Predict bill_result
Performance modeling of bill_result on Challenge_8_AK_AZ_WA_prepared

Version	Author	Date
1.0	mcclenahan.kevin	2023-04-10 19:31:05

TABLE OF CONTENTS

EXECUTIVE SUMMARY

A Binary classification Machine Learning model was built using Dataiku DSS Visual ML. Its goal is to predict **bill_result** given a total of 15 features. Using a dataset of 4738 rows, the process led to the selection of the Random Forest algorithm.

Methodology

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

Policy	Split the dataset
Use time ordering	No
Sampling method	First records
Record limit	100000
Split mode	Random
Use K-fold cross-testing	No
Train ratio	0.8
Random seed	1337

See section 0 for detailed explanations about these options.

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

Search strategy	
Strategy	Random search
Search parameters	
Random state (hyperparameter search)	1337
Max number of iterations	24
Max search time	0 (no limit)
Parallelism	4

Cross-validation	
Cross-validation strategy	K-fold
Number of folds	5
Random state (cross-validation split)	1337
Stratified	Yes

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

Results

The Random Forest algorithm was selected. The evaluation metric used to tune the hyperparameters was ROC AUC 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 0.904.

METHODOLOGY

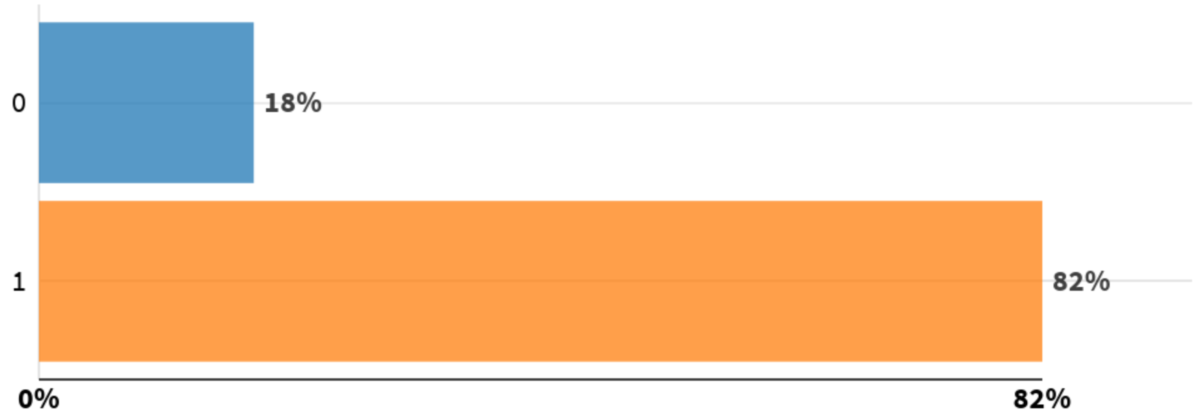
This section deals with the methodological details:

- The *Problem Definition* consists of selecting the target (**bill_result**) and the type of problem (Binary classification).
- *Data Ingestion* analyzes each feature in order to maximize its prediction potential.
- *Model and Feature Tuning* describes the tested algorithms and the way to find the best hyperparameter set for each of them.
- The *Model Evaluation and Selection* strategy indicates how to compute the metrics that allow for comparison between the best-tuned algorithms so that the user can select the best algorithm according to one of the computed metrics (Here Random Forest).

Problem Definition

A Binary classification Machine-Learning model was built using Dataiku DSS. Its goal is to predict **bill_result** given a total of 15 features.

The proportion of the target classes is shown on the graph below:



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.

Feature Name	Status	Type	Processing
identifier	Input	Category	Dummy encoding
jurisdiction.classification	Rejected	Category	
session	Input	Category	Dummy encoding
classification	Input	Category	Dummy encoding
from_organization.classification	Input	Category	Dummy encoding
bill_result	Target	Numeric	
bill_subject_3	Input	Category	Dummy encoding
bill_subject_4	Input	Category	Dummy encoding
bill_subject_5	Input	Category	Dummy encoding
state_party_affiliation	Input	Category	Dummy encoding
bill_subject_1	Input	Category	Dummy encoding
from_organization.name	Input	Category	Dummy encoding
bill_subject_2	Input	Category	Dummy encoding
jurisdiction.name	Input	Category	Dummy encoding
bill_party_affiliation	Input	Category	Dummy encoding

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…)

Model and Feature Tuning

Pre-processings

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

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

Tested Algorithms

A selection of algorithms (candidate models) was then trained on the Machine Learning dataset, with various combinations of hyperparameters. The section below details the tested algorithms and the space of hyperparameters for each of them. It begins with the selected algorithm and its hyperparameter selection and continues with the other tested algorithms.

Selected Model

The Random Forest algorithm has been finally selected.

A Random Forest is made of many decision trees. Each tree in the forest predicts a record, and each tree "votes" for the final answer of the forest.
The forest chooses the class having the most votes.
A decision tree is a simple algorithm which builds a decision tree. Each node of the decision tree includes a condition on one of the input features.
When "growing" (ie, training) the forest:
- for each tree, a random sample of the training set is used;
- for each decision point in the tree, a random subset of the input features is considered.
Random Forests generally provide good results, at the expense of "explainability" of the model.

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

Number of trees	Min: 80 Max: 200 Uniform distribution
Feature sampling strategy	Fixed proportion
Proportion of features to sample	Min: 0.1 Max: 0.7 Uniform distribution
Maximum depth of tree	Min: 6 Max: 20 Uniform distribution
Minimum samples per leaf	Min: 1 Max: 20 Uniform distribution
Parallelism	4

Alternative Models

Other algorithms are also tested. They are listed below, along with their settings:

Logistic Regression

Despite its name, Logistic Regression is a classification algorithm, using a linear model (i.e., it computes the target feature as a linear combination of input features).

Logistic Regression minimizes a specific cost function (called logit or sigmoid function), which makes it appropriate for classification.

A simple Logistic regression algorithm is prone to overfitting and sensitive to errors in the input dataset. To address these issues, it is possible to use a penalty (or regularization term) to the weights.

This implementation can use either 'L1' or 'L2' regularization terms.

Regularization	Try with L1 regularization: Yes Try with L2 regularization: No
C	Min: 0.01 Max: 100 Log uniform distribution

LightGBM

LightGBM is a tree-based gradient boosting library designed to be distributed and efficient. This algorithm provides fast training speed, low memory usage, good accuracy and is capable of handling large scale data.

For more information on gradient tree boosting, see the "Gradient tree boosting" algorithm.

Boosting type	Try Gradient Boosting Decision Tree: Yes Try Gradient One-Side sampling: No
Maximum number of trees	Min: 50 Max: 200 Uniform distribution
Maximum depth of trees	-1
Number of leaves	Min: 20 Max: 500 Uniform distribution
Learning rate	Min: 0.1 Max: 0.6 Uniform distribution
Minimum split gain	Min: 0 Max: 1 Uniform distribution
Minimum child weight	Min: 0.001 Max: 1 Uniform distribution
Minimum leaf samples	Min: 1 Max: 100 Uniform distribution
Columns subsample ratio for trees	Min: 0.5 Max: 1 Uniform distribution
L1 regularization	Min: 0 Max: 1 Uniform distribution
L2 regularization	Min: 0 Max: 1 Uniform distribution
Use bagging	Yes
Subsample ratio	0.75
Subsample frequency	2
Early stopping	Yes

Early stopping rounds	4
Random state	1337
Parallelism	4

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 (ROC AUC) computed on the validation dataset.

