DATA MINING

ASSESSMENT TASK 3

Abstract

This assignment is a practical data analytics project that follows on from the data exploration you did in Assignment 2.

Michelle Tanoto

13175144

Table of Contents

Data Mining

Problem

The assignment requires us to make a prediction on an insurance data on whether a customer purchased a policy from an insurance company. The attribute that is tasked to predict is **Quote_Flag** which is a binary attribute in the dataset with '1' indicating that a customer purchases a policy and '0' for not purchasing it. The dataset contains 29 independent variables and 1 target variable. Although, there is quite a lot of independent variables, due to privacy issues the variable names are anonymised therefore makes it harder to understand the context and meaning. The tools used to perform the data mining task on this dataset are KNIME and Python.

Inputs

There are three datasets provided for the assignment, which consist of training, unknown and Kaggle sample data. The training data is used to train the classifiers, the unknown data is used for the classifier to predict the target attribute and Kaggle sample data is used as a sample for the formatting requirements when submitting the prediction data to Kaggle.

Outputs

The task requires the submission of the prediction data to Kaggle. The prediction data is generated by evaluating the unknown data to the classifier and only include Quote_id and the predicted **Quote_Flag** attribute on the file.

Data Pre-processing

Initial Workflow

Data pre-processing is an important step in the data mining process. This section will outline the steps, including data quality assessment, partition, feature sampling and dimensionality reduction. Figure 1 shows the initial data pre-processing procedure that is done to observe its performance. Later, it will be shown which option yield the best accuracy.

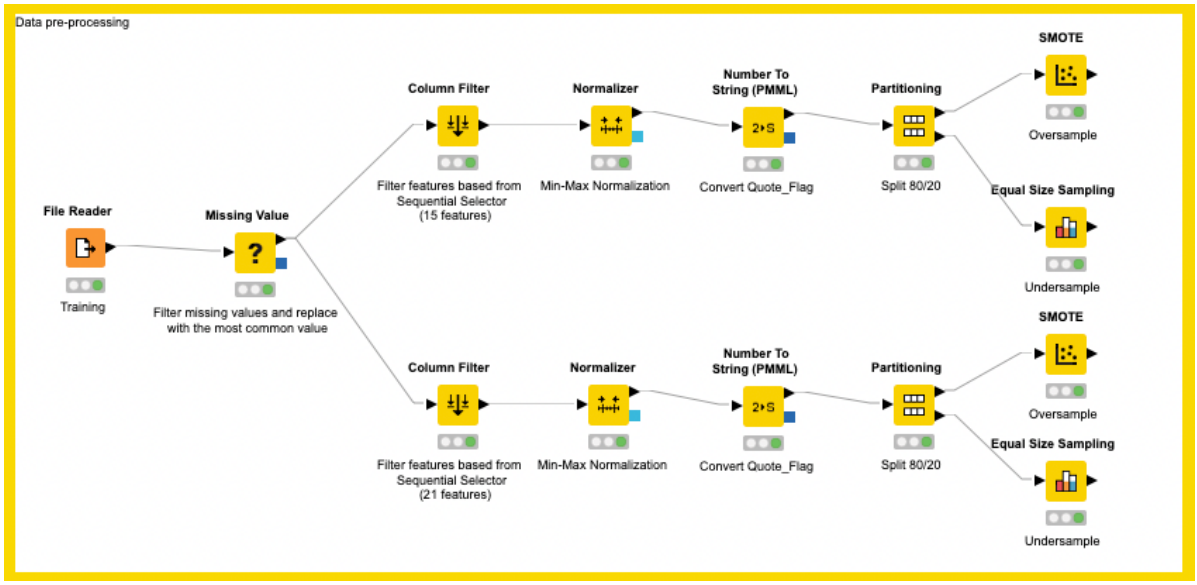

Figure 1 Initial Workflow for Data Pre-processing

Data Quality Assessment

Missing Value

Based on the Missing Value node warning, Personal_info5 has a lot of missing values, therefore it is decided to replace the missing values with the most frequent values of the attribute which is '2'.

Normaliser

The normalizer was used to normalise the integer values with min-max normalisation to bring all variables in the same range.

Number to String

Some of the classifiers requires the target attribute to be String type and the dataset has **Quote_Flag** as an Integer. This node is used to convert **Quote_Flag** from Integer to String type.

