DECISION TREE

To produce decision rules from Decision Tree, the idea is quite intuitive, we must follow the path leading to each leaf beginning from the root, which is the first data splitting of the process. In our case, we will use the algorithm for a classification problem (i.e., fraud or non-fraud), so we will focus on parameters that can fit a classification tree. To build the Decision Tree, we use the SAS's PROC HPSPLIT, enabling to build customizable Decision Trees with the high-performance (HP) statistics procedure giving better computing performances when there is a large number of observations and variables.

Decision Tree algorithm works by recursively partitioning the data into subsets based on the values of the features. At each node, the algorithm selects the best feature and a corresponding split point that optimally divides the data into subsets with high homogeneity in terms of the target variable. The homogeneity or impurity of the subsets is typically measured using metrics like Gini impurity, entropy, or mean squared error. For our experiment we choose to use the Gini impurity criterion. To quantify the degree of impurity, the idea is to calculate the probability of misclassifying a randomly chosen element if it were randomly labeled according to the distribution of classes in the node. A low Gini impurity indicates a more homogeneous node, while a high Gini impurity suggests a more mixed node. The decision tree algorithm seeks to minimize the Gini impurity when selecting the best split to create more pure and discriminative subsets during the tree growing process.

The splitting process continues until a stopping condition is met (for example, a maximum tree depth) or when further splits do not significantly improve the Gini impurity of the subsets. For rules generation, it is important to make sure that the splitting process is not too long, resulting in rules with many conditions that may overfit the training data, being unable to detect fraud cases in new data. For this, we ensure that a maximum number of splits is set. We decide to limit the depth of the tree to a maximum of 8 levels.

The number of branches per node (i.e., child nodes) is also an important parameter to determine. We decided to set it at maximum 10. Resulting in a splitting's stop once it reaches a total of 10 branches from a single parent node. A small number of branches can result in an oversimplified model that is inefficient to underly patterns in the data. On the contrary, a too high number of trees can result in a complex tree that may overfit the training data.

Another way to prevent the model from overfitting and to simplify the tree structure to remove the superfluous branches is to perform a pruning. We used the C4.5 pruning method which is a pessimistic approach. The idea behind pessimistic pruning is to consider the worst-case scenario when deciding to prune a subtree. It assumes that the observed error rate on the training data is an optimistic estimate, and the true error rate on unseen data might be higher. By using a pessimistic estimation of the error, the pruning process becomes more conservative and ensures that only those subtrees with a higher

certainty of being overfit are removed. HPSPLIT uses a variant of C4.5, it looks at the beta distribution rather than the binomial distribution for estimating the upper confidence limit.

The appropriate parameters for pruning and growing methods or maximum number of branches must be determined through experimentation and cross-validation to find a good balance between model complexity and predictive performance.

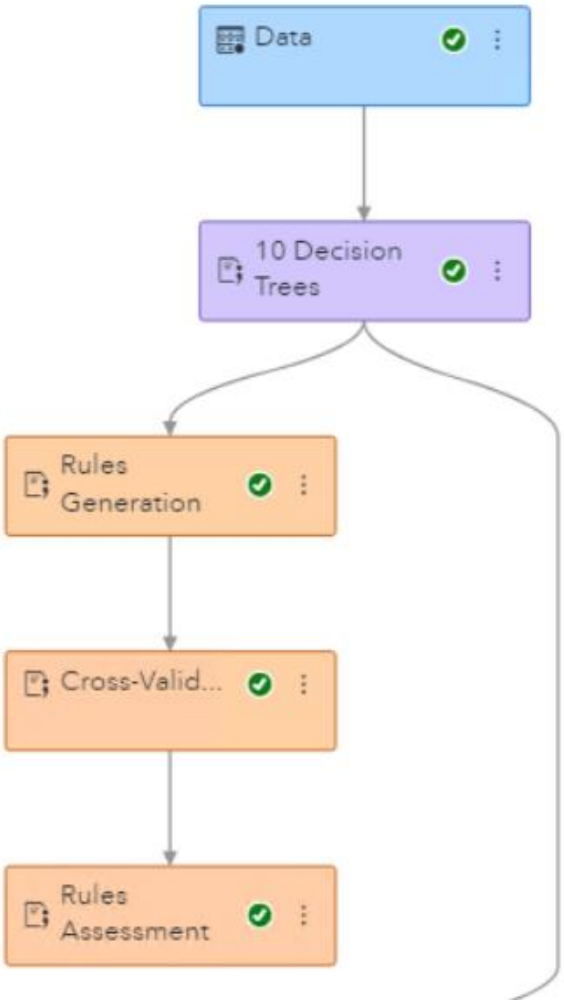

Fig.3., Pipeline for Decision Tree with cross-validation.

