes, each corresponding to a classification method tested with a scoring node. The first two meta nodes, *KNN* and *Decision Tree Learner* correspond to the two classification methods originally tested during Task 2, with the optimal parameters found within Task 2. The *Random Forest* meta node utilised the *Random Forest Decision Tree* classification method, which utilises multiple Decision Trees each with a random subset of attributes, with the goal of minimising overfitting within said classification model[13]. The *Naive Bayes* meta node utilises the Naive Bayes classification technique (with default KNIME parameters) based of the Bayes theorem (which assumes independence among predictors)[14]. The final classification technique tested is within the *MLP* meta node which utilises the *RProp MLP Learner* node that trains a multi-layered perceptron based on the *RPROP* algorithm (with default KNIME parameters)[15]. The meta node contents for each model can be seen in Appendix 2.1 to 2.5.

Model	**Precision**		**Recall**		**F-Measure**		**Overall Accuracy**	**Precision (*SMOTE*)**		**Recall (*SMOTE*)**		**F-Measure (*SMOTE*)**		**Overall Accuracy**
	(B)	**(S)**	**(B)**	**(S)**	**(B)**	**(S)**		**(B)**	**(S)**	**(B)**	**(S)**	**(B)**	**(S)**	
KNN	0.90	0.24	0.97	0.08	0.94	0.12	0.88	0.99	0.74	0.65	0.99	0.78	0.85	0.82
Decision Tree	0.91	0.3	0.94	0.21	0.92	0.24	0.87	0.77	0.77	0.76	0.77	0.77	0.77	0.77
Random Forest	0.90	0.79	0.99	0.05	0.94	0.09	0.90	0.90	0.89	0.89	0.90	0.90	0.90	0.90
Naive Bayes	0.89	0.32	0.99	0.01	0.94	0.02	0.89	0.74	0.59	0.42	0.85	0.53	0.70	0.63
MLP	0.90	0.62	0.99	0.10	0.94	0.17	0.90	0.75	0.75	0.75	0.74	0.75	0.75	0.75

Figure 8

Figure 8 shows the class Precision (positive predictions that belong to the class), Recall (positive predictions out of all positive examples), F-Measure (balance between Precision and F-Measure)[16] and Overall Accuracy validity measures for both the original, and oversampled datasets for the classification methods used (B columns referring to

background, and S columns referring to signal). The first notable observation is that the *SMOTE* technique as expected has a large impact on the performance of all models, with the signal Recall and F-Measure validity measures particularly seeing a significant improvement. Also, when comparing the background F-Measure for the original dataset against the *SMOTE* dataset (for all models) there is a decline in performance, which is also to be expected as a lower proportion of the training data would consist of background classifications. However, since the objective of this task is to identify signal datapoints, said performance drop in background predictions is negligible. The overall accuracy produced with the original dataset models are generally close to 90%, with the *Random Forest* and *MLP* models being the most accurate. The *SMOTE* dataset trained models however generally see a decline in overall accuracy most notably the Naive Bayes model which drops from 89% to 63%. The *MLP* model which scored joint highest total accuracy with the original dataset also sees a significant decline once trained with the *SMOTE* dataset (from 90% to 75%). The most interesting observation however is the *Random Forest* model, which sees no performance loss in overall accuracy between the two datasets, and produces quite high scores across all validity measures tested. Therefore, the selected model to be used for the actual prediction workflow (including high-level predictions) is the *Random Forest* classification technique. Another acceptable model that could potentially be used however is the KNN method, as it has an overall accuracy (with *SMOTE*) of 82%, which can be considered as being 'just' good enough.

Prediction Workflow

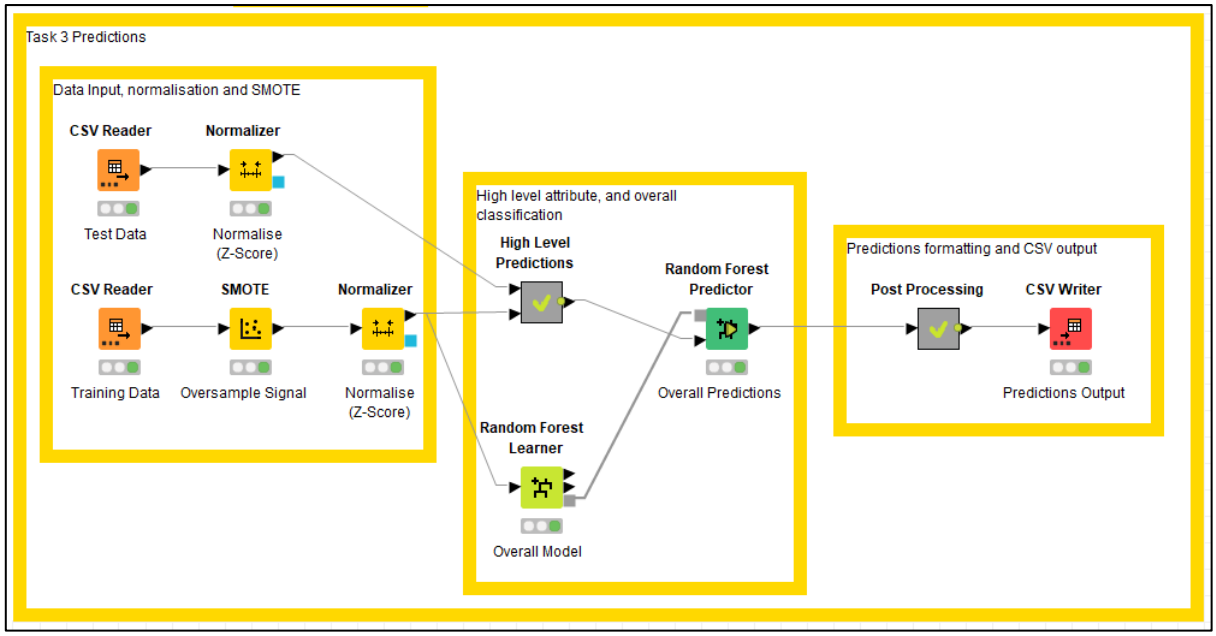


Figure 9

Figure 9 shows the overall prediction workflow for Task three that produces the *Task3-predictions.csv* file. Within the data input section on the left, both training and test datasets are input via *CSV Reader* nodes. The training data is then minority oversampled using a *SMOTE* node before being Z-Score normalised (with a *Normalizer* node). The test dataset is not oversampled however (as predictions of said oversampled records would be redundant), it is just Z-Score normalised using a *Normalizer* node. The training dataset is then used to train a *Random Forest Learner* node for which the resultant model is fed into a *Random Forest Predictor* node in order to generate the required predictions.