Dimensionality Reduction

Column Filter

The column filter is used to eliminate redundant attributes, as to avoid the curse of dimensionality which is caused by having too many attributes. In addition, it could also cause overfitting in the model which then result in terrible out of sample performance. Based on Figure 1, there are two column filters with 15 attributes and 21 attributes. These combinations of attributes are selected with a heuristic search method, Sequential Forward Selection.

Sequential Forward Selection method starts with selecting the best single attribute in relation to the target attribute, then pair the attribute with other attribute that yield the

best accuracy and the procedure continues until predefined number of attributes that are selected. In this case, 15 attributes and 21 attributes are selected due to both having the highest accuracy between its group. Based on figure 2, starting from left, it represents number of attributes, the accuracy score of the sets of attributes, the column number of the corresponding attributes.

```
-- GROUP 1 --
Features: 10/10 -- score: 0.8445589425086505[3, 6, 7, 8, 13, 14, 15, 17, 21, 25]
Features: 11/11 -- score: 0.8044702725656517[3, 5, 7, 12, 14, 15, 16, 17, 23, 24, 25]
Features: 12/12 -- score: 0.8363095089305557[3, 7, 8, 12, 13, 14, 15, 16, 17, 23, 24, 25]
Features: 13/13 -- score: 0.8405240939089355[0, 3, 6, 7, 8, 12, 13, 14, 15, 16, 17, 23, 24]
Features: 14/14 -- score: 0.8374480727424697[0, 3, 6, 7, 8, 12, 13, 14, 15, 16, 17, 19, 23, 24]
Features: 15/15 -- score: 0.8443393139199467[0, 3, 6, 7, 8, 9, 12, 13, 14, 15, 16, 17, 21, 23, 24]
Features: 16/16 -- score: 0.8445591220519229[3, 5, 6, 7, 8, 9, 12, 13, 14, 15, 16, 17, 21, 23, 24, 25]
Features: 17/17 -- score: 0.8443593138816524[0, 3, 6, 7, 8, 9, 12, 13, 14, 15, 16, 17, 20, 21, 23, 24, 25]

-- GROUP 2 --
Features: 18/18 -- score: 0.8543265044060548[0, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 20, 21, 23, 24]
Features: 19/19 -- score: 0.8561440437427235[0, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 20, 21, 23, 24]
Features: 20/20 -- score: 0.8555847448660165[3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25]
Features: 21/21 -- score: 0.8570629590154416[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 19, 20, 21, 22, 24]
Features: 22/22 -- score: 0.8563238279992221[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25]
Features: 23/23 -- score: 0.8566434384712542[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
Features: 24/24 -- score: 0.8569231407763598[0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25]
Features: 25/25 -- score: 0.8568232606341262[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
```

Figure 2 Results of Sequential Forward Selector with Python

To generate the results shown in Figure 2, Python is used with numpy, pandas, sklearn and mlxtend libraries as shown below with a detailed step to step explanation.

```
# import libraries
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from mlxtend.feature_selection import SequentialFeatureSelector as sfs

# read data
df = pd.read_csv('Assignment3-TrainingData.csv')

# encode data to string
https://stackoverflow.com/questions/46500357/valueerror-could-not-convert-
string-to-float-med
le = preprocessing.LabelEncoder()
balance_data = df.apply(le.fit_transform)

# split to independent columns (use to predict target) and target column
(want to predict)
x = balance_data.iloc[:, 0:26].values  # independent columns
y = balance_data.iloc[:, -1].values    # target column

# train/test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# outline the Quote_Flag train and test data
y_train = y_train.ravel()
y_test = y_test.ravel()

print('Training dataset shape:', x_train.shape, y_train.shape)
print('Testing dataset shape:', x_test.shape, y_test.shape)

# build RF classifier to use in feature selection
clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
```

```
# Iterate on 10 - 26 features
for i in range(10, 26):
    # Build step forward feature selection
    sfs1 = sfs(clf,
               k_features=i,
               forward=True,
               floating=False,
               verbose=2,
               scoring='accuracy',
               cv=5)

    # Perform SFS
    sfs1 = sfs1.fit(x_train, y_train)
    feat_cols = list(sfs1.k_feature_idx_)
    # print number of features, accuracy score and features subsets
    print(feat_cols)
```