Cross-validation process (Fig.4.) for rules generation consist in creating K Decision Trees with K the number of folds we want to use for the model calibration. To create the folds, we split the training data into the desired number of sets. In our case, we perform a 10-folds cross-validation. We use a random seed with the `ranuni` function to generate a random number for each observation and thus, based on this number, generate samples of approximately equivalent size (Code 2.).

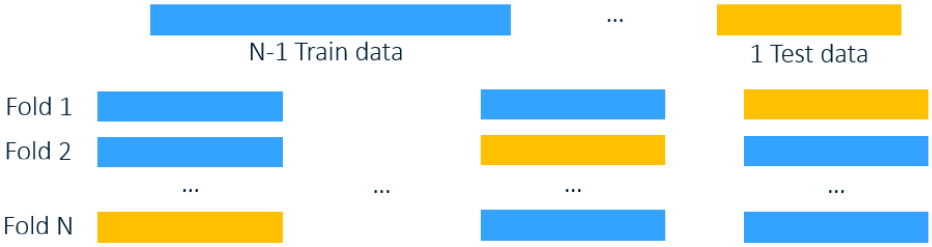

Fig.4., Cross-Validate rules to find a good set of parameters and reduce the risk of overfitting.

```
data fold1 fold2 fold3 fold4 fold5 fold6 fold7 fold8 fold9 fold10;
    set &dm_data;
    r=ranuni(998789);
    if r<0.1 then output fold1;
    else if r<0.2 then output fold2;
    else if r<0.3 then output fold3;
    else if r<0.4 then output fold4;
    else if r<0.5 then output fold5;
    else if r<0.6 then output fold6;
    else if r<0.7 then output fold7;
    else if r<0.8 then output fold8;
    else if r<0.9 then output fold9;
    else output fold10;
    drop r;
run;
```

Code 2., SAS code to generate the 10 folds in SAS Visual Data Mining and Machine Learning.

We train K models on K-1 folds and assess the performance over the remaining fold, so that each model learns on data with slight changes using the determined parameters in Code 3. The parameters of the implemented model were obtained after the process of cross-validation, and we selected the one that showed the best performance according to our evaluation metric. This is probably not the optimal model.

```
proc hpsplit data = outf&&i.
    maxbranch=10 minleafsize=10 intervalbins=20 mincatsize=10 maxdepth=8
    assignmissing=none nodes=detail cvmodelfit cvmethod=random(10)
    splitonce seed=88532;
    criterion gini ;
    prune C45;
    class %dm_nominal_input %dm_binary_input %dm_dec_target;
    model %dm_dec_target (event="1") = %dm_ordinal_input %dm_interval_input %dm_binary_input %dm_nominal_input;
    ods output HPSplit.ModelAssessment.NodeTable='/home/sasdemo/casuser/tree_rules/rules_outf&&i.';
run ;
```

Code 3., SAS code to generate a Decision Tree with High-Performance procedure.

As we can see, the dataset inserted into the 'data=' parameter of the model is randomized. The goal is to create an iterative loop that will select the K folds separately to train a model on K-1 folds. Therefore, 'outf1' represents the dataset with all folds combined except the first one. As mentioned earlier in this paper, we decide to set the maximum number of branch and the minimum number of observations per leaf at 10, we determine 20 possible splitting points for interval values and a maximum depth of 8 levels. We also perform a 10-folds cross-validation for the model itself (not for the rules. Here the idea is not to see if the model can create efficient rules or to select a good set of parameters but to requests model assessment and a confusion matrix with cross validation, this step is not necessary but is a good indicator for the overall performance of the Decision Tree, both on classifying genuine and fraudulent transactions. The 'class' parameter is used to tell the model what variables should be considered as nominal. Instead of using the variables' names, we call macro-variable that store our real variables as determined in the data preparation step. These macro-variables are automatically defined when the pipeline is created in VDMML. For the 'model' parameter, we must include the target variable and the event tag indicator. In our case the target variable is either fraud (1) or not (0). After the equal sign, we precise all the variables we want to include as predictor for the target. In our case, we select all the variables (both continuous and nominal). Finally, the last line of code allows us to specify where we want to save the table containing the paths created by the decision tree. In our case, we save it locally to easily reuse this table in a subsequent node.

