Version	Author	Date
1.0	gandhiax2024 no email	2025-02-21 11:51:25

Table of contents

I. Executive Summary

A Binary classification Machine Learning model was built using Dataiku DSS Visual ML. Its goal is to predict given a total of features. Using a dataset of 4041 rows, the process led to the selection of the Random Forest algorithm.

A. Methodology

To ensure a good generalization capability for the ML model, a test strategy was set up. Data on which the ML model was not trained on was used for this purpose. The testing strategy was the following:

See section II.D for detailed explanations about these options.

Before being tested, the ML model has been tuned to find the best combination of hyperparameters according to the metric. This optimal hyperparameter search, based on assessing performance on a validation set, was done using the following methodology:

See section **Error! Reference source not found.** for detailed explanations about these options.

B. Results

The Random Forest algorithm was selected. The evaluation metric used to tune the hyperparameters was computed on the validation dataset. After the best hyperparameter combination was found, the same metric was also computed on the test dataset. The final value was 0.947.

II. Methodology

This section deals with the methodological details:

- The *Problem Definition* consists of selecting the target () and the type of problem (Binary classification).
- *Data Ingestion* analyzes each feature in order to maximize its prediction potential.
- *Model and Feature Tuning* finds the best hyperparameter set for the selected algorithm.
- The *Model Evaluation and Selection* strategy indicates how to compute the metrics that allows to evaluate the performance of the model.

A. Problem Definition

A Binary classification Machine-Learning model was built using Dataiku DSS. Its goal is to predict given a total of features.
The proportion of the target classes is shown on the graph below:

B. Data Ingestion

During the data ingestion phase, the features are transformed into numerical features without missing values so as to be ingestible by the Machine Learning algorithm. The table below summarizes the processing applied to each of them.

Legend

- *Feature name:* Name of the feature column
- *Feature status:* Input, Target or Rejected
- *Feature type:* Numeric, Category, Text, or Array
- *Processing:* Type of processing applied (Avg-std rescaling, dummy-encode…)

C. Model and Feature Tuning

1. Pre-processings

Once each feature has been processed, it is possible to combine them to generate new features:

- Pairwise linear feature generation:
- Pairwise polynomial feature generation (A*B) for all pairs of features:

- Explicit pairwise polynomial feature generation.
  - Two numerical features are multiplied
  - Categorical features produce dummies. Interaction with a numerical feature results in the multiplication of dummies by the numerical feature value. Interactions between two categorical features produce dummies in the cross-product space of the two features.

  The concerned feature pairs are listed below:

Legend

- *Column 1:* First feature name
- *Column 2:* Second feature name
- *Max features:* Maximum number of generated features (when the interaction occurs between features that have already been expanded with, for example, one-hot encoding).
- *Rescale:* Indicator of rescaling of generated features.

The last feature processing step consists of keeping only the most promising ones. Feature reduction operates on the preprocessed features. It allows you to reduce the dimension of the feature space in order to regularize your model or make it more interpretable. The feature reduction technique used is given in the following table:

2. Tested Algorithm

The Random Forest algorithm has been tested.

The settings for this algorithm are given below. For hyperparameters, the possible values or ranges are listed:

3. Hyperparameter Search

The hyperparameter search is done for each algorithm separately. It consists of finding the combination of hyperparameters that results in the best-trained model according to the validation metric () computed on the validation dataset.
The actual search settings for the selected algorithm are the following:

Legend

- *Randomize grid search:* If true, the grid was shuffled before the search.
- *Max number of iterations:* This parameter sets the number of points of the grid that have been evaluated.
- *Max search time:* Maximum search time in minutes.
- *Parallelism:* -1 for automatic. It sets the number of hyperparameter searches that are performed simultaneously.
- *Stratified:* If true, the same target distribution is kept in all the splits.

Illustration:

Note: A grey area appears on the graphic to illustrate the data that is used for the test dataset.

4. Weighting Strategy

5. Calibration

The current calibration strategy is “”.

> Probability calibration helps adjust the predicted probabilities to the actual class frequencies.
>
> It should only be used if the problem involves actual probabilities of events, not just the ordering of these probabilities (ranking).
>
> For instance, upon predicting some probability of the positive class occurring, non-calibrated models can underestimate or overestimate the actual frequency of the positive class occurring, which can lead to sub-optimal decisions.
>
> Calibrated models can be especially useful when the predicted probabilities are used to compute expectations of another quantity.
>
> Note that isotonic regression is more prone to overfitting and metrics alteration than Platt scaling.

6. ML Overrides

Model overrides ensure predetermined prediction outcomes based on a set of defined rules.

ML Overrides
No overrides defined in the settings

D. Evaluation and Selection

The last part of the methodology consists of comparing the performance of each algorithm trained using the best hyperparameter combination. The policy can consist in either:

- Splitting the dataset by setting apart a test dataset, also called the hold-out dataset, for this performance evaluation. The train ratio indicates the amount of the dataset used in training, the remaining being used for evaluation.

- Performing a K-fold evaluation. It allows a more precise performance evaluation, at the expense of increased computation time.

This is indicated by the policy and the split mode in the table below.

When the original dataset is very big, the required computational resources may be too large compared to the expected benefit of training algorithms on it. As a result, the training, validation, and testing may be performed on a subset of the dataset. The sampling method given in the table below defines how it is built.