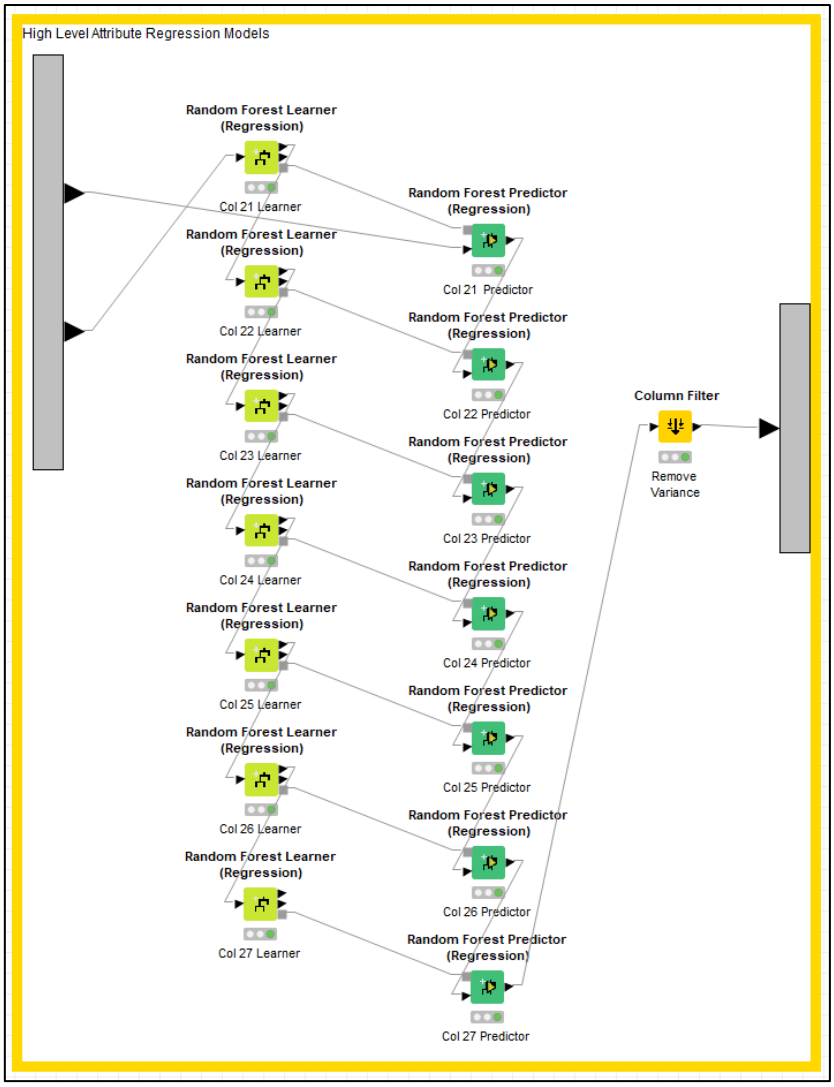


Figure 10

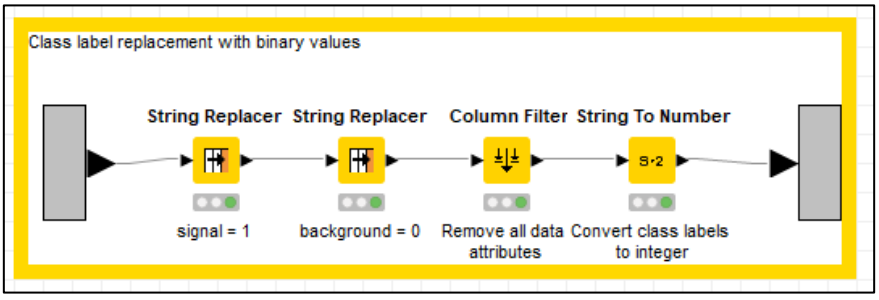


Figure 11

Since the test data does not contain the seven high-level attributes present within the training dataset, a meta node named *High Level Predictions* is used to generate said attributes within the test dataset. The contents of said meta node can be seen in Figure 10. There are seven *Random Forest Learner (Regression)* nodes within said meta node which are each trained using the input training dataset to predict each of the high-level attributes independently (based only on the 21 low level attributes). Seven respective *Random Forest Predictor (Regression)* nodes are then used to generate the missing high-level attributes for the input test dataset (predicted columns are named the same as with the training dataset). Both model training and prediction happens sequentially. The resultant test data with appended high-level attributes then has the prediction variance columns removed (generated within the *Random Forest Predictor (Regression)* nodes) before subsequently being output from the meta node.

The test dataset can then be used within the previously configured *Random Forest Predictor* node (Figure 9) to classify the dataset. The dataset is then put though the *Post Processing* meta node (seen in Figure 11) which first replaces the signal and background class labels with 1 and 0 respectively, before removing all attributes other than the class and *Record ID*. The binary class labels are converted to integer (from string) before the dataset is returned from the meta node. A *CSV Writer* node is then used to save said dataset in CSV format with the filename: *Task3-predictions.csv*.

Prediction Validation

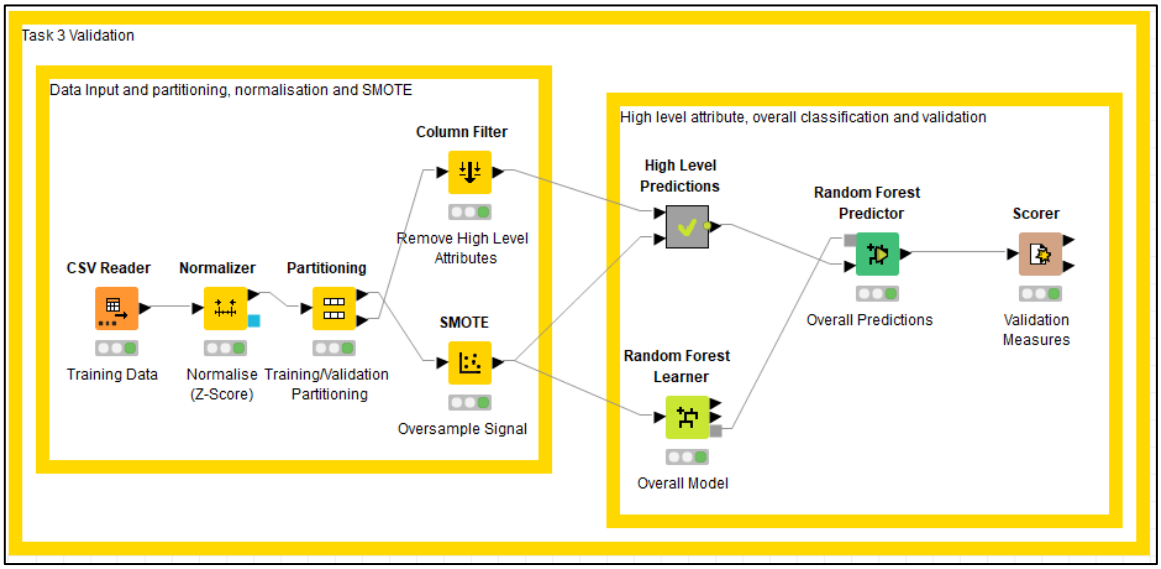


Figure 12

Figure 12 shows the prediction validation workflow which is somewhat similar to the workflow described in Figure 9, however only the training dataset is used. 30% of said training dataset is partitioned using *Partitioning* node and has its high-level attributes removed, which acts as the test dataset (note that the *SMOTE* node is not applied on this partition). After training and prediction is complete, the dataset is then put through a *Scorer* to generate validation measures.