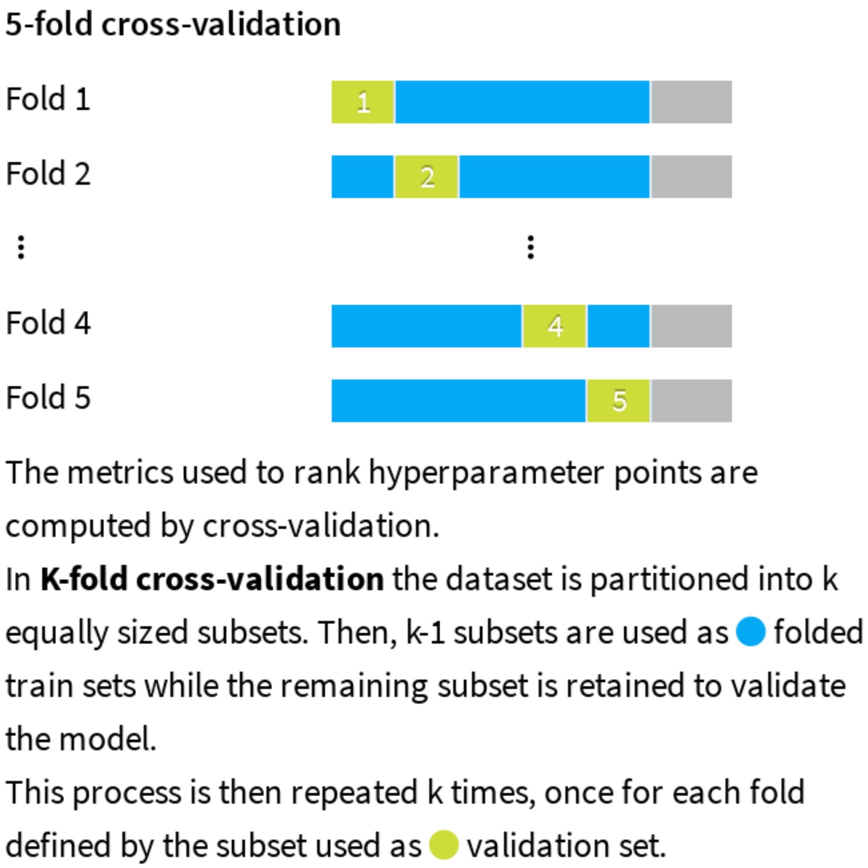
The actual search settings for all the tested algorithms, including the selected one, are the following:

Search strategy	
Strategy	Random search
Search parameters	
Random state (hyperparameter search)	1337
Max number of iterations	24
Max search time	0 (no limit)
Parallelism	4
Cross-validation	
Cross-validation strategy	K-fold
Number of folds	5
Random state (cross-validation split)	1337
Stratified	Yes

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:



The metrics used to rank hyperparameter points are computed by cross-validation.

In **K-fold cross-validation** the dataset is partitioned into k equally sized subsets. Then, k-1 subsets are used as ● folded train sets while the remaining subset is retained to validate the model.

This process is then repeated k times, once for each fold defined by the subset used as ● validation set.

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

Weighting Strategy

Each row weight is defined as inversely proportional to the cardinality of its target class.

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:

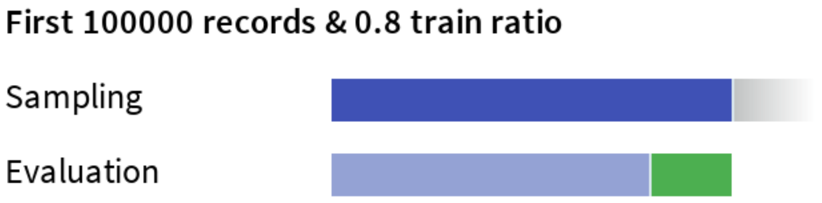
- Splitting the dataset by setting apart a test dataset, also called the hold-out dataset, for this performance evaluation. The train ratio indicates the amount of the dataset used in training, the remaining being used for evaluation.
- Performing a K-fold evaluation. It allows a more precise performance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

Policy	Split the dataset
Use time ordering	No
Sampling method	First records
Record limit	100000
Split mode	Random
Use K-fold cross-testing	No
Train ratio	0.8
Random seed	1337

Illustration:



The metrics used to rank models obtained by different algorithms are computed on the ● test set. The final model is trained on the ● train set.

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 4738 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.

EXPERIMENT RESULTS

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

Selected Model

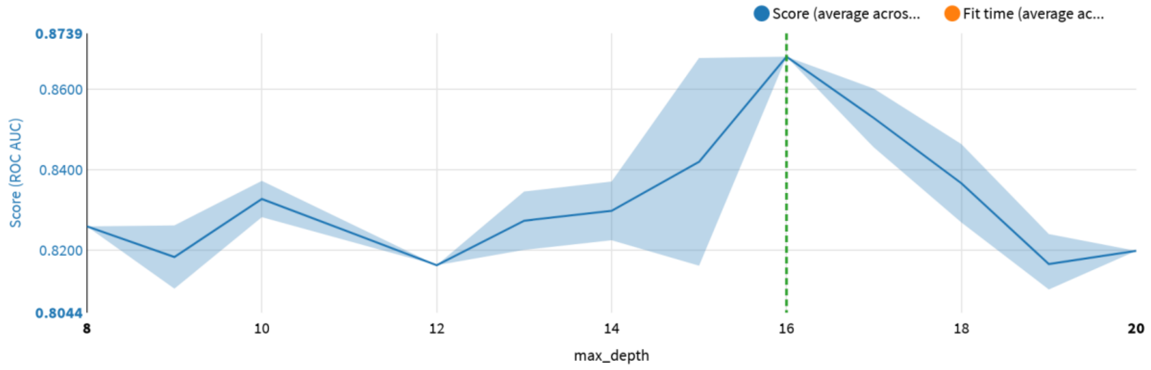
Random Forest was finally selected by the user with the optimal set of hyperparameters given in the table below:

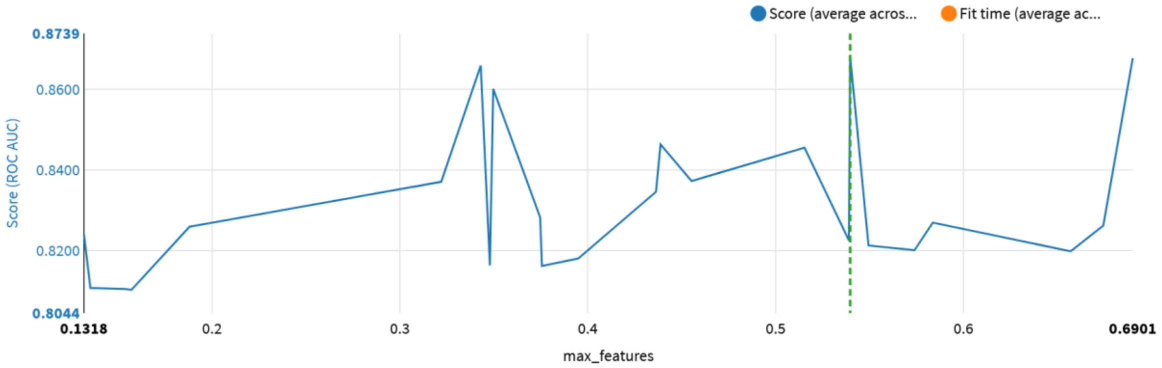
Algorithm	Random forest classification	Split quality criterion	Gini
Number of trees	107	Use bootstrap	Yes
Max trees depth	16	Feature sampling strategy	prop
Min samples per leaf	2	Used features	54%
Min samples to split	6		

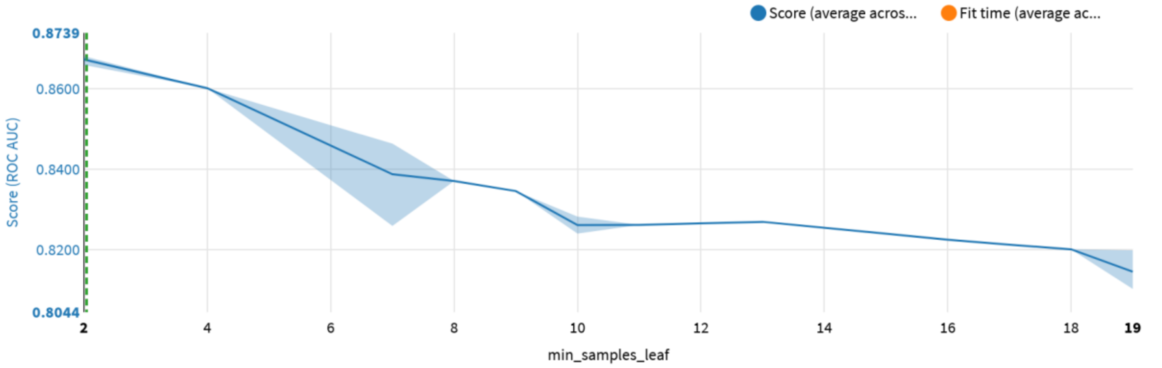
See section 0 for detailed explanations on the algorithm and its hyperparameters.