Legend

- Policy:
    - *Split the dataset:* Split a subset of the dataset.
    - *Explicit extracts from the dataset:* Use two extracts from the dataset, one for the train set, one for the test set.
    - *Explicit extracts from two datasets:* Use two extracts from two different datasets, one for the train set, one for the test set.
    - *Split another dataset:* Split a subset of another dataset, compatible with the dataset.
    - *Explicit extracts from another dataset:* Use two extracts from another dataset, one for the train set, one for the test set.
- *Sampling method:* A subset may have been extracted in order to limit the computational resources required by the evaluation and selection process. The *Record limit* gives its size.
    - *No sampling (whole data)*: the complete dataset has been kept.
    - *First records*: The first N rows of the dataset have been kept (or all the dataset if it has fewer rows. The current dataset has 4041 rows). It may result in a very biased view of the dataset.
    - *Random (approx. ratio)*: Randomly selects approximately X% of the rows.
    - *Random (approx. nb. records)*: Randomly selects approximately N rows.
    - *Column values subset (approx. nb. records)*: Randomly selects a subset of values and chooses all rows with these values, in order to obtain approximately N rows. This is useful for selecting a subset of customers, for example.
    - *Class rebalance (approx. nb. records)*: Randomly selects approximately N rows, trying to rebalance equally all modalities of a column. It does not

oversample, only undersamples (so some rare modalities may remain under-represented). Rebalancing is not exact.

  - *Class rebalance (approx. ratio)*: Randomly selects approximately X% of the rows, trying to rebalance equally all modalities of a column. It does not oversample, only undersamples (so some rare modalities may remain under-represented). Rebalancing is not exact.
- Partitions:
  - *All partitions:* Use all partitions of the dataset.
  - *Select partitions:* Use an explicitly selected list of partitions.
  - *Latest partition:* Use the "latest" partition currently available in the dataset. "Latest" is only defined for single-dimension time-based partitioning.
- *Time variable:* By enabling time-based ordering, DSS checks that the train and the test sets are consistent with the time variable. Moreover, DSS guarantees that:
  - The train set is sorted according to the selected variable.
  - The hyperparameter search is done with training sets and validation sets consistent with the ordering induced by the time variable.
- *Split mode:* If "*K-fold cross-test*" is selected, it gives error margins on metrics, but strongly increases training time.
- *Train ratio:* Proportion of the sample that goes to the train set. The rest goes to the test set.
- *Number of folds:* Number of folds K to divide the dataset into.
- *Random seed:* Using a fixed random seed allows for reproducible results.

III. Experiment Results

The methodology detailed in the previous section has been run. The obtained results are presented in this section.

A. Selected Model

The optimal set of hyperparameters for the selected algorithm Random Forest is given in the table below:

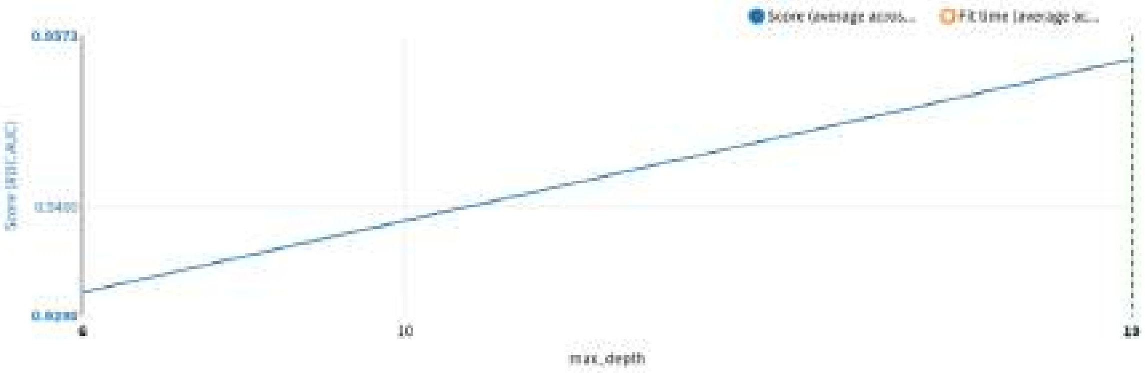
Algorithm	Random forest classification	Split quality criterion	Gini

Number of trees	100	Use bootstrap	Yes
Max trees depth	19	Feature sampling strategy	sqrt
Min samples per leaf	1		
Min samples to split	3		

See section **Error! Reference source not found.** for detailed explanations on the algorithm and its hyperparameters.

B. Alternative Models

For the selected algorithm, the following other hyperparameter combinations were tried and led to lower performance. As an example, the plot below shows the evolution of the performance for each hyperparameter:



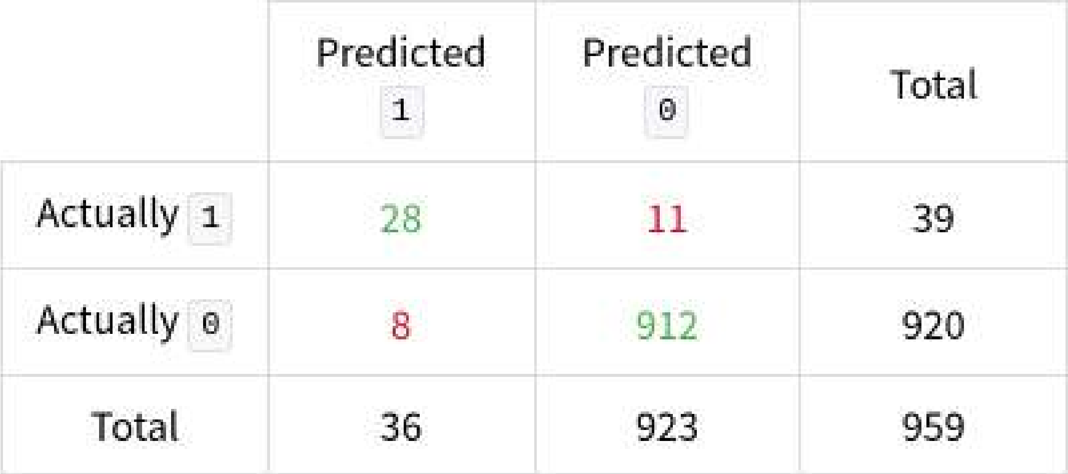
The table below lists all the performed trainings:

max_-depth	Score	Score StdDev	Fit Time	Fit Time StdDev	Score Time	Score Time StdDev
6	0.931400	0.009084	1.198800	0.134516	0.417400	0.201556
19	0.954949	0.015050	1.108400	0.281030	0.459200	0.275535