Class	Precision	Recall	F-Measure	Overall Accuracy
Signal	0.16	0.35	0.22	0.75
Background	0.92	0.79	0.85	

Figure 13

Figure 13 shows the validation measures produced within the prediction validation workflow. It is clear that the background class are predicted more accurately than the signal class as is observed in the Precision, Recall and F-Measure validation scores (with Precision being particularly low). This is somewhat expected however, since although the signal class data was oversampled within the training dataset (with *SMOTE*), no new signal record data is actually generated. Despite this, an overall accuracy of 75% is observed, which although not is ideal, given the difficulty of this classification task can be thought of as acceptable. These validation measures might not be entirely accurate however since training/validation partitioning was used rather than the *K-Fold Cross Validation* method described in Task 2.

Conclusion

To conclude, through the use of the *SMOTE* technique, and *Random Forest Decision Tree* models an acceptable classification workflow has been produced for the LHC dataset. Out of five different classification models tested it was found that the most effective model for said dataset is the *Random Forest Decision Tree* algorithm, however other classification techniques such as Linear Regression could be tested and could potentially improve the workflows accuracy.

A significant problem during development of the workflows was the limitations of the hardware said workflows are run on. Despite running each workflow on an Intel i7-10750H with 6 cores (12 threads) at 5Ghz boost clock speed and 16gb RAM, the workflow often would freeze and have to be terminated forcibly. This occurred when originally attempting to the run the high-level attribute *Random Forest Learner*/*Predictor* nodes in parallel, resulting in the current workflow where each node is executed sequentially. Each node also must be executed manually on said hardware, as attempting to execute the whole meta node (utilising node queueing) also results in workflow crashing (which is rather time consuming and tedious). This issue with system performance is also why *K-Fold Cross Validation* could not be carried out within the prediction validation workflow, which as mentioned previously could have provided more accurate statistics. A solution to this problem could have been to run the workflow using a cloud service such as AWS (Amazon Web Services), however such solutions are generally quite costly.

References

[1] Piech, C. (n.d.). *K-Means*. Https://Stanford.Edu/. https://stanford.edu/~cpiech/cs221/handouts/kmeans.html (Accessed: 2021, March 20)

[2] Garbade, M. J. (2018, September 13). Understanding K-means Clustering in Machine Learning. Medium. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1 (Accessed: 2021, March 20)

[3] Wiswedel, B., & R. Berthold, M. (2005, October). Fuzzy Clustering in Parallel Universes. University of Konstanz. https://www.uni-konstanz.de/bioml/bioml2/publications/Papers2007/WiBe07_fcum_ijar/Fuzzy%20Clustering%20in%20Parallel%20Universes_submitted.pdf (Accessed: 2021, March 20)

[4] Who is your Golden Goose?: Cohort Analysis. (n.d.). KDnuggets. https://www.kdnuggets.com/2019/05/golden-goose-cohort-analysis.html/2 (Accessed: 2021, March 20)

[5] Normalizer. (n.d.). KNIME Hub. https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.normalize3.Normalizer3NodeFactory (Accessed: 2021, March 20)

[6] Zhang, Z. (2016a). Introduction to machine learning: k-nearest neighbors. Annals of Translational Medicine, 4(11), 218. https://doi.org/10.21037/atm.2016.03.37. (Accessed: 2021, March 20)

[7] Nelson, D. (2020, August 24). What is a KNN (K-Nearest Neighbors)? Unite.AI. https://www.unite.ai/what-is-k-nearest-neighbors/ (Accessed: 2021, March 20)

[8] Decision Tree Tutorials & Notes | Machine Learning. (2018, May 10). HackerEarth. https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/ (Accessed: 2021, March 20)

[9] Chakure, A. (2020, November 6). Decision Tree Classification - The Startup. Medium. https://medium.com/swlh/decision-tree-classification-de64fc4d5aac (Accessed: 2021, March 20)

[10] Berrar, D. (2019). Cross-Validation. Encyclopedia of Bioinformatics and Computational Biology, 542–545. https://doi.org/10.1016/b978-0-12-809633-8.20349-x (Accessed: 2021, March 20)

[11] Koha, R. (n.d.). A Study of CrossValidation and Bootstrap for Accuracy Estimation and Model Selection. Stanford University. http://robotics.stanford.edu/~ronnyk/accEst.pdf (Accessed: 2021, March 20)

[12] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953 (Accessed: 2021, March 20)

[13] Random Forest - an overview | ScienceDirect Topics. (n.d.). Sciencedirect. https://www.sciencedirect.com/topics/engineering/random-forest (Accessed: 2021, March 20)

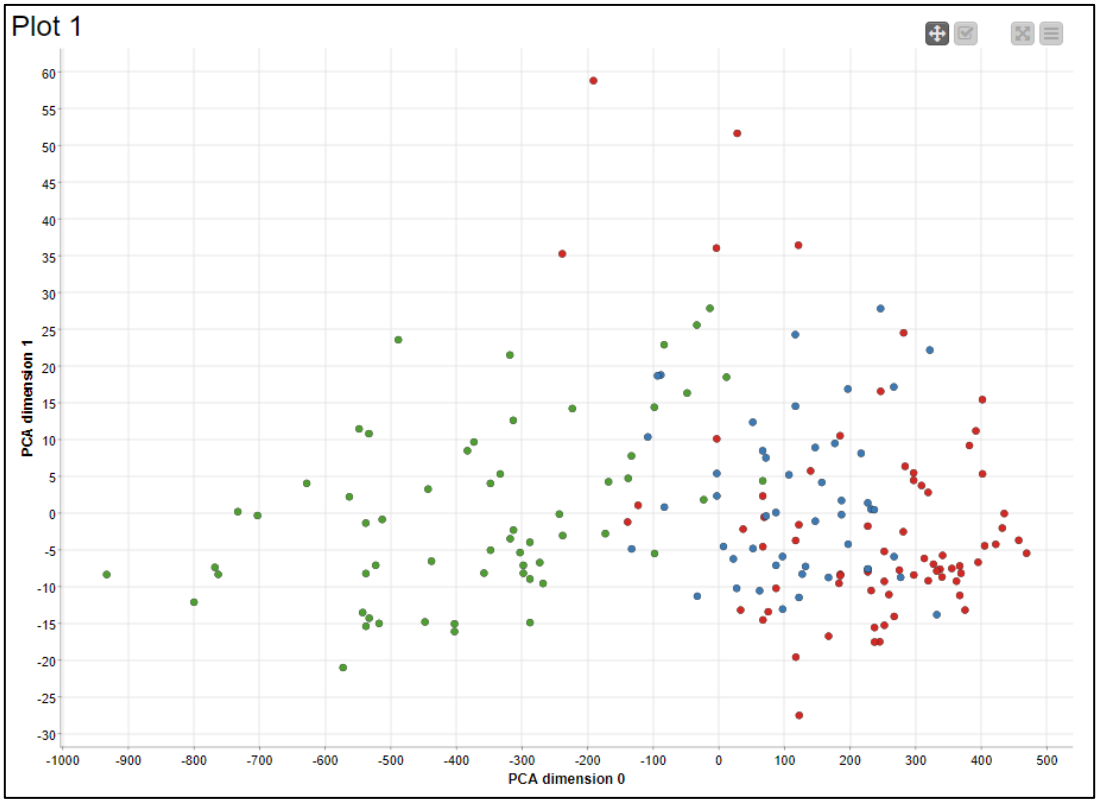
[14] Ray, S. (2020, October 18). 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/#:%7E:text=Naive%20Bayes%20Model-,What%20is%20Naive%20Bayes%20algorithm%3F,presence%20of%20any%20other%20feature. (Accessed: 2021, March 20)