Based on the Kaggle and KNIME performance of both option with different classifiers, it is concluded that the 15 attributes option has higher accuracy. The 15 attributes consist of:

- Field_info3 (String)
- Field_info4 (String)
- Coverage_info3 (String)
- Sales_info1 (Integer)
- Sales_info2 (Integer)
- Sales_info3 (Integer)
- Personal_info1 (String)
- Personal_info2 (Integer)
- Personal_info3 (String)
- Personal_info4 (Integer)
- Property_info2 (Integer)
- Property_info4 (Integer)
- Geographic_info3 (Integer)
- Geographic_info4 (String)
- Geographic_info5 (String)

Partition

The dataset is partitioned to 80% of training dataset and 20% test dataset with draw randomly option.

Feature Sampling

SMOTE & Equal Size Sampling

Based on figure 1, there are two sampling method that is used, oversampling with SMOTE and undersampling with Equal Size Sampling. SMOTE node is used to oversample the training subset of the datasets by **Quote_Flag** minority class, with 5 as the nearest neighbour predefined parameter. Equal Size Sampling is used to undersample the training subset of the datasets by **Quote_Flag** attribute using

approximate sampling option, since exact sampling option is more suitable to smaller dataset with less attributes based from KNIME documentation.

Based on the Kaggle and KNIME performance of both option with different classifiers, it is concluded that undersampling is better for the model due to its performance with different classifiers are slightly higher than the oversampled ones. In addition, the data is highly unbalanced. Based on the table below, class “0” has four times more distribution than “1” class, to oversample it means there is too much synthetic data, therefore undersampling is chosen for the sampling method.

Quote_Flag	Frequency	Distribution
0	50625	80.90%
1	11955	19.10%
Total	62580	100%

Final Workflow

Figure 3 shows the final workflow of the data pre-processing step, it is decided that the model uses 15 attributes, and the dataset will be undersampled.

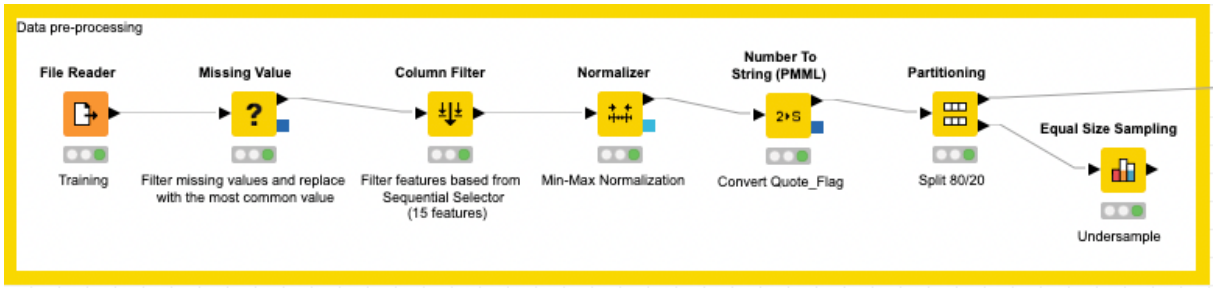

Figure 3 Final Workflow of Data Pre-processing

Problem Solving

Firstly, the problem is defined along with the expected output. Then, data exploration is performed to gain better understanding of the dataset. The exploration is done by looking through the KNIME workflow and the report from Assignment 2. Thus, any correlation and interesting attributes are found right away.

Secondly, data pre-processing is performed to the dataset. The data pre-processing done includes 6 steps, as mentioned on the data pre-processing section. Data cleaning is performed first to remove the missing value, then feature selection is performed. Due to the attribute names being anonymised, it is hard to determine the context of it, therefore it is difficult to do feature selection manually. Therefore, I research on the methods used for feature selection and there are several of them, namely Feature Importance, Univariate Selection and Sequential Selection. After running through all the methods, Sequential Selection is proven to be the best for this case. Then, the dataset is normalised and transformed to the expected input for each classifier.