IV. Selected Model Results

A. Selected Model Metrics

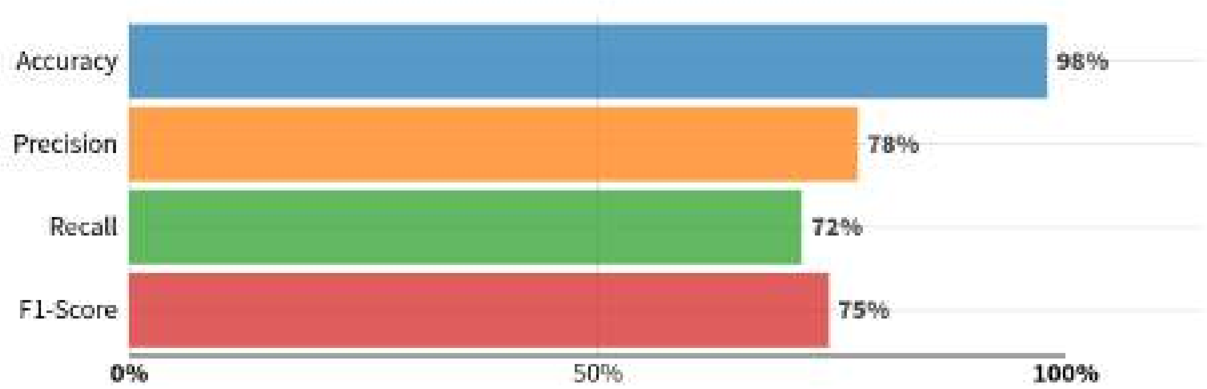
One way to assess the classification model performance is to use the “confusion matrix”, which compares actual values (from the test dataset) to predicted values:



	Predicted 1	Predicted 0	Total
Actually 1	28	11	39
Actually 0	8	912	920
Total	36	923	959

A classifier produces a probability that a given object belongs to the “positive” class (**1**). The threshold (or “cut-off”) is the number beyond which the prediction is considered “positive”. If set too low, it may predict “negative” too often, if set too high, too rarely. The confusion matrix was obtained with a threshold set at 0.350. The optimal value according to the F1 is 0.350.

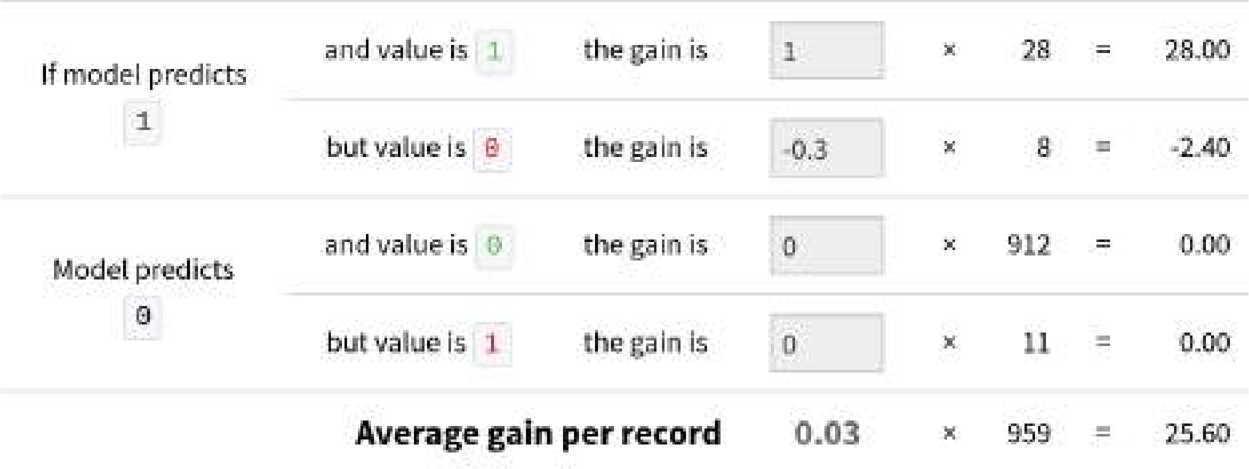
From this confusion matrix, several statistical metrics can be computed:



Legend

- *Precision*: Proportion of correct predictions among “positive” (**1**) predictions.
- *Recall*: Proportion of actually “positive” **(1)** records correctly predicted as “positive”.
- *F1-score*: Harmonic mean of precision and recall.
- *Accuracy*: Proportion of correct predictions among all predictions (“positive” or “negative”). Less informative than *F1-score* for unbalanced datasets.

The confusion matrix also allows to evaluate the average gain of using the classifier thanks to the provided costs of good and bad classifications:



If model predicts 1	and value is 1	the gain is	1	×	28	=	28.00
	but value is 0	the gain is	-0.3	×	8	=	-2.40
Model predicts 0	and value is 0	the gain is	0	×	912	=	0.00
	but value is 1	the gain is	0	×	11	=	0.00
	Average gain per record		0.03	×	959	=	25.60

The detailed metrics obtained on the test dataset are given below.

(1) Threshold independent

ROC AUC	Area under the ROC — From 0.5 (random model) to 1 (perfect model)	0.9471
Lift at 40%	Ratio between the true positive rate of this model and the one of a random model	2.3755

Average Precision	Summarizes a precision-recall curve - 1 (perfect model), positive-rate (random model)	0.0000
Log Loss	Error metric that takes into account the predicted probabilities (the lower the better)	0.1129
Calibration Loss	Average distance between calibration curve and diagonal — From 0 (perfectly calibrated) up to 0.5	0.0297

(2) Threshold dependent

Accuracy	Proportion of correct predictions (positive and negative) in the test set	0.9802
Precision	Proportion of positive predictions that were indeed positive (in the test set)	0.7778
Recall	Proportion of actual positive values found by the classifier	0.7179
F1-score	Harmonic mean between Precision and Recall	0.7467
Cost Matrix Gain	Average gain per record that the test set (959 rows) would yield given the specified gain for each outcome. Specified gains: TP = 1, TN = 0, FP = -0.3, FN = 0.	0.0267
Hamming Loss	Fraction of labels that are incorrectly predicted (the lower the better)	0.0198
Matthews Correlation Coefficient	Correlation coefficient between actual and predicted values. +1 = perfect, 0 = no correlation, -1 = perfect anti-correlation	0.7370

The threshold dependent metrics have been computed thanks to the confusion matrix while the others are based on predicted probabilities.