[15] RProp *MLP* Learner. (n.d.). KNIME Hub. https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.mine.neural.rprop.RPropNodeFactory2 (Accessed: 2021, March 20)

[16] Brownlee, J. (2020, August 1). How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification. Machine Learning Mastery. https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/ (Accessed: 2021, March 20)

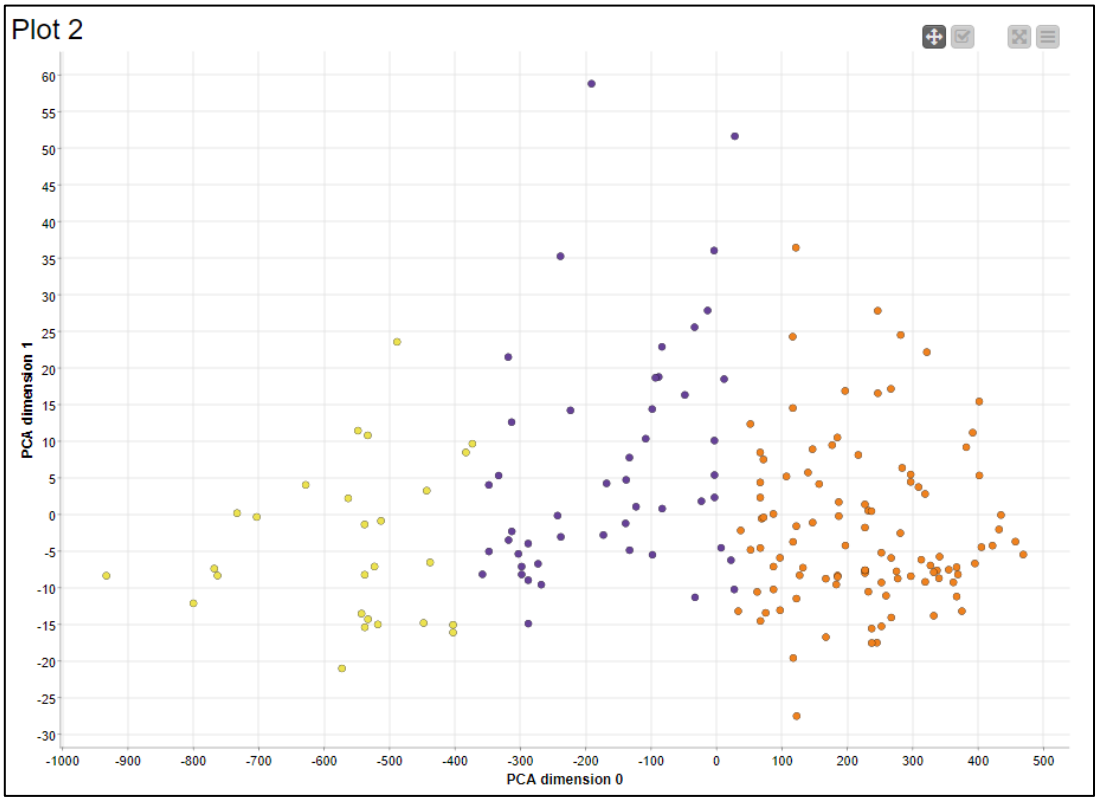
Appendix

Appendix 1.1.1



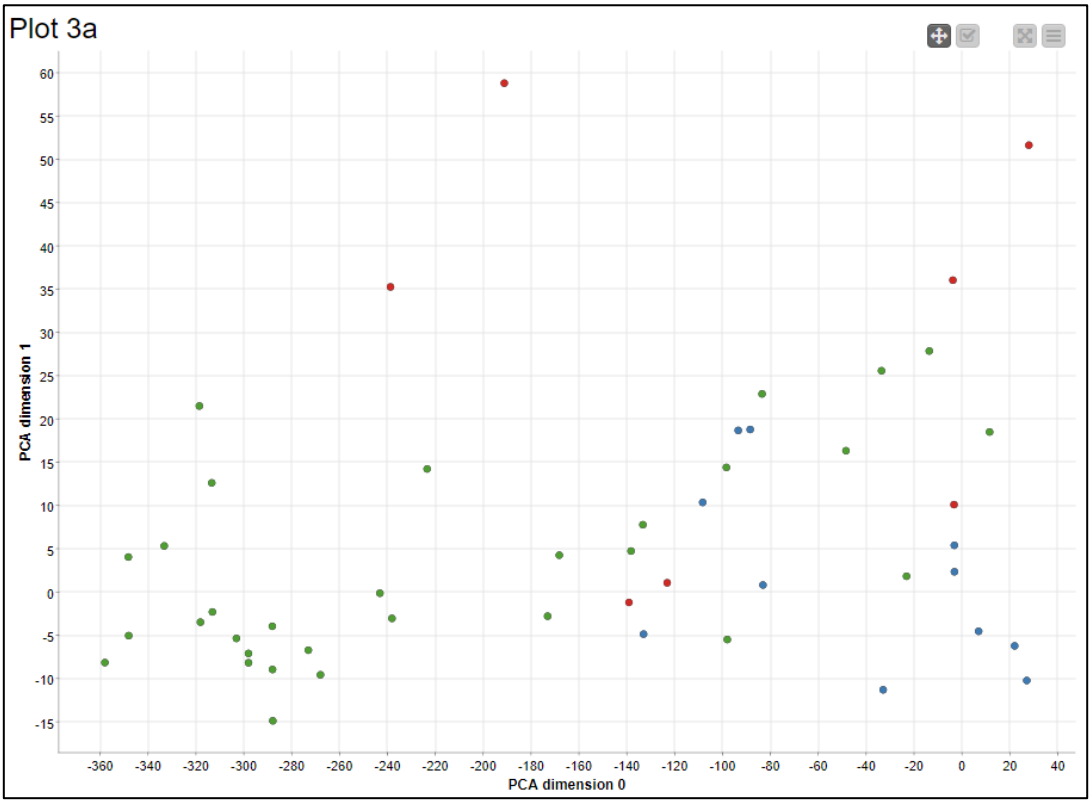
Plot 1

Appendix 1.1.2



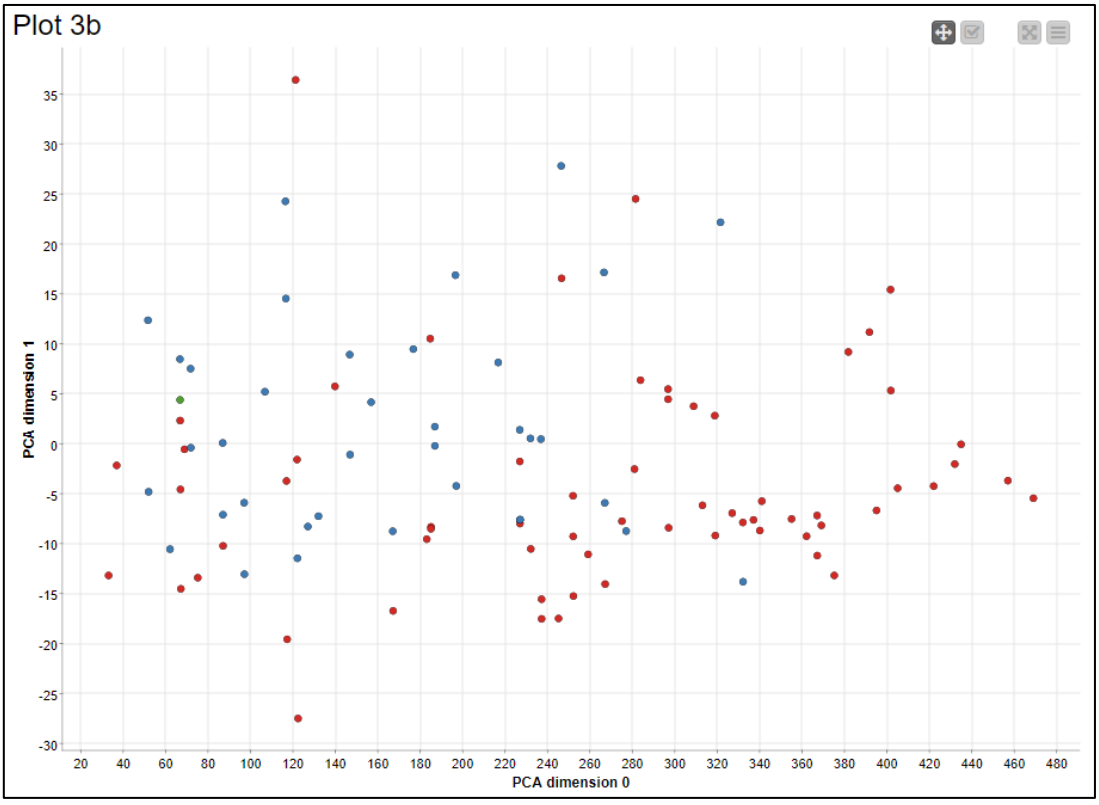
Plot 2