Thirdly, the data is partitioned for the classifier to train and test. Then, the data is undersampled due to it being highly unbalanced to avoid biased prediction. After that,

various classifiers are built to compare its accuracy, specificity, and sensitivity with confusion matrix along with ROC curve. Then, as the best model is submitted to Kaggle the accuracy does not always match the one shown in KNIME. Thus, iterating and observing on different model performance in Kaggle and KNIME to find the best classification model for the dataset.

Classifiers Used

Random Forest

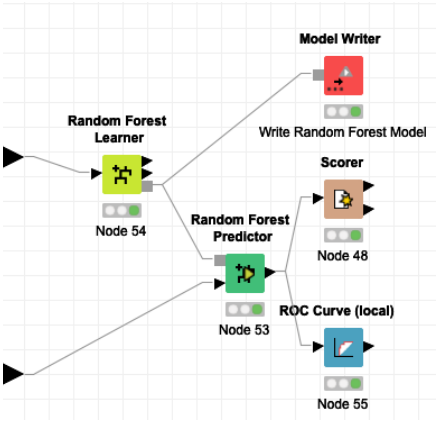

Figure 4 Random Forest Classifier

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
0	8006	363	1976	2171	0.787	0.957	0.787	0.845	0.863	?	?
1	1976	2171	8006	363	0.845	0.476	0.845	0.787	0.609	?	?
Overall	?	?	?	?	?	?	?	?	?	0.798	0.487

Figure 5 Scorer accuracy statistics

The pre-processed data acts as an input to the Random Forest classifier as shown in figure 4. The settings used for the Random Forest Learner are the default settings, with 100 number of models and using static random seed. All the settings for split criterion are tested and Information Gain (81.50% accuracy) has slightly higher accuracy than Gini Index (81.15% accuracy) and Information Gain Ratio (80.17% accuracy) in KNIME. Therefore, the split criterion is decided to be Information Gain. Figure 5 shows the accuracy statistics from the Scorer node. In addition, The AUC value is 0.879 in KNIME and 0.81613 in Kaggle.

Decision Tree

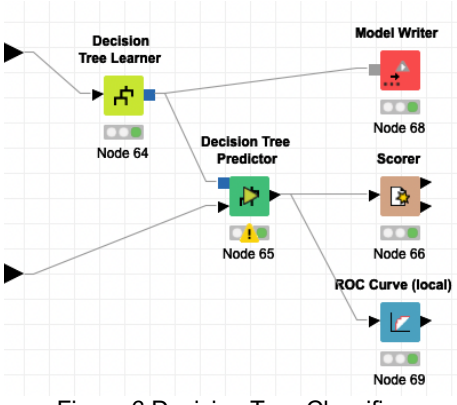

Figure 6 Decision Tree Classifier

Row ID	TruePositives	FalsePositi...	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
0	7515	331	2008	2662	0.738	0.958	0.738	0.858	0.834	?	?
1	2008	2662	7515	331	0.858	0.43	0.858	0.738	0.573	?	?
Overall	?	?	?	?	?	?	?	?	?	0.761	0.431

Figure 7 Scorer accuracy statistics

The pre-processed data acts as an input to the Decision Tree classifier as shown in figure 6. The settings used for the Decision Tree Learner are the default settings, with quality measure as Gini Index and pruning Method as MDL. The quality measure and pruning method are determined based from the confusion matrix accuracy as follows:

- Gain Ratio, No Pruning (77.182% accuracy)
- Gain Ratio, MDL (76.526% accuracy)
- Gini Index, No Pruning (77.065% accuracy)
- Gini Index, MDL (77.798% accuracy)

The highest accuracy is applied to the settings. Figure 7 shows the accuracy statistics from the Scorer node. In addition, the AUC value is 0.857 in KNIME and 0.80281 in Kaggle.