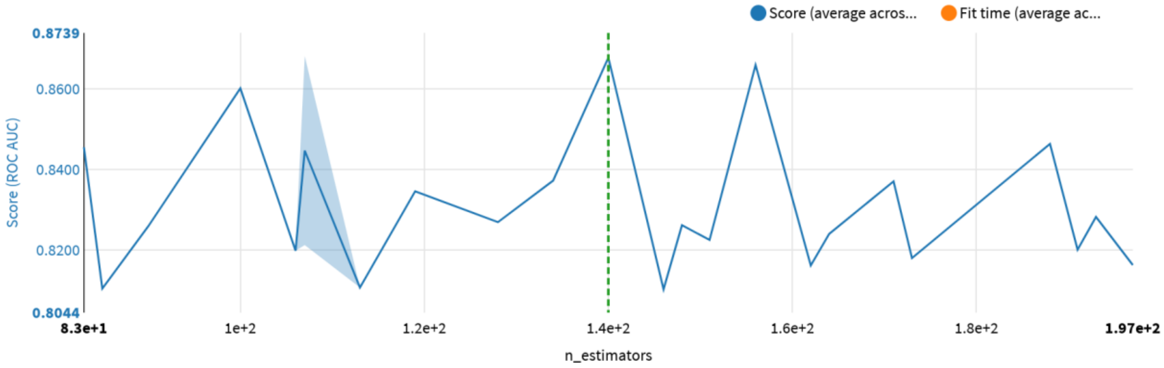
Alternative Models

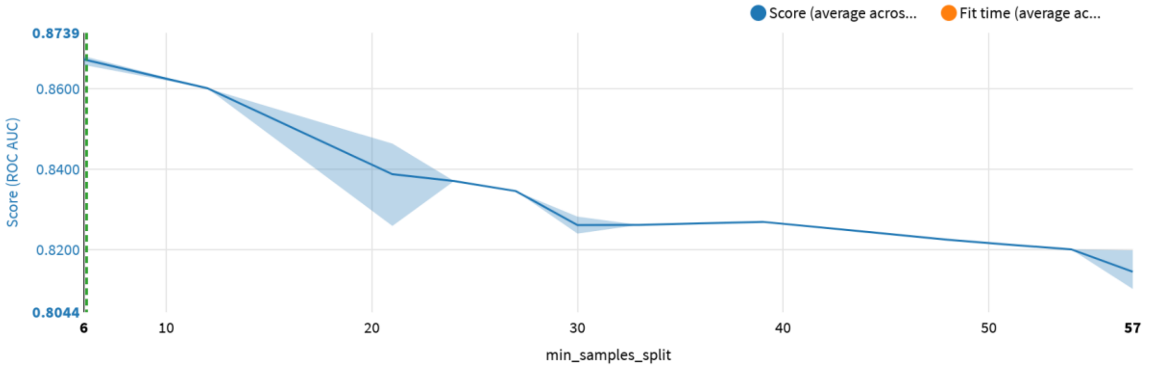
For the selected algorithm, the following other hyperparameter combinations were tried and led to lower performance. As an example, the plot below shows the evolution of the performance for each hyperparameter:




● Score (average acros...
● Fit time (average ac...
0.8739
0.8600
0.8400
0.8200
0.8044
Score (ROC AUC)
0.1318
0.2
0.3
0.4
0.5
0.6
0.6901
max_features


● Score (average acros...
● Fit time (average ac...
0.8739
0.8600
0.8400
0.8200
0.8044
Score (ROC AUC)
2
4
6
8
10
12
14
16
18
19
min_samples_leaf


● Score (average acros...
● Fit time (average ac...
0.8739
0.8600
0.8400
0.8200
0.8044
Score (ROC AUC)
8.3e+1
1e+2
1.2e+2
1.4e+2
1.6e+2
1.8e+2
1.97e+2
n_estimators

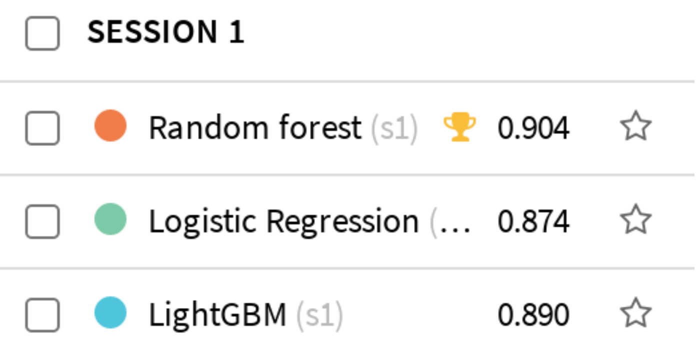

● Score (average acros...
● Fit time (average ac...
0.8739
0.8600
0.8400
0.8200
0.8044
Score (ROC AUC)
6
10
20
30
40
50
57
min_samples_split

The table below lists all the performed trainings:

max_-depth	max_-features	min_-samples_leaf	n_estimators	min_-samples_split	Score	Score StdDev	Fit Time	Fit Time StdDev	Score Time	Score Time StdDev
8	0.1880151778946672	7	90	21	0.825892	0.018287	4.489200	1.546666	1.235600	0.949484
9	0.6744155342027564	11	148	33	0.826137	0.018417	21.998400	1.515265	3.609800	0.910091
9	0.1534270403214963	19	85	57	0.810404	0.014966	4.757600	0.915467	1.333200	0.423742
10	0.45526782093320606	7	134	21	0.837228	0.018892	16.251400	0.880829	2.881800	0.513276
10	0.3746275505124097	10	193	30	0.828204	0.016574	18.621200	0.953925	4.960000	0.323375
12	0.34788478085479535	19	197	57	0.816245	0.015336	15.860600	0.348978	5.124400	0.421736
13	0.4363178483784972	9	119	27	0.834552	0.018130	16.240000	0.661549	1.361000	0.714538
13	0.5739371275638874	18	191	54	0.820062	0.016518	23.199000	1.059500	4.242200	0.990697
14	0.32197125936064896	8	171	24	0.837041	0.016813	15.959200	0.939976	5.366400	0.733848
14	0.5390140412005745	16	151	48	0.822473	0.016281	18.598000	0.887362	3.730200	0.484997
15	0.37559013232873995	19	162	57	0.816123	0.014418	16.241400	0.744571	2.467200	0.573539
15	0.6901291630861847	2	140	6	0.867782	0.014386	27.801000	1.165011	4.920800	0.617881
15	0.34301252886495703	2	156	6	0.865930	0.015701	16.743800	0.400108	5.178200	0.515406
15	0.39488891789885183	19	173	57	0.817982	0.015155	15.313800	1.716636	3.265000	0.906563
16	0.5396887316142894	2	107	6	0.868116	0.014325	18.594600	0.904440	2.881400	0.334764
17	0.3496623637425988	4	100	12	0.860146	0.016830	11.154800	1.295199	3.364400	0.606931
17	0.5153530858925176	7	83	21	0.845516	0.017585	12.137200	0.872214	2.362000	0.509182
18	0.583659121407126	13	128	39	0.826900	0.016254	15.237200	0.922665	3.111000	0.494431
18	0.43871138768472306	7	188	21	0.846340	0.018090	22.942800	2.710669	5.180800	1.185817
19	0.15708867592439582	19	146	57	0.810212	0.015155	6.896800	0.692430	3.403000	0.637022
19	0.13520759534806753	19	113	57	0.810667	0.013995	5.441600	1.702908	2.208000	0.829332

19	0.1317641885691451	10	164	30	0.823986	0.015232	8.059200	0.908996	3.957200	1.111253
19	0.5494346253691668	17	107	51	0.821226	0.015818	11.418800	1.162745	2.725200	0.524342
20	0.6570442788259865	19	106	57	0.819792	0.015824	12.935800	1.099623	2.902600	0.334616

The selected algorithm was compared with other algorithms. The table below gives the performance obtained with the combination of hyperparameters that optimizes the ROC AUC metric:



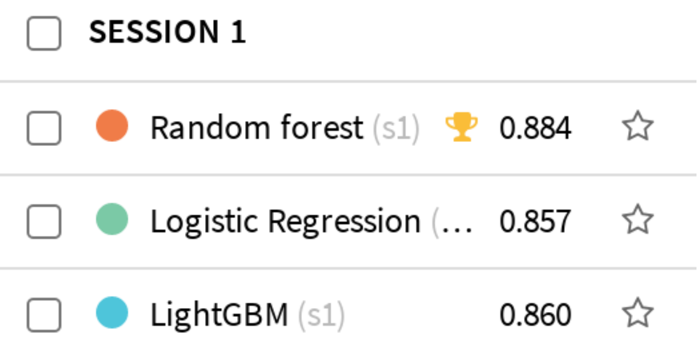
□ **SESSION 1**

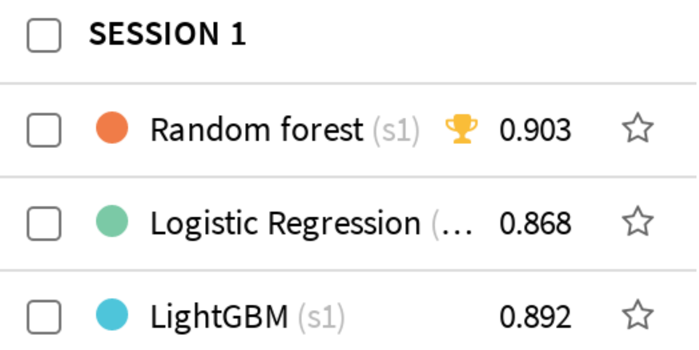
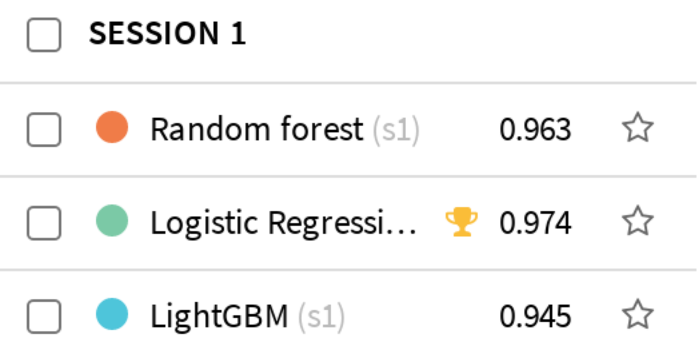
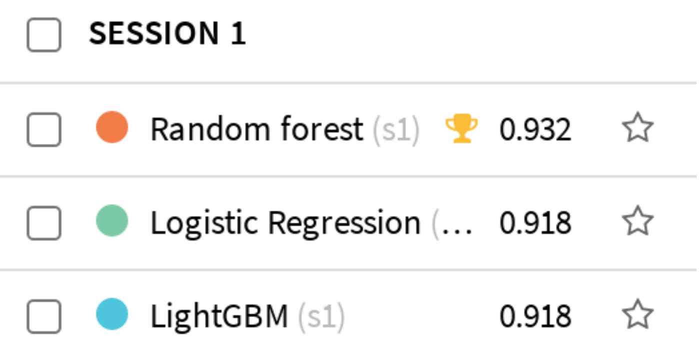
□ ● Random forest (s1) 🏆 0.904 ☆

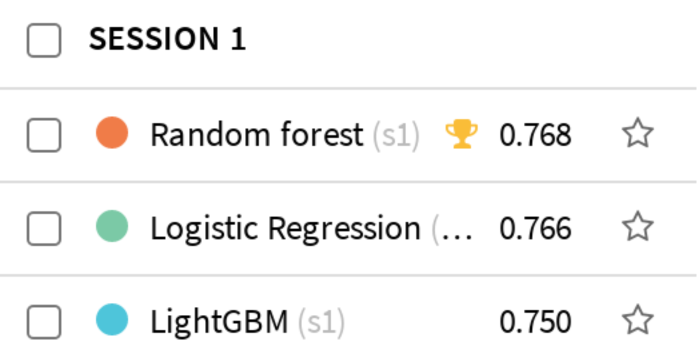
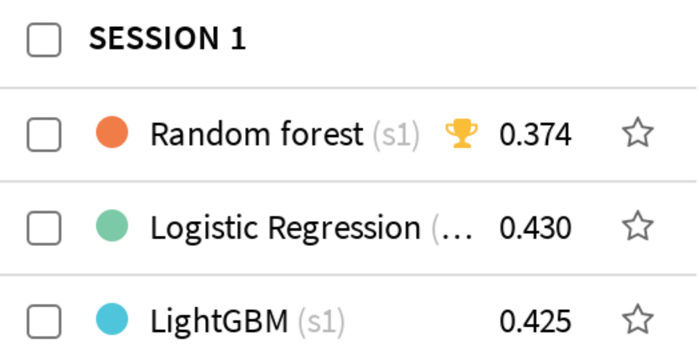
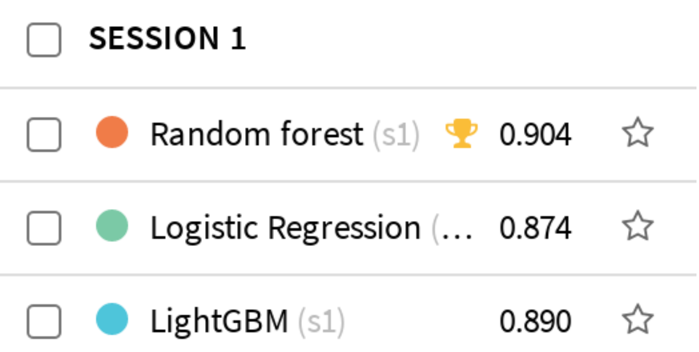
□ ● Logistic Regression (… 0.874 ☆

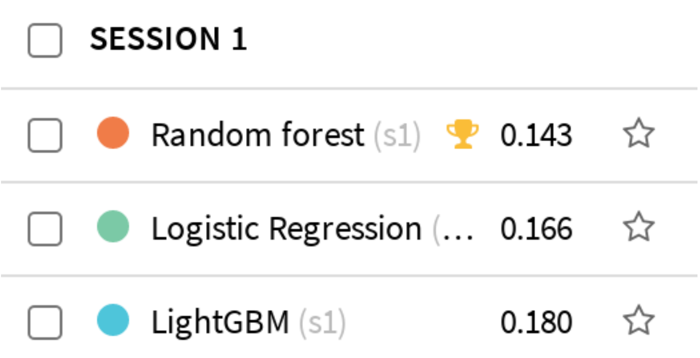
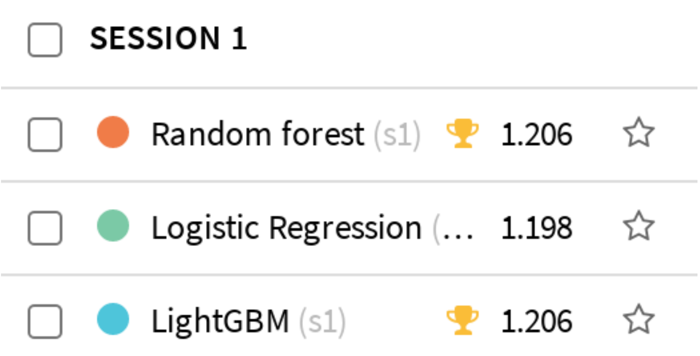
□ ● LightGBM (s1) 0.890 ☆

Complete performance results obtained with the other evaluated metrics are given below:

Accuracy	 □ **SESSION 1** □ ● Random forest (s1) 🏆 0.884 ☆ □ ● Logistic Regression (… 0.857 ☆ □ ● LightGBM (s1) 0.860 ☆

Precision	 □ **SESSION 1** □ ● Random forest (s1) 🏆 0.903 ☆ □ ● Logistic Regression (… 0.868 ☆ □ ● LightGBM (s1) 0.892 ☆
Recall	 □ **SESSION 1** □ ● Random forest (s1) 0.963 ☆ □ ● Logistic Regressi… 🏆 0.974 ☆ □ ● LightGBM (s1) 0.945 ☆
F1 Score	 □ **SESSION 1** □ ● Random forest (s1) 🏆 0.932 ☆ □ ● Logistic Regression (… 0.918 ☆ □ ● LightGBM (s1) 0.918 ☆