Appendix 1.1.3



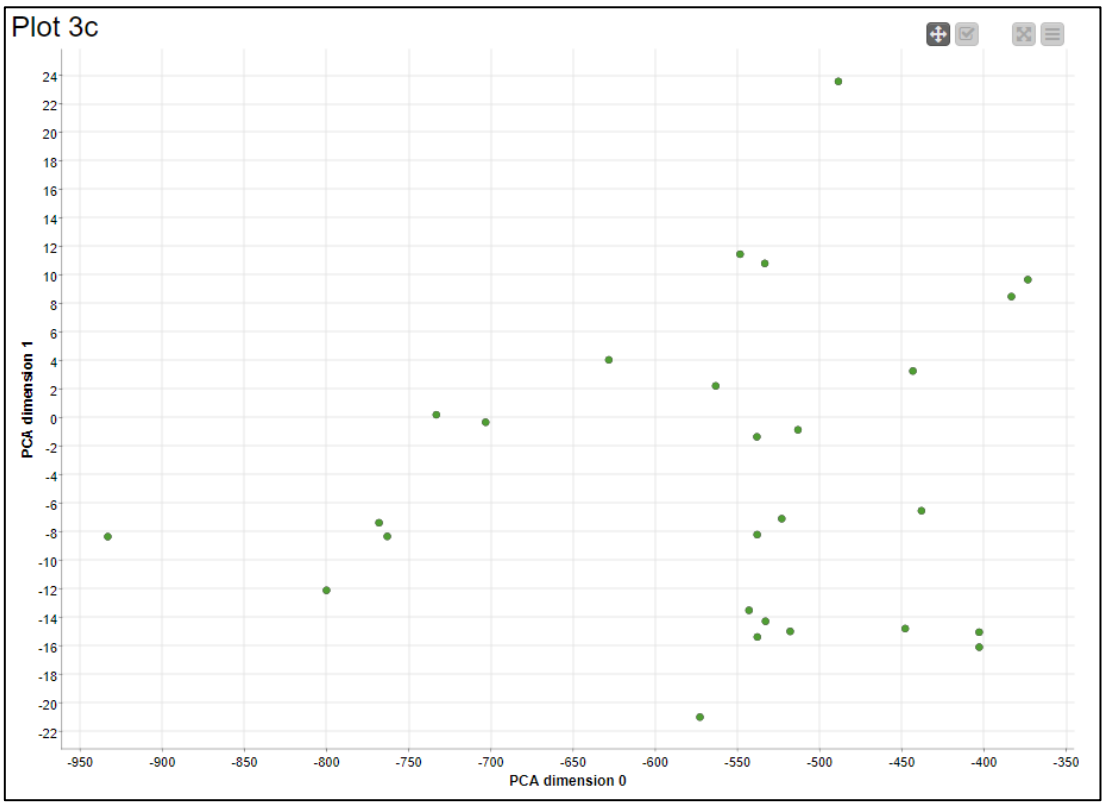
Plot 3a

Appendix 1.1.4



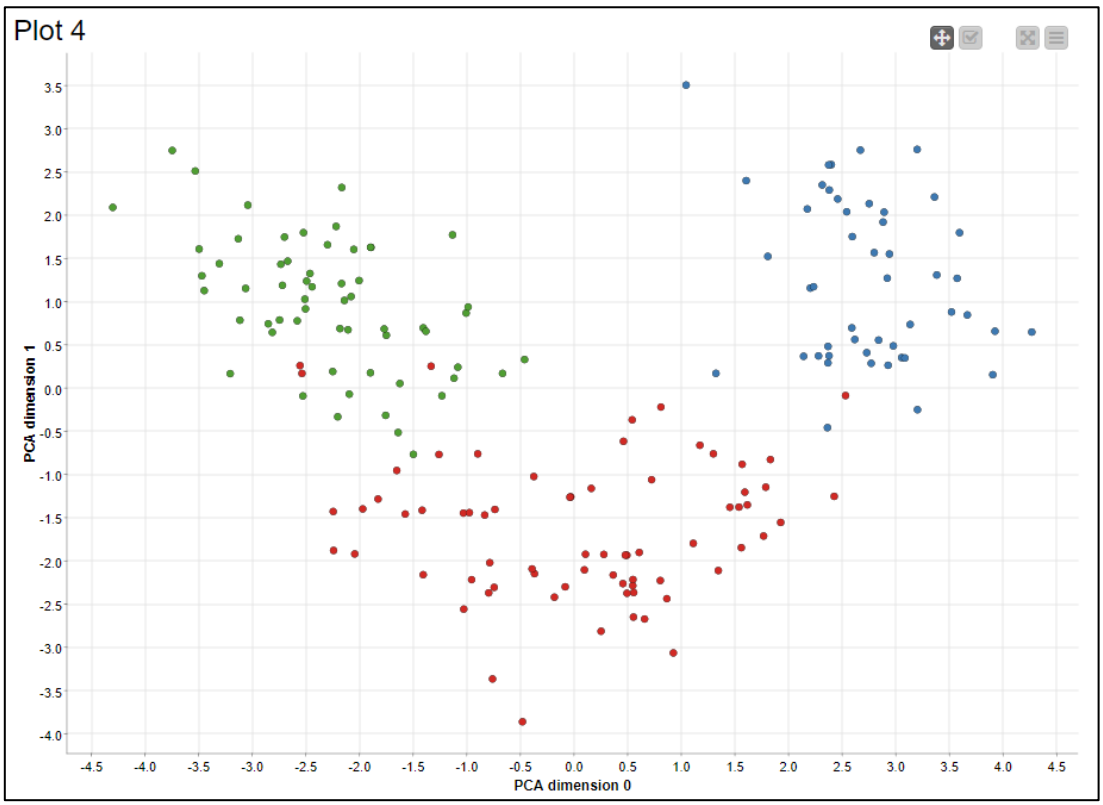
Plot 3b

Appendix 1.1.5



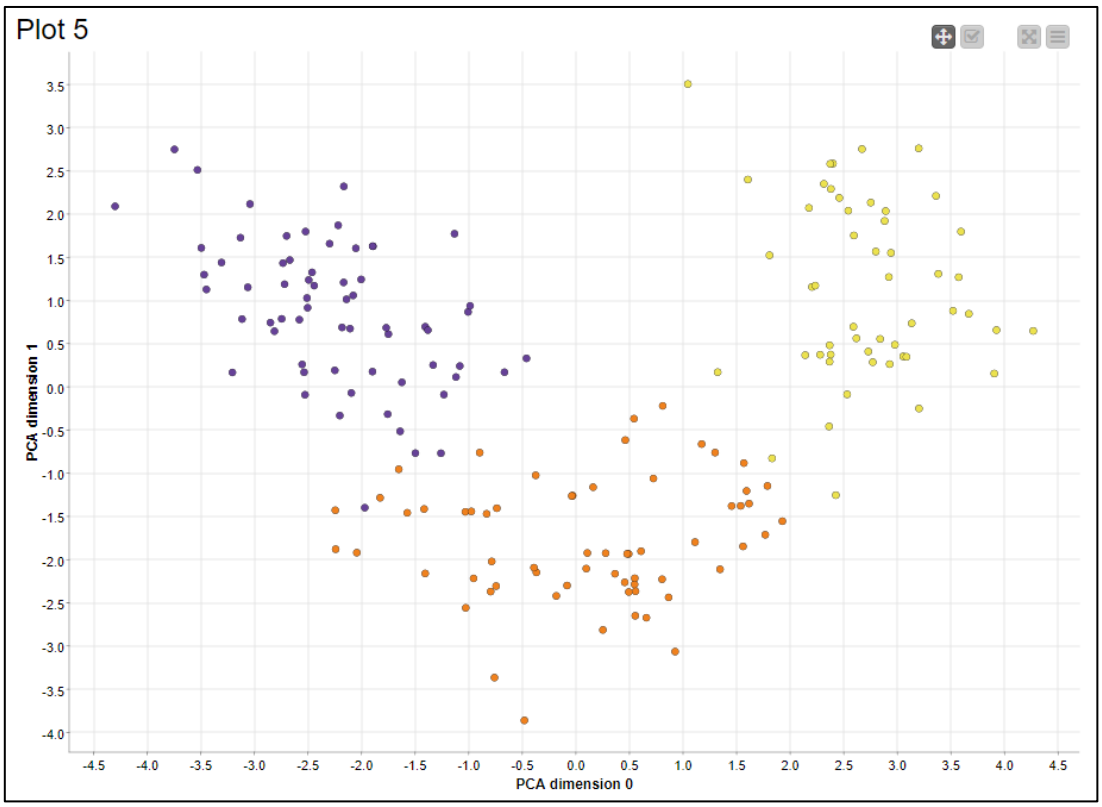
Plot 3c

Appendix 1.2.1



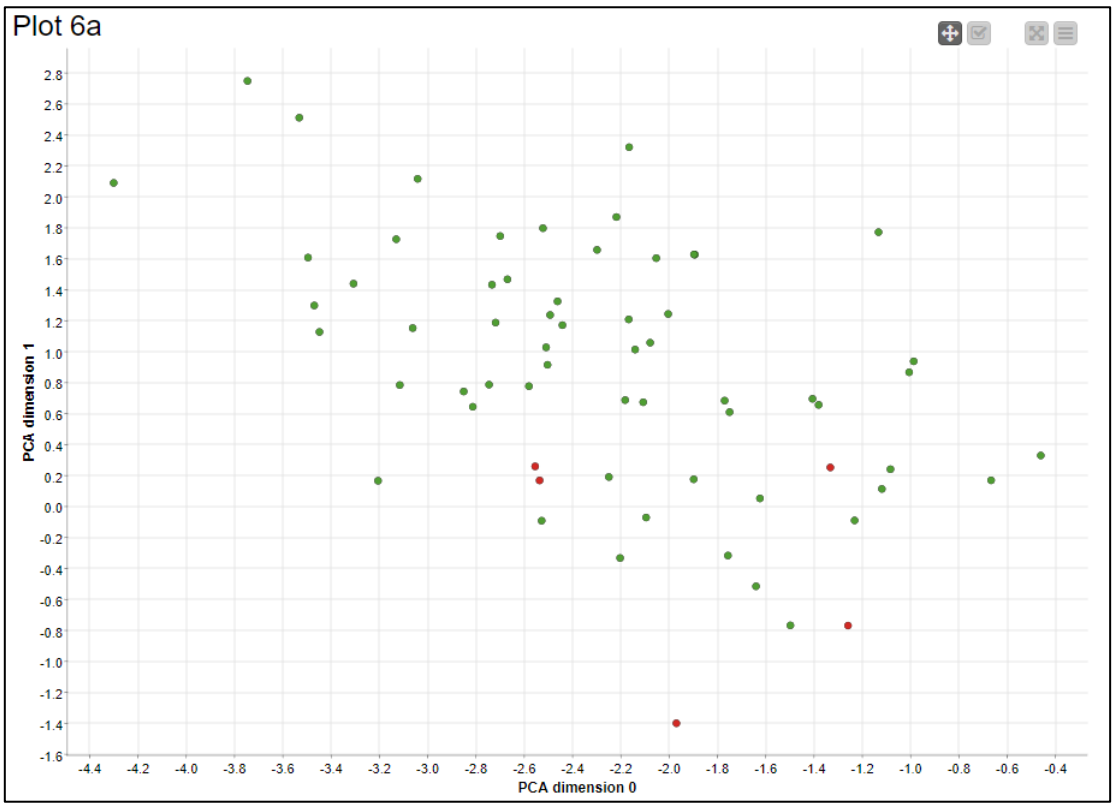
Plot 4

Appendix 1.2.2