K Nearest Neighbour

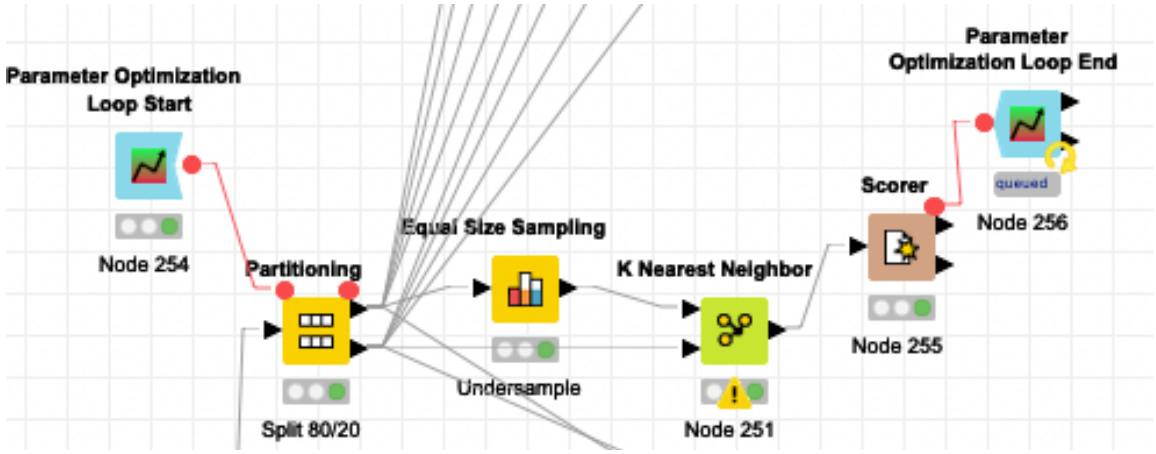

Figure 8 K Nearest Neighbour Classifier

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
0	7784	433	1906	2393	0.765	0.947	0.765	0.815	0.846	?	?
1	1906	2393	7784	433	0.815	0.443	0.815	0.765	0.574	?	?
Overall	?	?	?	?	?	?	?	?	?	0.774	0.438

Figure 9 Scorer accuracy statistics

The pre-processed data acts as an input to the K Nearest Neighbour classifier as shown in figure 8. The number of neighbours (k) is determined by the Parameter Optimization node, with start value of 1 and stop value of 15 with step size as 1 using brute force search strategy with maximised accuracy as the objective function value. It was later found that 3 is the best parameter for k. Figure 9 shows the accuracy statistics from the Scorer node. KNIME yield the accuracy of 0.7238%. Due to the low accuracy level, it is decided not to submit the model to Kaggle.

Multilayer Perceptron

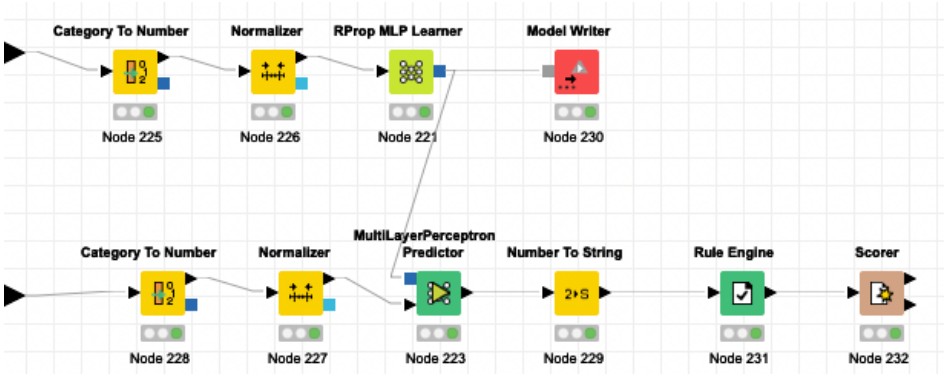

Figure 10 Multilayer Perceptron Classifier

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
0	4476	263	2076	5701	0.44	0.945	0.44	0.888	0.6	?	?
1	2076	5701	4476	263	0.888	0.267	0.888	0.44	0.41	?	?
Overall	?	?	?	?	?	?	?	?	?	0.523	0.173

Figure 11 Scorer accuracy statistics

The node used for this method are RProp MLP Learner as shown in figure 10 and there are extra data pre-processing steps need to be done as the node requires the data to be on Double type. Therefore, firstly String attribute type are transformed to Integer with Category to Number node and then all values are normalised. The RProp MLP Learner is set to use 2 hidden layers and 10 hidden neurons per layer with 100 iterations. As the MultiLayerPerceptron Predictor returns a double value for the **Quote_Flag** prediction, it is transformed to String type and applied a rule such as for example, if the **Quote_Flag** value is more than 0.5 it is treated as 1 and vice versa. Several parameters are tried, namely 0.1, 0.3, 0.5, 0.7 and 0.9. Figure 11 shows the accuracy statistics from the Scorer. The highest accuracy from the Scorer node is 52.349% with 0.3 as parameter therefore, it is not submitted to Kaggle due to its low accuracy and the datasets provided is believed to be unsuitable for the classifier.