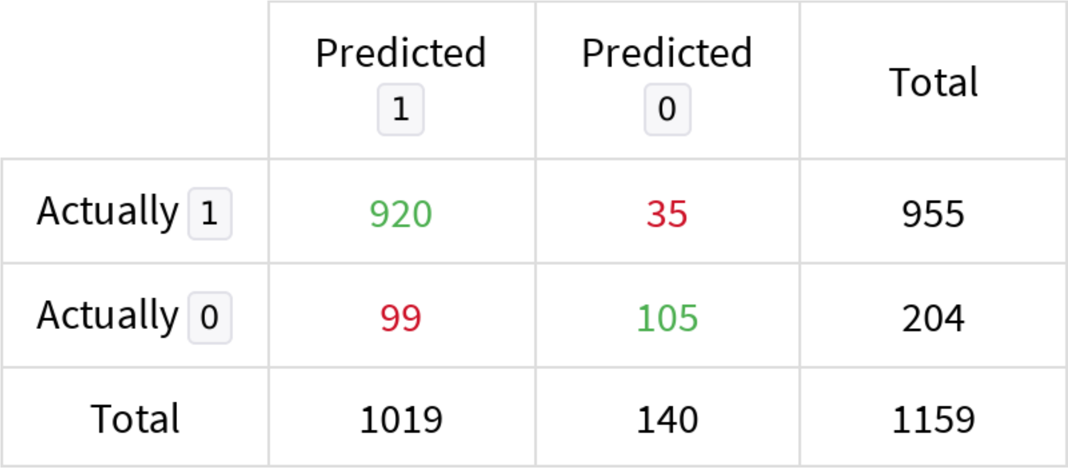
Cost Matrix Gain	 □ SESSION 1 □ ● Random forest (s1) 🏆 0.768 ☆ □ ● Logistic Regression (… 0.766 ☆ □ ● LightGBM (s1) 0.750 ☆
Log Loss	 □ SESSION 1 □ ● Random forest (s1) 🏆 0.374 ☆ □ ● Logistic Regression (… 0.430 ☆ □ ● LightGBM (s1) 0.425 ☆
ROC AUC	 □ SESSION 1 □ ● Random forest (s1) 🏆 0.904 ☆ □ ● Logistic Regression (… 0.874 ☆ □ ● LightGBM (s1) 0.890 ☆

Calibration Loss	 □ SESSION 1 □ ● Random forest (s1) 🏆 0.143 ☆ □ ● Logistic Regression (… 0.166 ☆ □ ● LightGBM (s1) 0.180 ☆
Lift	 □ SESSION 1 □ ● Random forest (s1) 🏆 1.206 ☆ □ ● Logistic Regression (… 1.198 ☆ □ ● LightGBM (s1) 🏆 1.206 ☆

SELECTED MODEL RESULTS

Selected Model Metrics

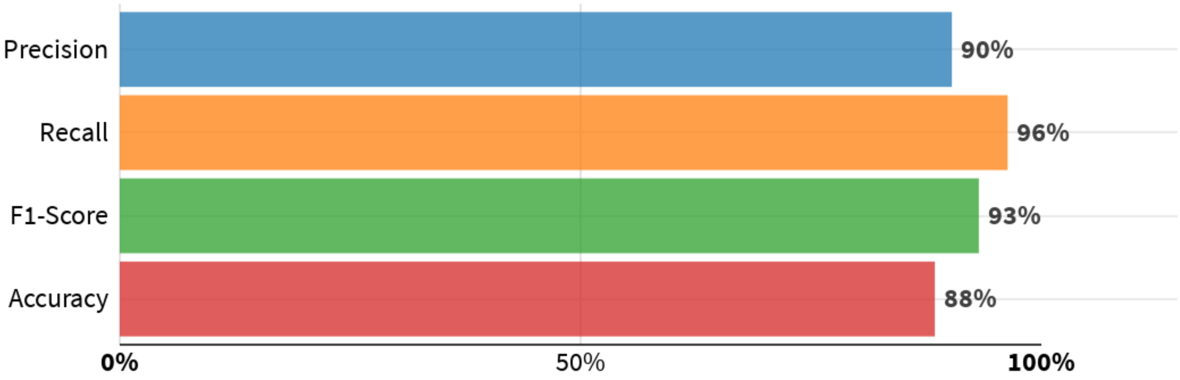
One way to assess the classification model performance is to use the “confusion matrix”, which compares actual values (from the test dataset) to predicted values:



	Predicted 1	Predicted 0	Total
Actually 1	920	35	955
Actually 0	99	105	204
Total	1019	140	1159

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 0.275. The optimal value according to the F1 is 0.275.

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

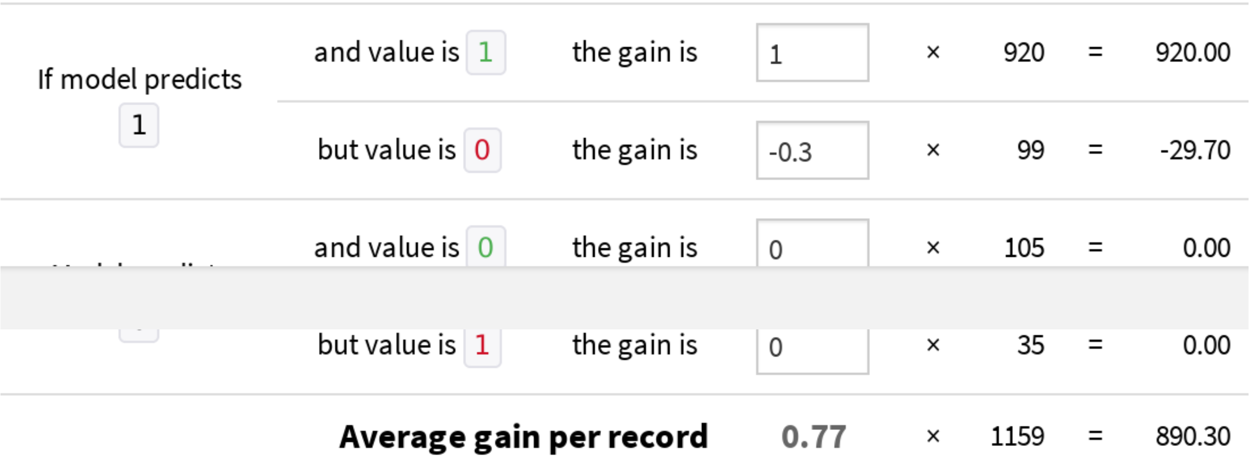


Legend

- *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	×	920	=	920.00
	but value is 0	the gain is	-0.3	×	99	=	-29.70
	and value is 0	the gain is	0	×	105	=	0.00
	but value is 1	the gain is	0	×	35	=	0.00
		Average gain per record	**0.77**	×	1159	=	890.30

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

Threshold independent

Log loss	Error metric that takes into account the predicted probabilities (the lower the better)	0.3739
ROC - AUC Score	Area under the ROC; from 0.5 (random model) to 1 (perfect model)	0.9039
Calibration loss	Average distance between calibration curve and diagonal. From 0 (perfectly calibrated) up to 0.5.	0.1431

Threshold dependent

Accuracy	Proportion of correct predictions (positive and negative) in the test set	0.8844
Precision	Proportion of positive predictions that were indeed positive (in the test set)	0.9028
Recall	Proportion of actual positive values found by the classifier	0.9634
F1 Score	Harmonic mean between Precision and Recall	0.9321
Hamming loss	Fraction of labels that are incorrectly predicted (the lower the better)	0.1156
Cost matrix gain	Average gain per record that the test set (1159 rows) would yield given the specified gain for each outcome.Specified gains: TP = 1, TN = 0, FP = -0.3, FN = 0.	0.7682
Matthews Correlation Coefficient	Correlation coefficient between actual and predicted values. +1 = perfect, 0 = no correlation, -1 = perfect anti-correlation	0.5587