The ml assertions metrics are given below.

Name	Criteria	Expected class	Expected valid ratio	Rows matching criteria	Rows dropped by the model	Valid ratio	Result
	No assertions defined in the settings						

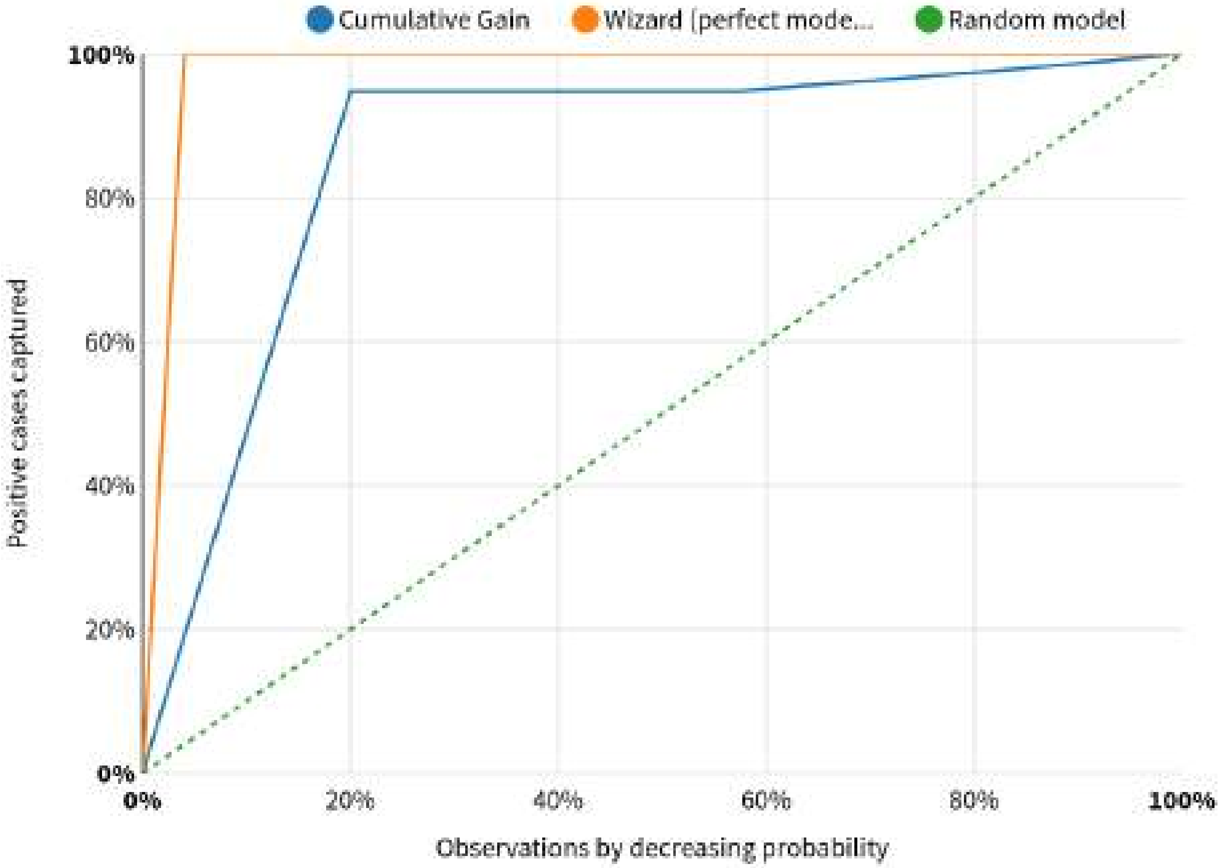
The ml override metrics are given below.

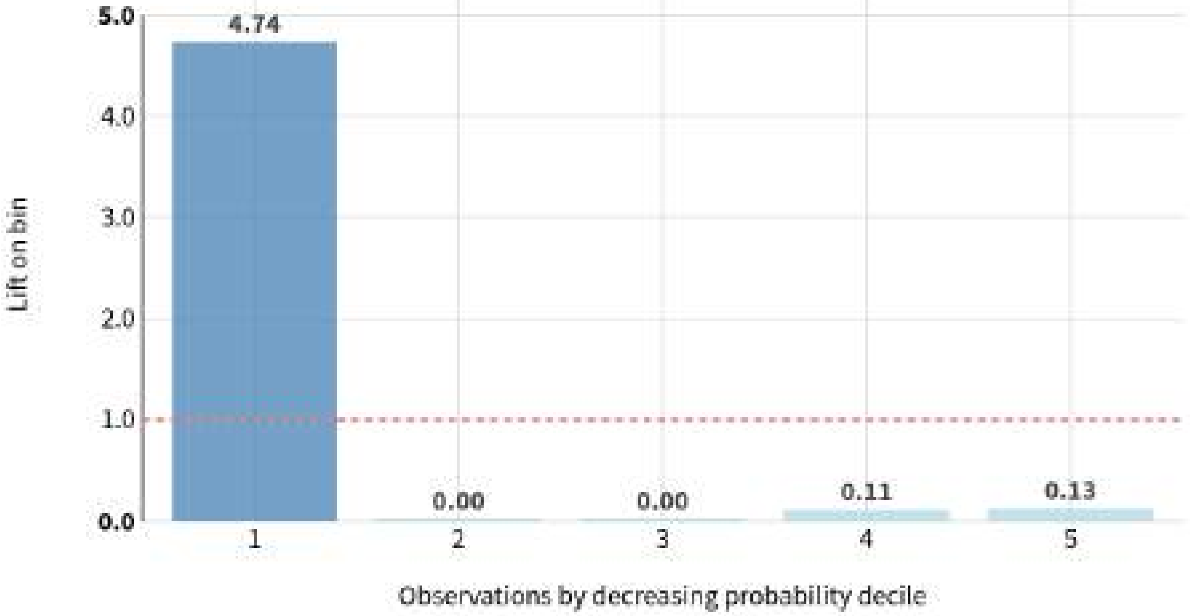
ML Overrides
No overrides defined in the settings

B. Selected Model Performance Charts

1. Lift Charts

A binary classifier produces a probability that a given record is "positive" (Here **1**). The lift is the ratio between the results of this model and the results obtained with a random model. Lift curves are particularly useful for "targeting" kinds of problems (churn prevention, marketing campaign targeting...)