Support Vector Machine

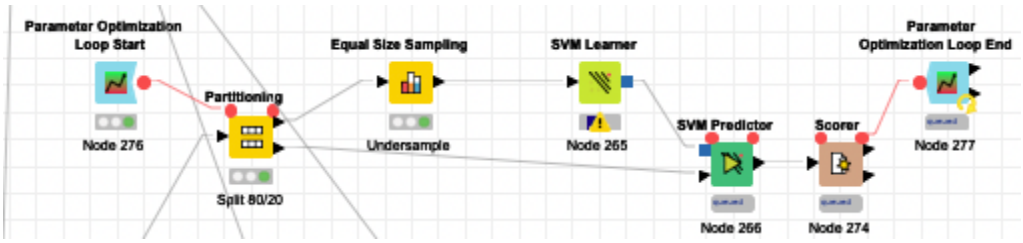

Figure 12 Support Vector Machine Classifier

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
0	7608	889	1476	2543	0.749	0.895	0.749	0.624	0.816	?	?
1	1476	2543	7608	889	0.624	0.367	0.624	0.749	0.462	?	?
Overall	?	?	?	?	?	?	?	?	?	0.726	0.295

Figure 13 Scorer accuracy statistics

The pre-processed data acts as an input to the Support Vector Machine (SVM) classifier as shown in figure 12. After two days, there is still no result which is likely to be caused by the parameter optimisation. The SVM Learner is set to use the default settings, with overlapping penalty of 1 and Polynomial kernel. The value of Power, Bias and Gamma is determined by the parameter optimisation with start value of 1, stop value of 30 and step size as 1 using brute force search strategy with maximised accuracy as the objective function value. This classifier took too many computing

resources and time; therefore, it is believed to not be suitable for big datasets. In the end, the parameter optimisation node is stopped, and it is decided to use default parameters. figure 13 shows the accuracy statistics from the Scorer. KNIME yield 72.579% accuracy, and this is the worst classifier compared to the rest.

Tree Ensemble

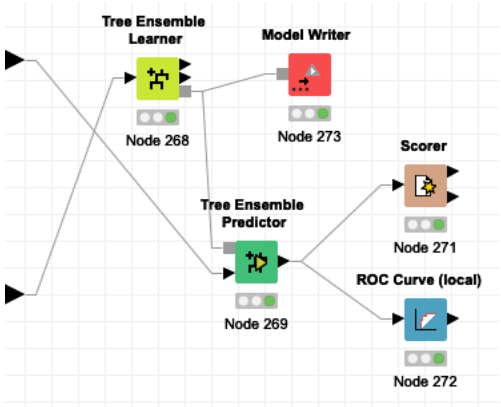

Figure 14 Tree Ensemble Classifier

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
0	7896	349	1990	2281	0.776	0.958	0.776	0.851	0.857	?	?
1	1990	2281	7896	349	0.851	0.466	0.851	0.776	0.602	?	?
Overall	?	?	?	?	?	?	?	?	?	0.79	0.475

Figure 15 Scorer accuracy statistics

The pre-processed data acts as an input to the Tree Ensemble classifier as shown in figure 14. The settings used for Tree Ensemble Learner is the default settings as there is not much option is provided. The model is similar with the Random Forest classifier as Random Forest uses an ensemble learning method and the AUC value are also close to each other although Random Forest is higher. Figure 15 shows the accuracy statistics from the Scorer. The AUC value is 0.873 in KNIME and 0.81317 in Kaggle.

Best Classifier

Through testing and observing different classifiers performance with KNIME and Kaggle, the best results are produced by the Random Forest classifier with Kaggle AUC result of 0.81613. Random Forest classifier works by constructing a multitude of decision trees and the output the average prediction of the individual class. In addition, it also overcome overfitting tendency from Decision Tree classifier. The significant process for this classifier is it uses only 15 features of the datasets and the data is undersampled thus, there should not be any bias. The Random Forest classifier also has the highest AUC value (0.879) compared to the rest.