{{design.mltask.name}}

{{design.visual\_analysis.name}}





|  |  |  |
| --- | --- | --- |
| **Version** | **Author** | **Date** |
| 1.0 | {{config.author.name}}{{config.author.email}} | {{config.generation\_date.name}} |

Table of contents

[I. Executive Summary 4](#_Toc44339452)

# Executive Summary

A {{design.prediction\_type.name}} Machine Learning model was built using Dataiku DSS Visual ML. Its goal is to predict {{design.target.name}} given a total of {{design.features\_count.value}} features. Using a dataset of {{result.train\_set.sample\_rows\_count.value}} rows, the process led to the selection of the {{result.chosen\_algorithm.name}} algorithm.

## Methodology

{{if config.is\_saved\_model.value != Yes}}

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:

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value == Yes}}

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:

{{endif config.is\_saved\_model.value}}

{{design.training\_and\_testing\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.training\_and\_testing\_strategy.table}}

{{if design.training\_and\_testing\_strategy.policy.value != Split the dataset}}

{{design.train\_set.image}}

{{design.test\_set.image}}

{{endif design.training\_and\_testing\_strategy.policy.value}}

See section [II.E](#_l7vf8w6y5ebt) for detailed explanations about these options.

{{if config.is\_saved\_model.value != Yes}}

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

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value == Yes}}

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

{{endif config.is\_saved\_model.value}}

{{design.hyperparameter\_search\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.hyperparameter\_search\_strategy.table}}

See section II.D.3 for detailed explanations about these options.

## Results

{{if design.k\_fold\_cross\_testing.value== Yes}} }}

The {{result.chosen\_algorithm.name}} algorithm was selected. The evaluation metric used to tune the hyperparameters was {{design.test\_metrics.name}} computed on the validation dataset. After the best hyperparameter combination was found, the same metric was also computed on the test datasets. The final value was {{result.test\_metrics.value}}.

{{endif design.k\_fold\_cross\_testing.value}}

{{if design.k\_fold\_cross\_testing.value!= Yes }}

The {{result.chosen\_algorithm.name}} algorithm was selected. The evaluation metric used to tune the hyperparameters was {{design.test\_metrics.name}} 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 {{result.test\_metrics.value}}.

{{endif design.k\_fold\_cross\_testing.value}}

# Methodology

This section deals with the methodological details:

{{if dataset.prepare\_steps.status != No}}

* *Data Preparation* steps may first be applied to generate the initial set of features used in the process and may also transform the target.

{{endif dataset.prepare\_steps.status }}

{{if design.partitioned\_model.status == Yes}}

* The *Problem Definition* consists of selecting the target (**{{design.target.name}}**) and the type of problem ({{design.prediction\_type.name}}) together with the segmentation.

{{endif design.partitioned\_model.status}}

{{if design.partitioned\_model.status != Yes}}

* The *Problem Definition* consists of selecting the target (**{{design.target.name}}**) and the type of problem ({{design.prediction\_type.name}}).

{{endif design.partitioned\_model.status}}

* *Data Ingestion* analyzes each feature in order to maximize its prediction potential.

{{if config.is\_saved\_model.value == Yes}}

* *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.

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value != Yes}}

* *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 {{result.chosen\_algorithm.name}}).

{{endif config.is\_saved\_model.value}}

{{if dataset.prepare\_steps.status != No}}

## Data Preparation

The following data preparation steps are applied to the initial dataset in order to produce the Machine Learning dataset with {{design.features\_count.value}} features and {{result.train\_set.sample\_rows\_count.value}} samples:

{{dataset.prepare\_steps.table}}

|  |
| --- |
|  |
|  |

{{/dataset.prepare\_steps.table}}

{{endif dataset.prepare\_steps.status }}

## Problem Definition

{{if design.partitioned\_model.status == Yes}}

### Target Selection

{{endif design.partitioned\_model.status}}

A {{design.prediction\_type.name}} Machine-Learning model was built using Dataiku DSS. Its goal is to predict **{{design.target.name}}** given a total of {{design.features\_count.value}} features.

{{if design.prediction\_type.name == Binary classification}}

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

{{design.target\_proportion.plot}}

{{endif design.prediction\_type.name}}

{{if design.partitioned\_model.status == Yes}}

### Segmentation

One independent model is made for each partition listed below.