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

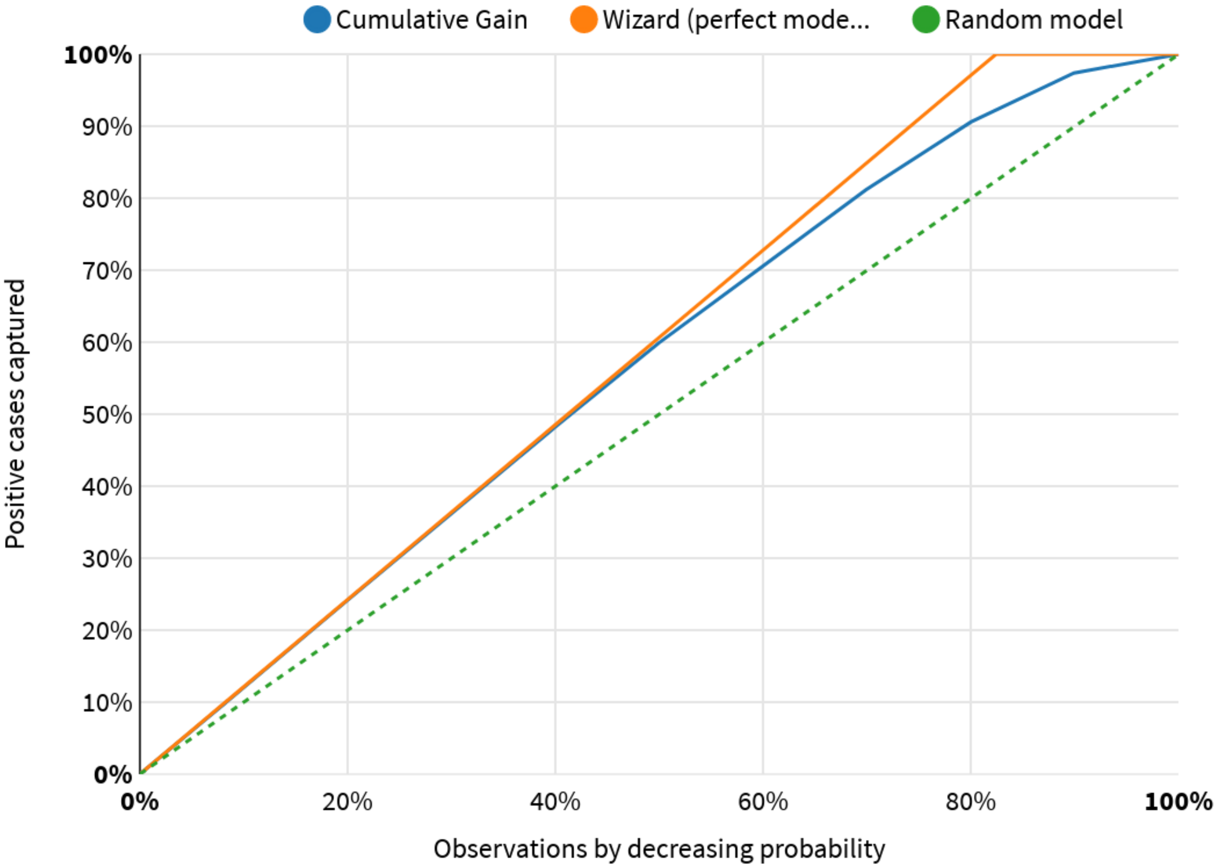
The ml assertions metrics are given below.

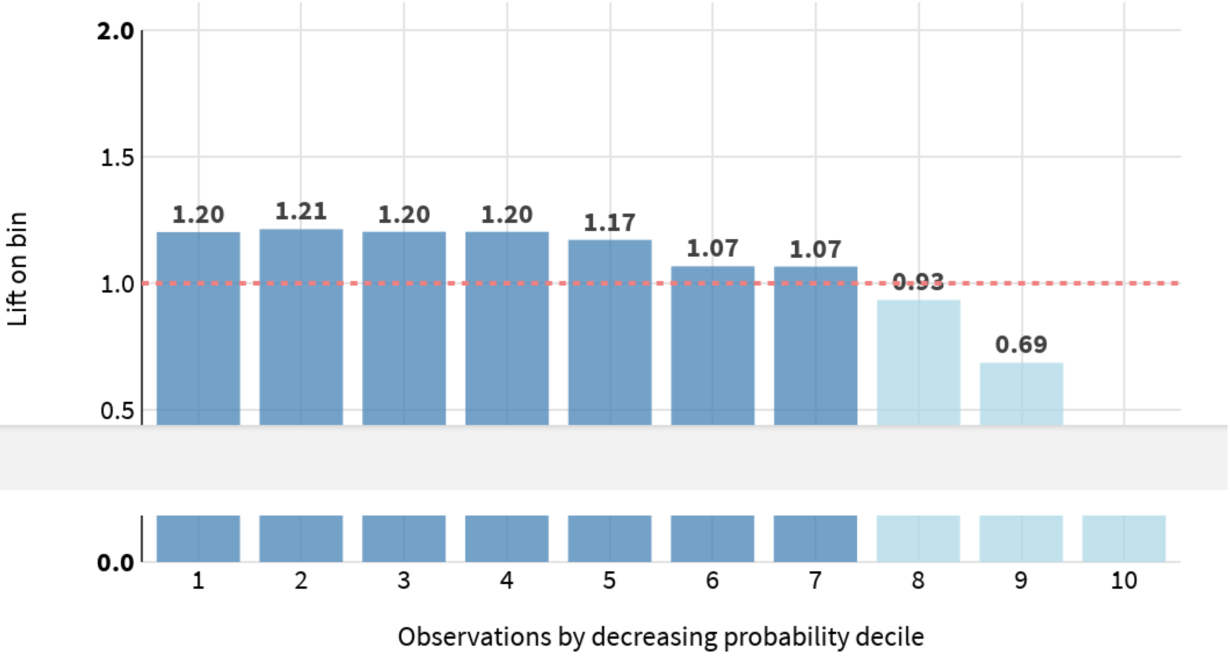
Name	Criteria	Expected class	Expected valid ratio	Rows matching criteria	Rows dropped by the model	Valid ratio	Result
	No assertions defined in the settings						

Selected Model Performance Charts

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 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

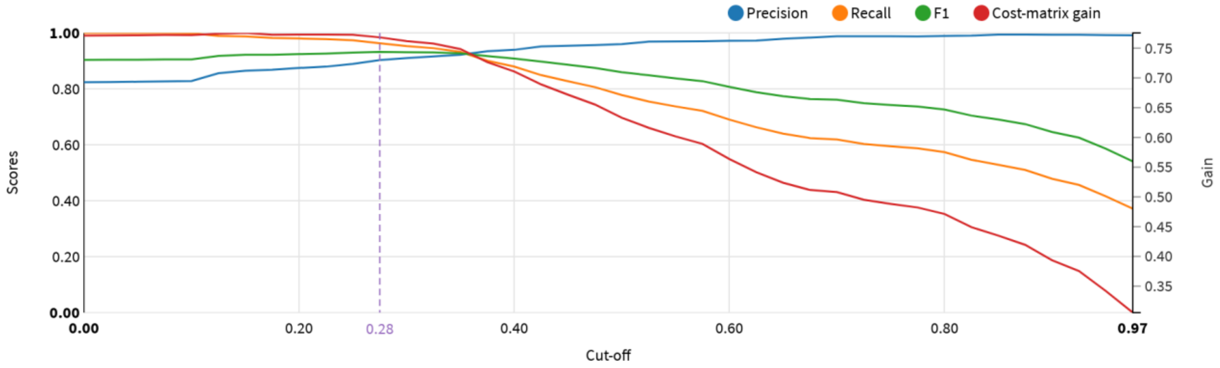
This chart sorts the observations by deciles of decreasing predicted probability. It shows the lift in each of the bins.

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.