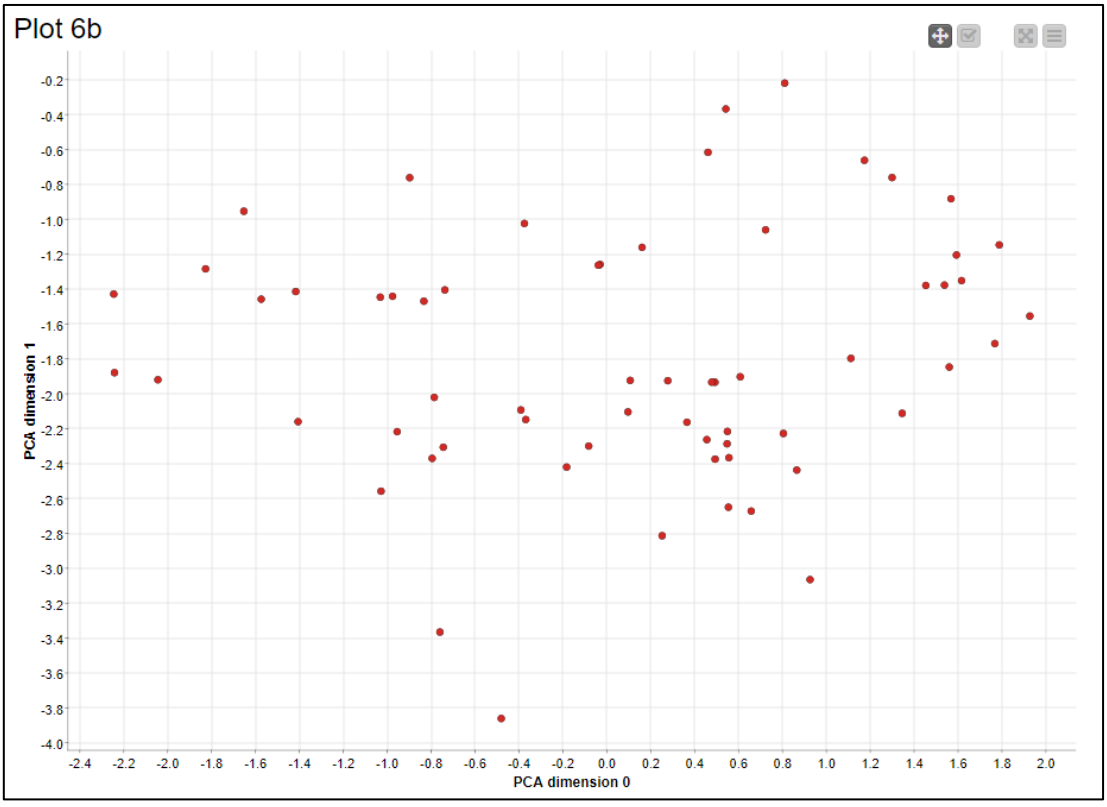
Plot 5

Appendix 1.2.3



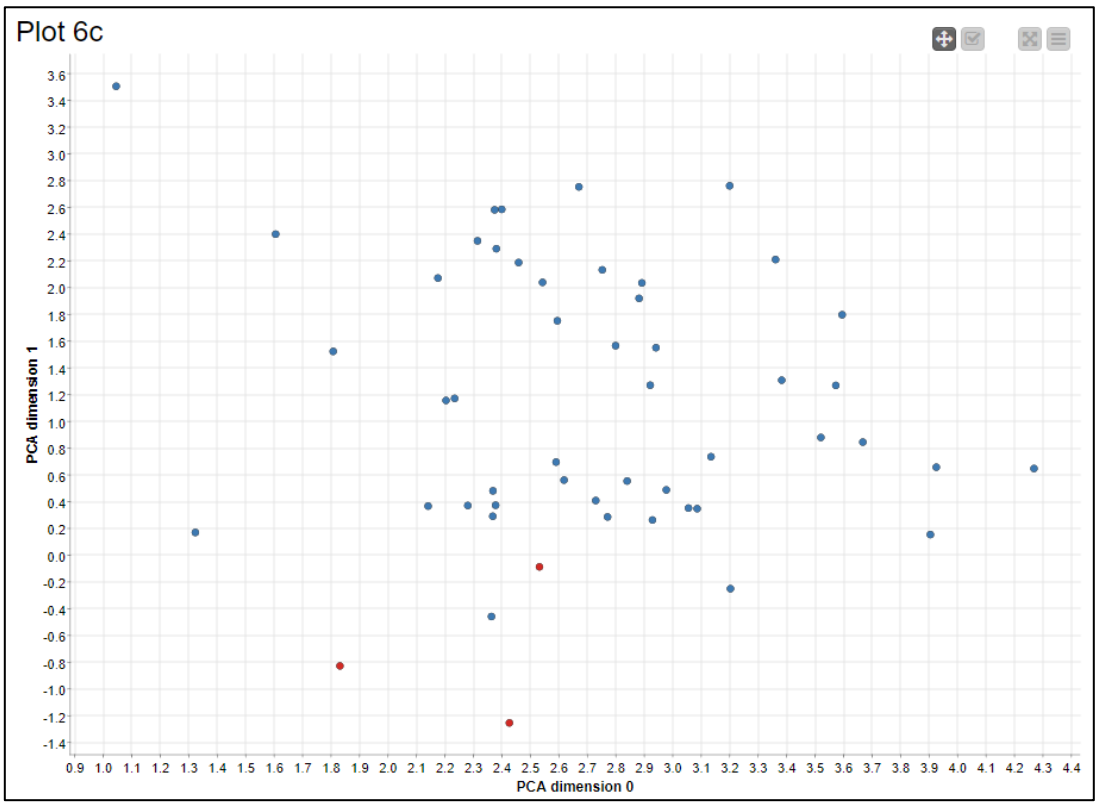
Plot 6a

Appendix 1.2.4



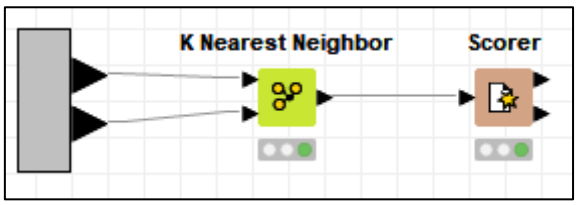
Plot 6b

Appendix 1.2.5



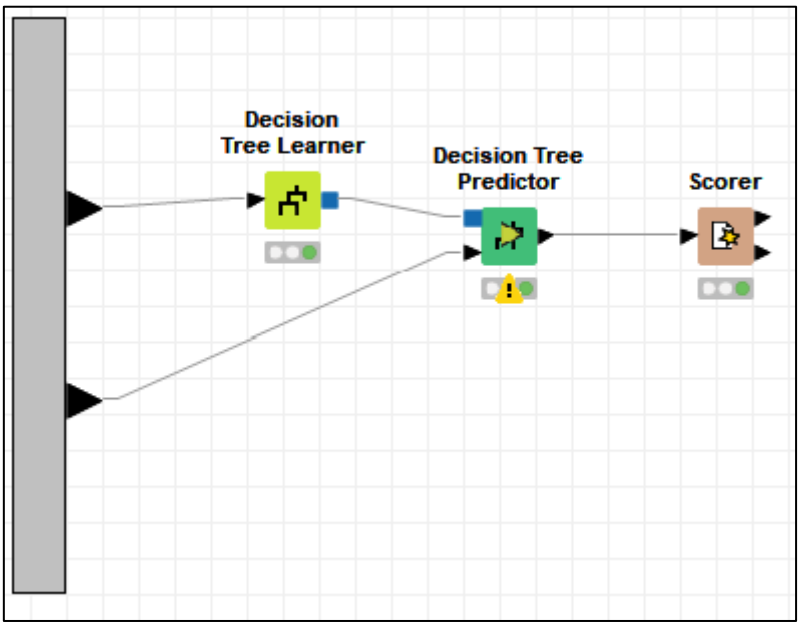
Plot 6b

Appendix 2.1