● Cumulative Gain
● Wizard (perfect mode...
● Random model
100%
80%
60%
40%
20%
0%
Positive cases captured
0%
20%
40%
60%
80%
100%
Observations by decreasing probability


5.0
4.0
3.0
2.0
1.0
0.0
Lift on bin
4.74
0.00
0.00
0.11
0.13
1
2
3
4
5
Observations by decreasing probability decile

Cumulative Lift Curve

The curve displays the benefits of targeting a population subset with a model. On the horizontal axis, the percentage of the population which is targeted is shown. On the vertical axis, it is the percentage of found positive records (Here **1**).

- The dotted diagonal illustrates a *random model* (i.e., targeting 40% of the population will find 40% of the positive records).
- The *wizard* curve above shows a perfect model, i.e., a model that selects first all actually positive records.
- The cumulative gain curve shows the actual percentage of actually positives found by this model. The steeper the curve, the better.

Per-bin lift chart

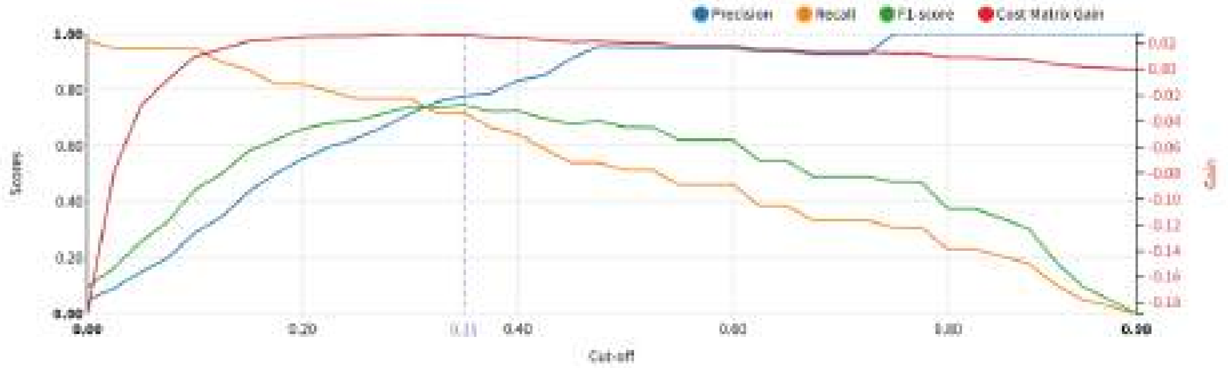
This chart sorts the observations by deciles of decreasing predicted probability. It shows the lift in each of the bins.

If there is 20% of positives (here **1**) in your test set, but 60% in the first bin of probability, then the lift of this first bin is 3. This means that targeting only the observations in this bin would yield 3 times as many positive results as a random sampling (equally sized bars at the level of the dotted line).

The bars should decrease progressively from left to right, and the higher the bars on the left, the better.