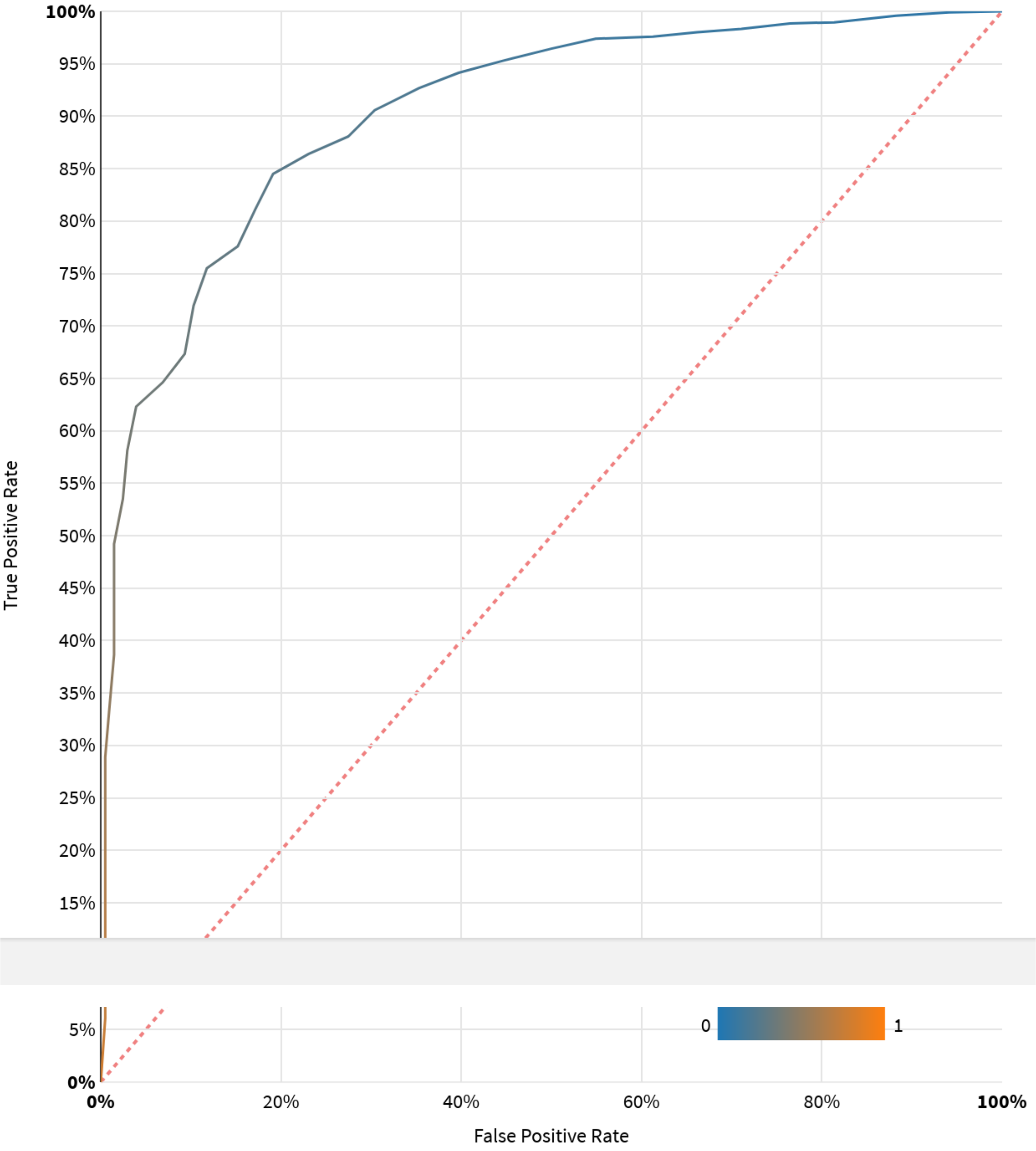
Decision Chart

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



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.



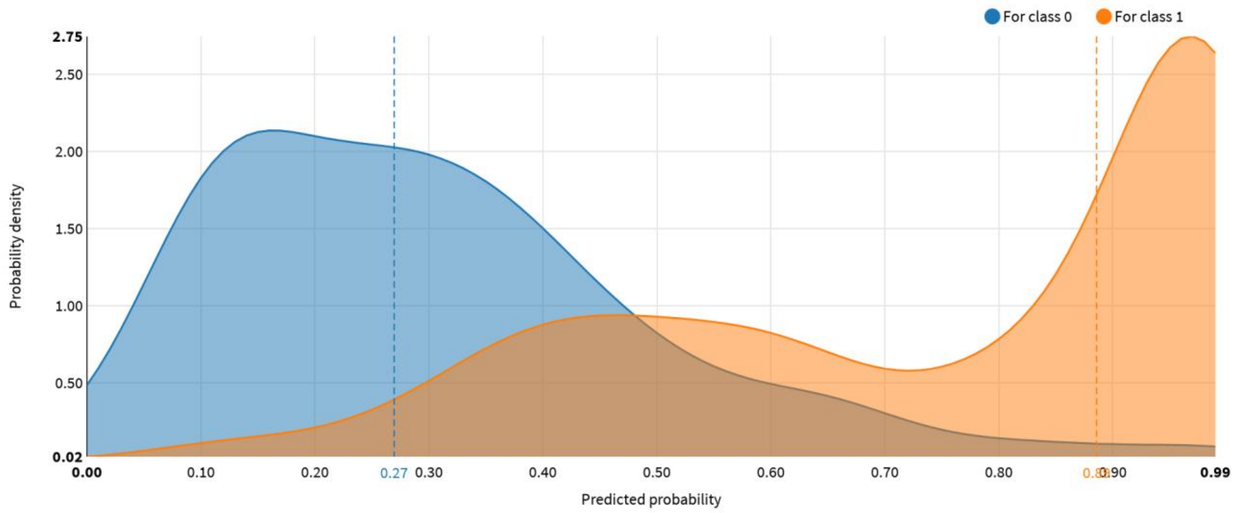
Density Chart

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:

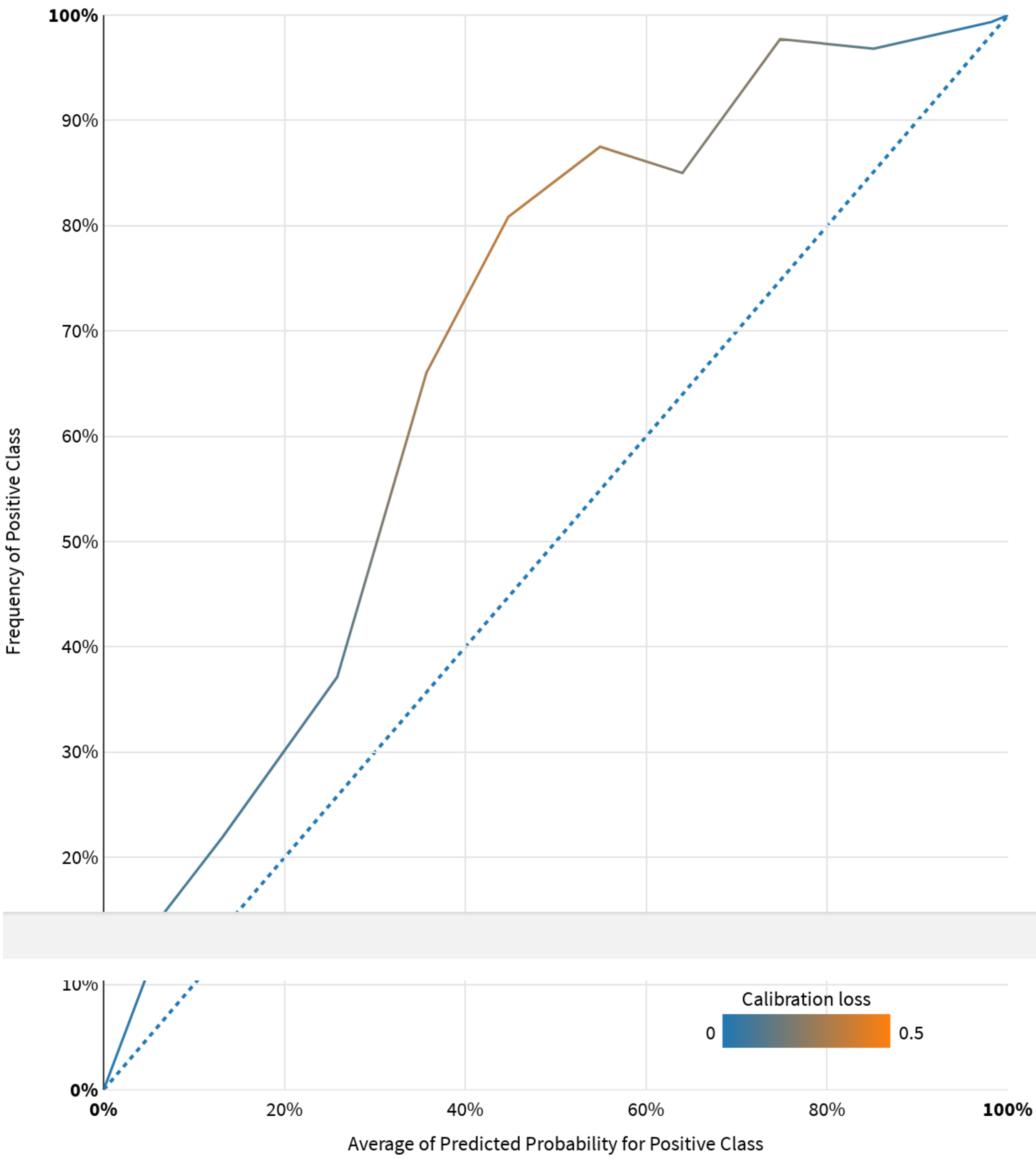
- The colored areas should not overlap.
- The density function of the “positive” class (**1**) should be entirely on the right.
- The density function of the “negative” class (**0**) should be entirely on the left.