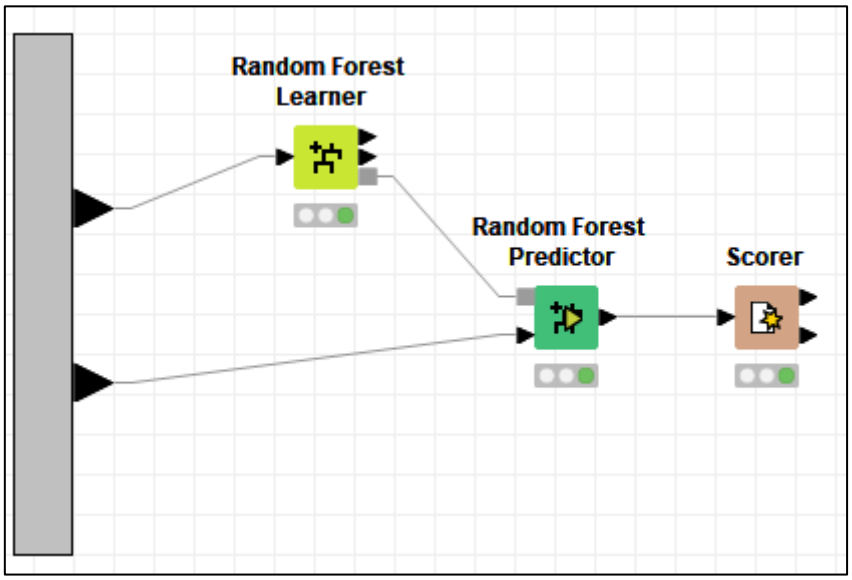
KNN Meta Node

Appendix 2.2



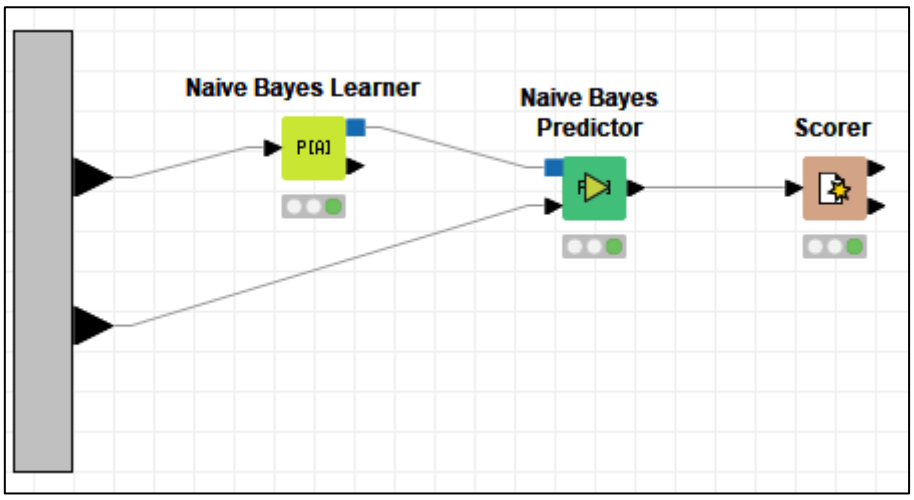
Decision Tree Learner Meta Node

Appendix 2.3



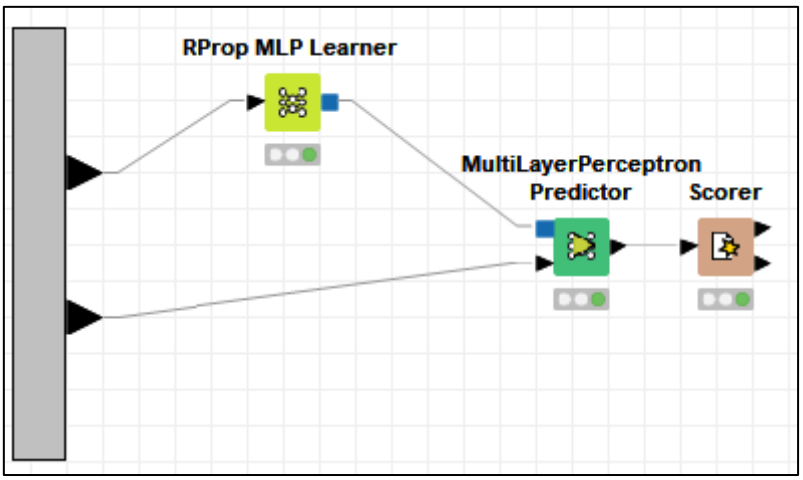
Random Forest Meta Node

Appendix 2.4



Naive Bayes Meta Node

Appendix 2.5



MLP Meta Node