Once we found a satisfying set of parameters (method describe below), we must get the file containing all the paths generated by the Decision Tree. In SAS Visual Data Mining and Machine Learning (VDMML), this table is named as 'HPSplit.ModelAssessment.NodeTable'. In our experiment we save that table as a local file so we can use it in another node. This table contains all the paths and their probability to belong to each class. For the model's configuration used, the model generates a set of rules that we will subject to the cross-validation process on new data to ensure the model's performance. But first, the syntax of the rules must be created so that they can be integrated into a SAS procedure. To achieve this, we create the 'rulegen' function along with its parameter 'k', which allows the creation of the syntactical form of rules derived from the k-folds, where 'k' represents the number of folds used for cross-validation.

Thus, the macro-function 'rulegen' (Code 4.) enables the conversion of the condition sequences (paths) obtained from the model's output table into logical conditions. It also ensures filtering only the paths of the decision tree that lead to a fraud probability greater than 50%. Then, we create the conditions by adding the necessary operators between each condition, ensuring that the correct operators are used for the appropriate variable types (e.g., the '=' operator should be replaced by 'IN()' when multiple nominal values form a condition).

```
%macro rulegen(k);
    %do i = 1 %to &&k;

    proc sort data=f&&i out=step1;
        by StringID;
    run;

    data step2;
        length Type $5 Condition $200;
        set step1 (rename=(StringID=Type Path=Condition));
        by Type;
        if Condition='Root Node' then delete;
        if findc(Condition, ',') > 0 then do;
        Condition = trim(tranwrd(Condition, '=', 'IN ('));
        Condition = catx('', Condition, ')');
        end;
        if findc(Condition, 'AND') >0 then do;
        Condition = trim(tranwrd(Condition, 'AND', cat('AND ', scan(Condition, 1, ' '))));
        end;
    run;


    data step3 (keep=Type Condition PFraud1 RuleID);
        set step2 (rename=(Selected2=PFraud1));
        by Type;
        retain RuleID 0;
        if first.Type then do;
            Type='IF';
            RuleID+1;
        end;
        else do;
            Type='AND';
        end;
    run;

    data step4;
       do _n_ = 1 by 1 until (last.RuleID);
         set step3;
         by RuleID;
       end;
       fraud = PFraud1 = '*';
       do _n_ = 1 to _n_;
         set step3;
         if fraud then
           output;
       end;
       drop fraud;
    run;
```

```
    proc delete data=casuser.rules_set&&i;
    run;

    data casuser.rules_set&&i (keep=Rules promote=yes replace=yes) ;
    length Rules $1000.;
       do until (last.RuleID);
          set step4;
          by RuleID ;
          Rules=catx(' AND ',Rules,Condition);
          if findc(Rules, ',...') > 0
          then Rules = tranwrd(Rules, ',...', '');
          if findc(Rules, 'or Missing') > 0
          then Rules = tranwrd(Rules, 'or Missing', '');
          if findc(Rules, ',') > 0 and
          findc(Rules, '...') > 0
          then Rules = tranwrd(Rules, '...', '');
       end;
    run;

    proc print data=casuser.rules_set&&i;
    run;
    %end;
%mend;

%rulegen(k=10)
```

Code 4., Function generating the rules.

The first input table '`f`' followed by a number corresponds to the table containing the tree paths for each model that learned on N-1 folds. Thus, the table '`f1`' corresponds to the decision tree of the model that learned on all folds except fold number 1. We promote the final table with the argument '`promote=yes`' so the table can be accessed in subsequent node. Once we created the rules, we process the cross-validation to verify the choice of the parameters and see how the model perform on the different folds. The steps of model configuration and cross-validation are the outcome of multiple trials (Fig. 5.). We build a model, evaluate its performance using cross-validation, and then repeat this process until we find a set of satisfactory parameters.