The dotted vertical lines mark the medians.



Calibration

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



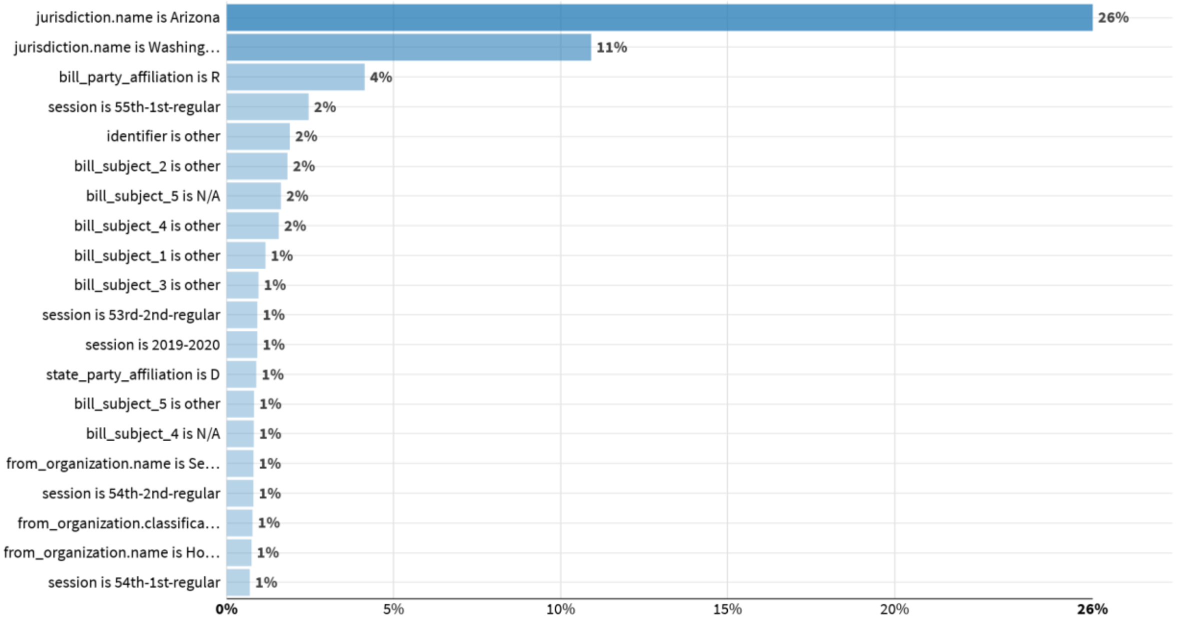
A perfectly calibrated model should have a calibration curve that is exactly on the diagonal line.

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).

Sensitivity Testing and Analysis

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



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	Nothing to report
Modeling parameters	Nothing to report

Training speed	Nothing to report
Overfit detection	Nothing to report
Leakage detection	Nothing to report
Model check	Nothing to report
ML assertions	Nothing to report
Abnormal predictions detection	Nothing to report

DEPLOYMENT AND MONITORING

Implementation Details

- The backend used by the model is: Python (in memory)
- The model can be found here: https://dss-efcd7e91-005a5115-dku.us-east-1.app.dataiku.io/projects/GEORGIA_TECH_PRACTICUM_2023_SPRING/analysis/GsSL7IFi/ml/p/P72qF6zQ/A-GEORGIA_TECH_PRACTICUM_2023_SPRING-GsSL7IFi-P72qF6zQ-s1-pp1-m1/report
- The name of the generated file is: Dataiku Model Documentation - Predict bill_result on Performance modeling of bill_result on Challenge_8_AK_AZ_WA_prepared - Random Forest.docx
- The timing of the training was the following:

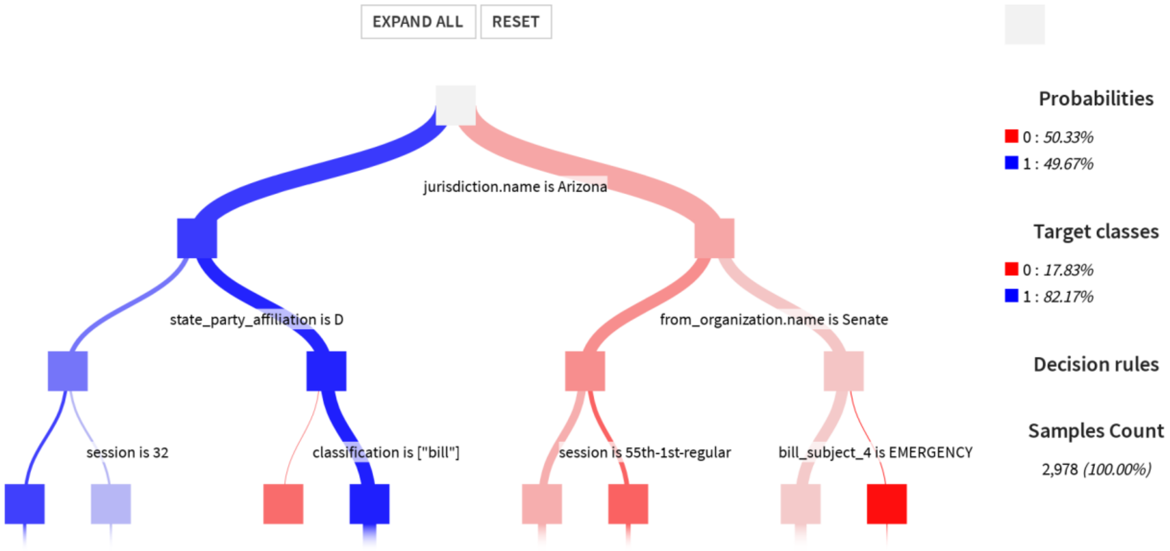
Preprocessed in	0.2s
Trained in	557.5s
Loading train set	0.0s
Loading test set	0.0s
Collecting statistics	0.1s
Preprocessing train set	0.1s
Preprocessing test set	0.0s
Hyperparameter searching	551.2s
Fitting model	5.2s
Saving model	0.1s
Scoring model	0.9s
Preparing explainability	0.2s

Version Control

- The model was trained at 2023-04-09 07:52:13 (In the DSS server time zone).
- The model was trained with the following version of DSS: 11.3.2
- With the following code environment: DSS builtin environment

ANNEXES

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


EXPAND ALL
RESET
jurisdiction.name is Arizona
state_party_affiliation is D
from_organization.name is Senate
session is 32
classification is ["bill"]
session is 55th-1st-regular
bill_subject_4 is EMERGENCY
Probabilities
🟥 0 : *50.33%*
🟦 1 : *49.67%*
Target classes
🟥 0 : *17.83%*
🟦 1 : *82.17%*
Decision rules
Samples Count
2,978 *(100.00%)*