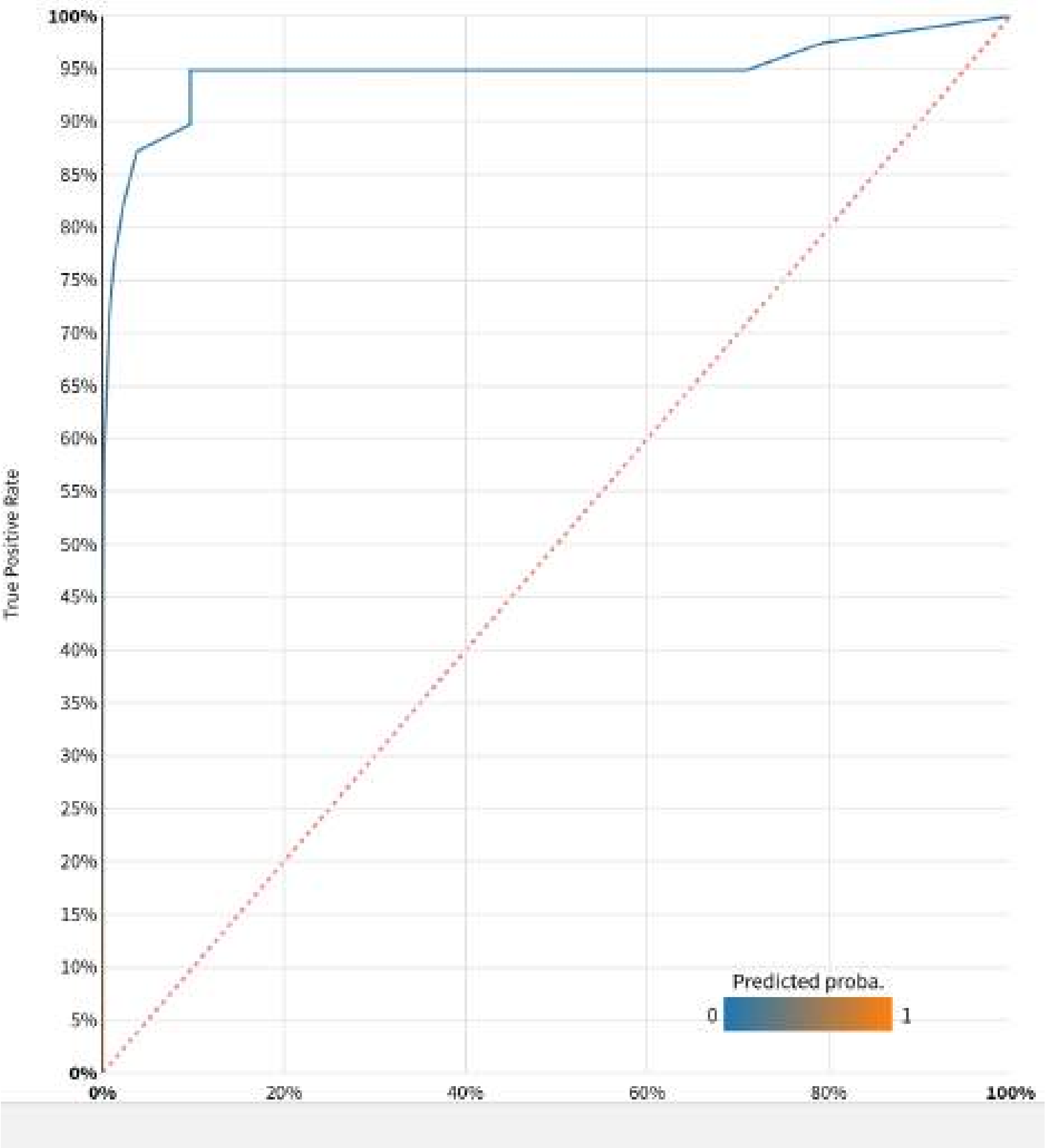
2. Decision Chart

The chart below shows how the threshold-based performance metrics of the classifier vary depending on the threshold.



3. ROC Curve

The Receiver Operating Characteristic (or ROC) curve shows the true positive rate versus the false positive resulting from different cutoffs in the predictive model. The “faster” the curve climbs, the better it is. On the contrary, a curve close to the diagonal line corresponds to a model with bad predictive power.



4. PR Curve

The Precision-Recall (or PR) curve illustrates the trade-off between precision and recall at different classification thresholds. A large area under the curve signifies both high precision (low false positive rate) and high recall (low false negative rate).

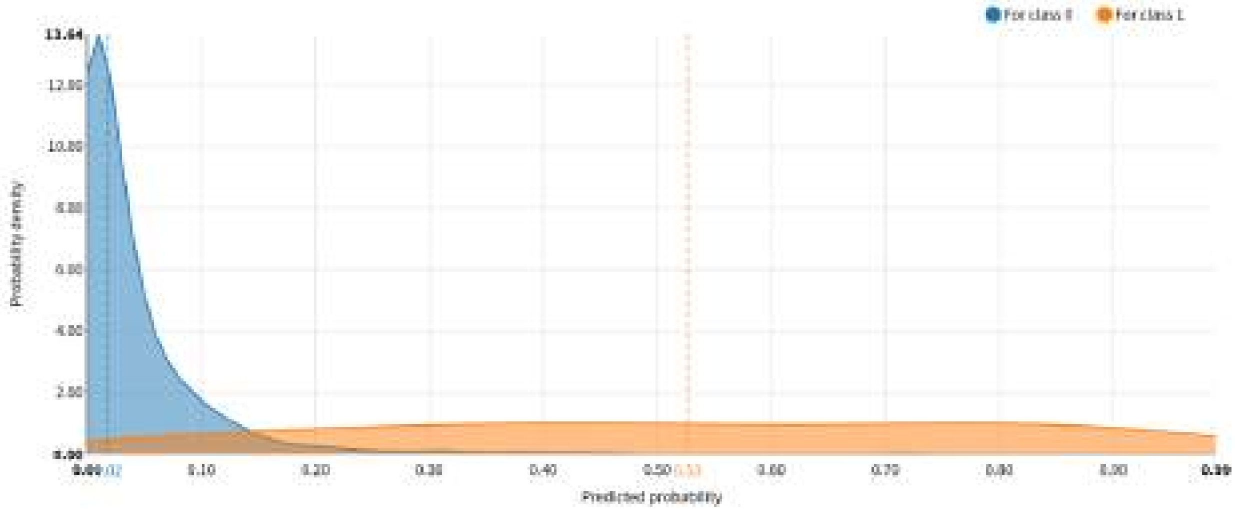
5. Density Chart

The density chart illustrates how the model succeeds in recognizing (and separating) the classes (e.g., 1 and 0 for binary classification). It shows the probability distribution of the actual classes in the test set given the predicted probability of being of the “positive” class (Here **1**). The two density functions show the probability density of rows in the test set that actually belongs to the “positive” class vs. rows that do not.

A perfect model entirely separates the density functions:

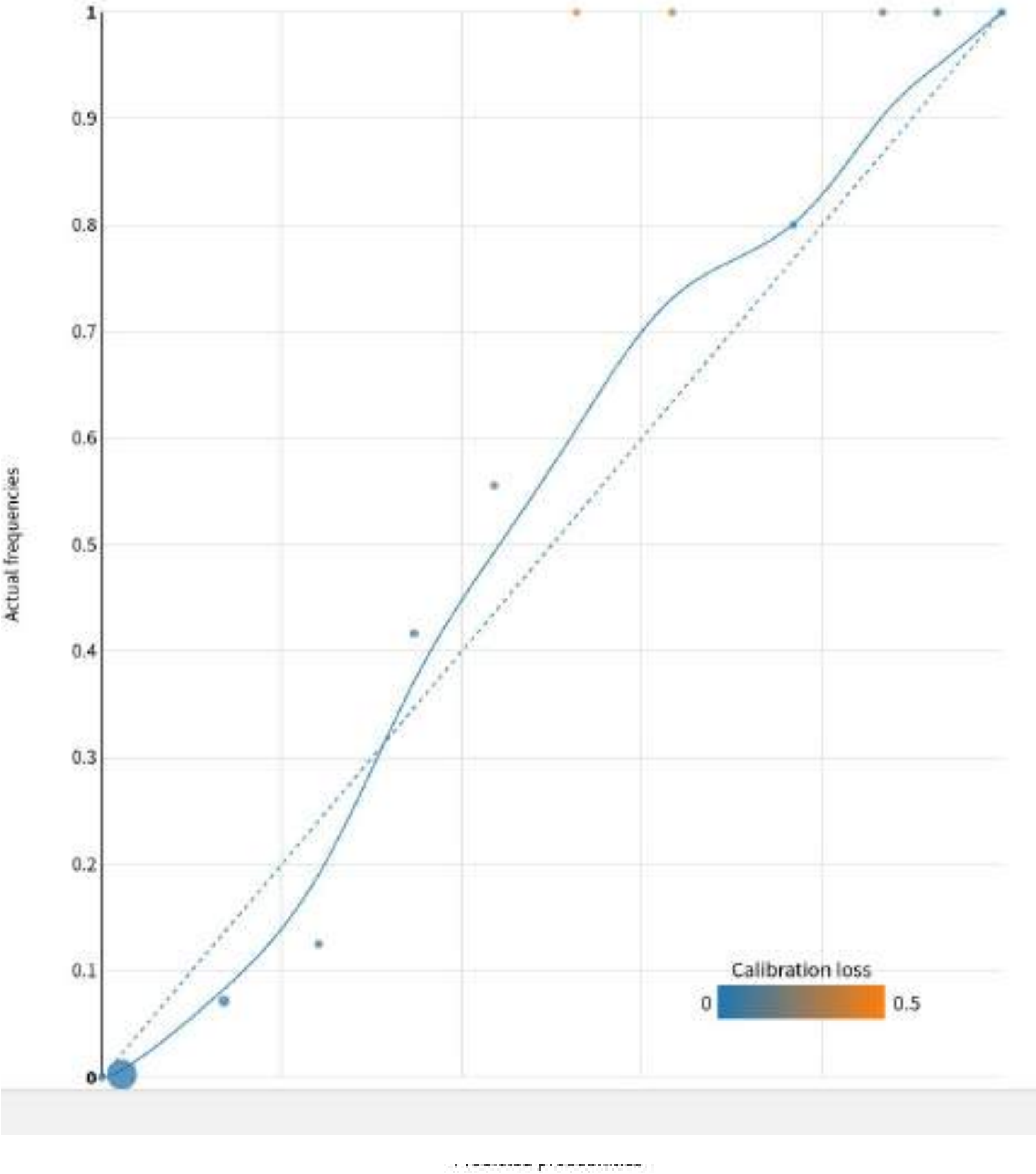
- The colored areas should not overlap.
- The density function of the “positive” class (**1**) should be entirely on the right.
- The density function of the “negative” class (**0**) should be entirely on the left.

The dotted vertical lines mark the medians.



6. Calibration

Calibration denotes the consistency between predicted probabilities and their actual frequencies observed on a test dataset.



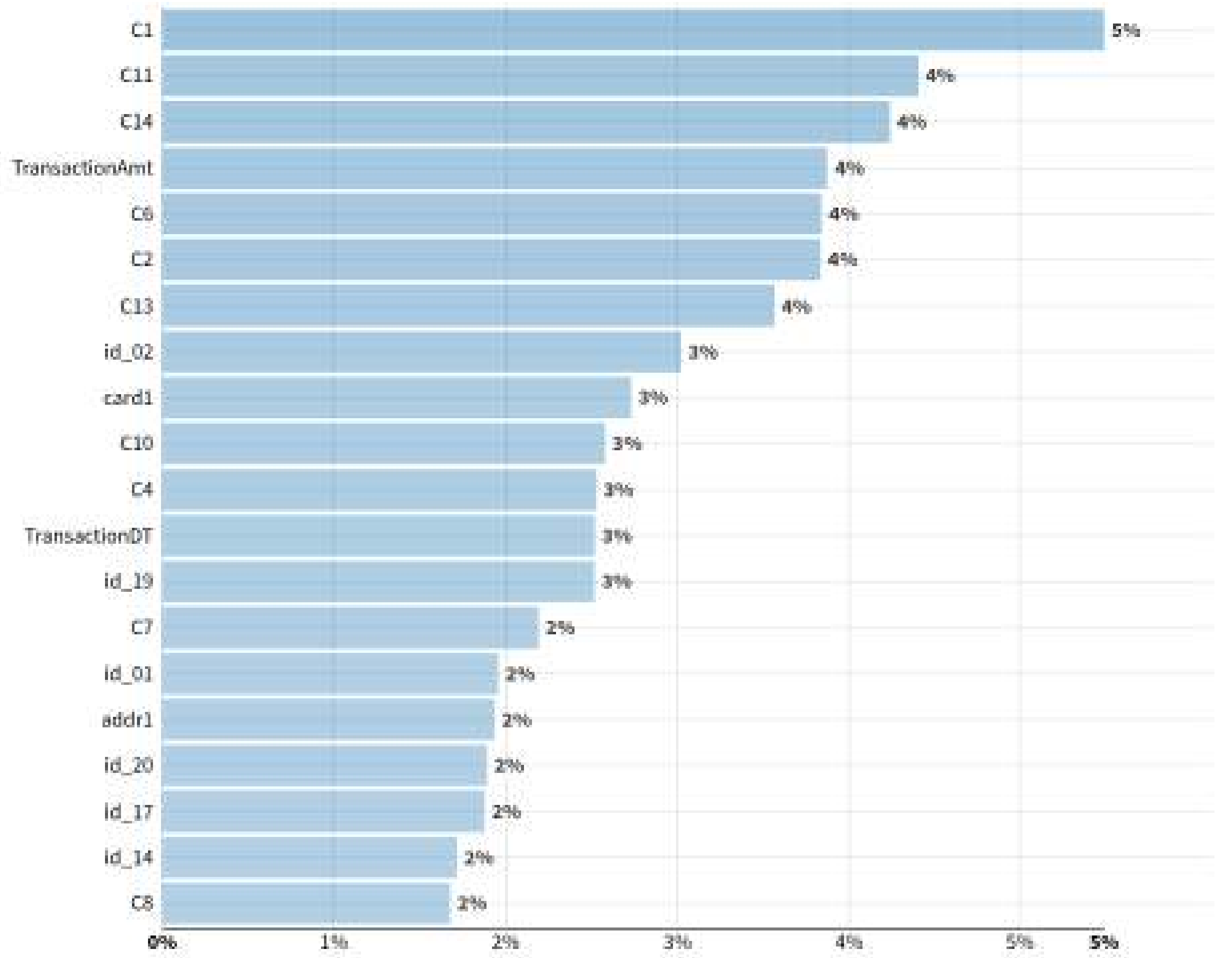
A perfectly calibrated model should have a calibration curve that is exactly on the diagonal line.