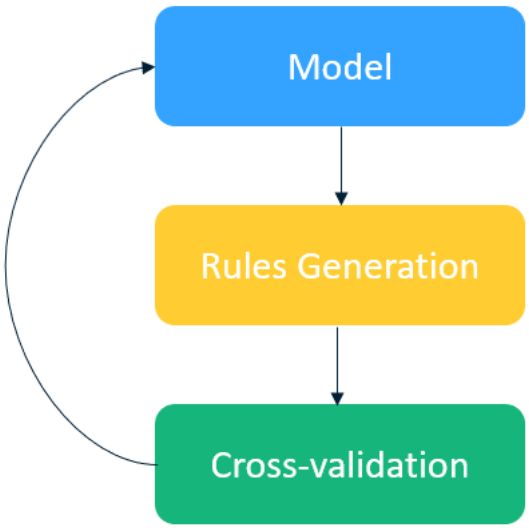

Fig. 5., Parameters choice process.

We can use the average false positive rate of a rule set as a statistic for assessing the rules created with the K-1 folds. Then we compare this metric for different model parameters until we find a set of parameters that optimize the statistic we use. In our case, we aim to minimize the average false positive rate among the rules (Table 4.). With this set of parameters, we get an average false positive rate of 38.1%.

Fold	FPR
1	57,12%
2	66,72%
3	55,69%
4	63,63%
5	56,13%
6	59,72%
7	65,59%
8	61,34%
9	73,76%
10	65,50%

Table 4., Average false positive rates for rules from each fold (table obtained from 10-folds cross validation).

We can also use the different folds to select the tree with the best evaluation statistic. In our case, for example, we will select the rules from tree 5, which has an average False Positive Rate of 56.13%. Tree 5 was trained on folds 1 to 10 excluding fold 5 and evaluated on fold 5. This false positive rate may seem quite high; however, it represents an average across all rules. It is important to note that rules with the poorest performances can be removed from the set during the individual evaluation phase, thereby reducing the risk of false positives.

The individual evaluation phase involves evaluating the rules one by one on the test data, on new data, to further reduce the number of generated rules by eliminating those that achieve limited performance on the new data. For instance, we can remove rules that do not trigger any alerts or those that have a significantly high false positive rate, detecting more legitimate transactions than fraud cases.

Table 5 contains a sample of rules generated with the Decision Tree technique, we selected the rules from the fold that perform the best, on among the validation set which is the tree 5.

Rule	DR	Frauds Detected	Amount Saved	FPR	Fraud Rate for Rule	Lift	Number of Alerts	IR	Number of Conditions
hct_term_cntry_code IN (124,196,724,752,826) AND avg_val_day_pos >= 38.729 AND avg_val_day_pos < 42.9585 AND rqo_tran_time < 01:12:04	0,59%	7	$ 74,73	0:7	100%	8,55	7	0,07%	4
hct_term_cntry_code IN (124,196,724,752,826) AND avg_val_day_pos >= 38.729 AND avg_val_day_pos < 42.9585 AND rqo_tran_time >= 16:47:57 AND rqo_tran_time < 17:59:57	0,85%	10	$ 97,23	6:10	63%	5,34	16	0,16%	5
hct_term_cntry_code IN (124,196,724,752,826) AND avg_val_day_pos >= 38.729 AND avg_val_day_pos < 42.9585 AND rqo_tran_time >= 17:59:57 AND rqo_tran_time < 19:11:56	1,44%	17	$ 324,70	5:17	77%	6,61	22	0,22%	5
hct_term_cntry_code IN (56,280,528) AND rqo_tran_time >= 17:59:57 AND rqo_tran_time < 19:11:56	0,34%	4	$ 52,30	5:4	44%	3,80	9	0,09%	3
hct_term_cntry_code IN (56,280,528) AND rqo_tran_time >= 21:35:55 AND rqo_tran_time < 22:47:55	0,51%	6	$ 41,91	3:6	67%	5,70	9	0,09%	3

Table 5., Sample of rules generated with Decision Tree.