In reality, the calibration curve is often quite distinct from the diagonal line, and the average distance between the two measures the quality of the calibration.

The calibration loss is computed as the absolute difference between the calibration curve and the diagonal, averaged over the test set, weighted by the number of elements used to calculate each point (or the sum of sample weights when it applies).

C. Sensitivity Testing and Analysis

The selected algorithm has provided Gini feature importance values that assess which features have a significant impact on its performance.



D. Diagnostics

ML Diagnostics are designed to identify and help troubleshoot potential problems and suggest possible improvements at different stages of training and building machine learning models.

Dataset sanity checks	The train dataset is imbalanced (balance=0.30), metrics can be misleading Test set might be too small (959 rows) for reliable performance estimation The test dataset is imbalanced (balance=0.25), metrics can be misleading
Modeling parameters	Nothing to report

Training speed	Nothing to report
Training reproducibility	Disabled
Overfit detection	Nothing to report
Leakage detection	Nothing to report
Model check	Nothing to report
ML assertions	Nothing to report
Abnormal predictions detection	Disabled
Scoring dataset sanity checks	Disabled
Evaluation dataset sanity checks	Disabled
Time series resampling checks	Disabled
Treatment checks	Disabled
Propensity checks	Disabled
Evaluation error	Disabled

V. Deployment and Monitoring

A. Implementation Details

- The backend used by the model is:
- The model can be found here: https://dss-ebc67cd3-554cf99a-dku.us-east-1.app.dataiku.io/projects/CUSTOMERTRANSACTIONFRAUDDETECTION/savedmodels/i4M7ePdD/p/S-CUSTOMERTRANSACTIONFRAUDDETECTION-i4M7ePdD-1630307527537/report
- The name of the generated file is: Dataiku Model Documentation - Prediction (RANDOM_FOREST_CLASSIFICATION) on byTransactionID_joined_prepared3.docx
- The timing of the training was the following:

Preprocessed in	0.0s
Trained in	8.3s
Loading train set	0.1s
Loading test set	0.0s

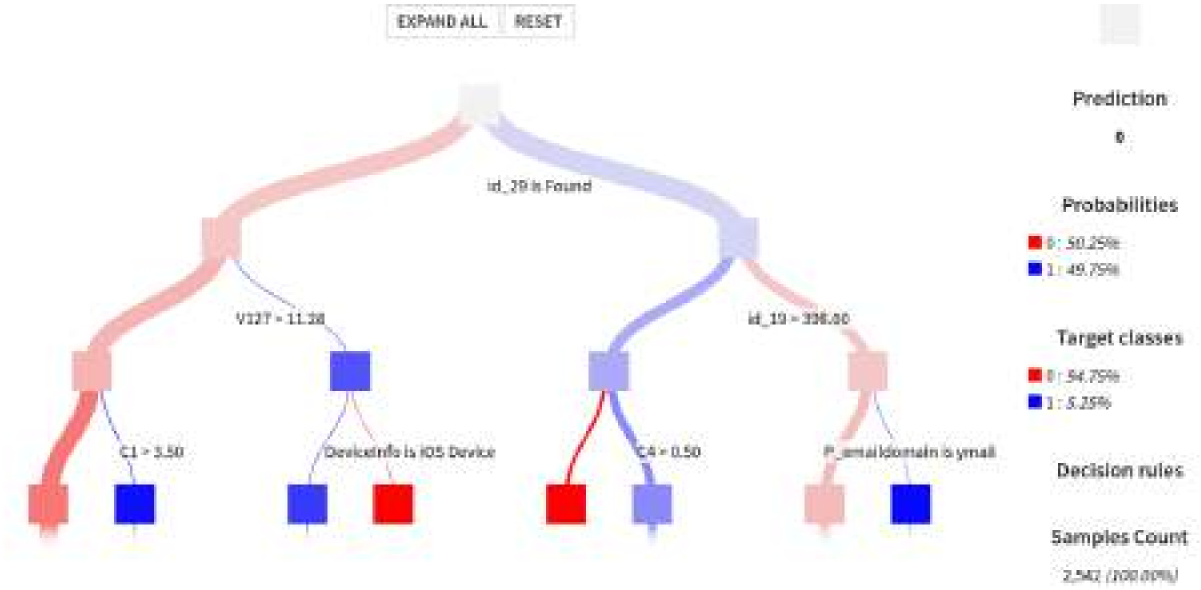
Collecting preprocessing data	0.2s
Preprocessing train set	0.3s
Preprocessing test set	0.2s
Hyperparameter searching	5.1s
Fitting model	0.8s
Saving model	0.0s
Scoring model	1.3s

B. Version Control

- The model was trained at 2021-08-30 07:12:07 (In the DSS server time zone).
- The model was trained with the following version of DSS: 13.4.0
- With the following code environment: DSS builtin environment

VI. Annexes

The first 3 levels of the decision tree are represented below:



Prediction

0

Probabilities

🟥 0 : 50.25%
🟦 1 : 49.75%

Target classes

🟥 0 : 94.75%
🟦 1 : 5.25%

Decision rules

Samples Count

2,542 (100.00%)