{{result.partitioned.summary.image}}

{{endif design.partitioned\_model.status}}

## 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.

{{design.input\_feature.table}}

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

{{/design.input\_feature.table}}

|  |
| --- |
| **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[[1]](#footnote-1): {{design.feature\_generation\_pairwise\_linear.value}}
* Pairwise polynomial feature generation (A\*B) for all pairs of features[[2]](#footnote-2): {{design.feature\_generation\_pairwise\_polynomial.value}}

{{if design.feature\_generation\_explicit\_pairwise.status != No}}

* 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:

{{design.feature\_generation\_explicit\_pairwise.table}}

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

{{/design.feature\_generation\_explicit\_pairwise.table}}

|  |
| --- |
| **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.
 |

{{endif design.feature\_generation\_explicit\_pairwise.status}}

{{if design.feature\_reduction.name != No reduction}}

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:

{{design.feature\_reduction.image}}

{{endif design.feature\_reduction.name}}

{{if config.is\_saved\_model.value != Yes}}

### 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 {{result.chosen\_algorithm.name}} algorithm has been finally selected.

|  |
| --- |
| {{design.chosen\_algorithm\_search\_strategy.text}} |

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

{{design.chosen\_algorithm\_search\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.chosen\_algorithm\_search\_strategy.table}}

#### Alternative Models

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

{{design.other\_algorithms\_search\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.other\_algorithms\_search\_strategy.table}}

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value == Yes}}

### Tested Algorithm

The {{result.chosen\_algorithm.name}} algorithm has been tested.

|  |
| --- |
| {{design.chosen\_algorithm\_search\_strategy.text}} |

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

{{design.chosen\_algorithm\_search\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.chosen\_algorithm\_search\_strategy.table}}

{{endif config.is\_saved\_model.value}}

### Hyperparameter Search

{{if design.cross\_validation\_strategy.value == K-fold cross-test}}

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 ({{design.test\_metrics.name}}) computed on the validation datasets.

{{endif design.cross\_validation\_strategy.value}}

{{if design.cross\_validation\_strategy.value!= K-fold cross-test}}

{{if design.cross\_validation\_strategy.value == Time-based K-fold (with overlap)}}

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 ({{design.test\_metrics.name}}) computed on the validation datasets.

{{endif design.cross\_validation\_strategy.value}}

{{if design.cross\_validation\_strategy.value != Time-based K-fold (with overlap)}}

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 ({{design.test\_metrics.name}}) computed on the validation dataset.

{{endif design.cross\_validation\_strategy.value}}

{{endif design.cross\_validation\_strategy.value}}

{{if config.is\_saved\_model.value != Yes}}

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

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value == Yes}}

The actual search settings for the selected algorithm are the following:

{{endif config.is\_saved\_model.value}}

{{design.hyperparameter\_search\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.hyperparameter\_search\_strategy.table}}

|  |
| --- |
| **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:

{{design.hyperparameter\_search\_strategy.image}}

{{if design.k\_fold\_cross\_testing.value != Yes}}

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

{{endif design.k\_fold\_cross\_testing.value}}

{{if design.weighting\_strategy.method.name != No weighting}}

### Weighting Strategy

{{if design.weighting\_strategy.method.name == Row weights}}

Row weights are defined by the dataset column {design.weighting\_strategy.method.name} (The weights are positive).

{{endif design.weighting\_strategy.method.name}}

{{if design.weighting\_strategy.method.name == Class weights}}

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

{{endif design.weighting\_strategy.method.name}}

{{if design.weighting\_strategy.method.name == Class and sample weights}}

Weights are defined as the product of sample weights defined in column {design.weighting\_strategy.method.name} (weights are positive) and class weights (inversely proportional to the cardinality of the target class of each row.

{{endif design.weighting\_strategy.method.name}}

{{endif design.weighting\_strategy.method.name}}

### ML Overrides

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

{{result.ml\_overrides.definition.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/result.ml\_overrides.definition.table}}

## 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.

{{design.training\_and\_testing\_strategy.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/design.training\_and\_testing\_strategy.table}}

{{if design.training\_and\_testing\_strategy.policy.value != Split the dataset}}

{{design.train\_set.image}}

{{design.test\_set.image}}

{{endif design.training\_and\_testing\_strategy.policy.value}}

{{if design.training\_and\_testing\_strategy.policy.value == Split the dataset}}

Illustration:

{{design.sampling\_and\_splitting.image}}

{{endif design.training\_and\_testing\_strategy.policy.value}}

|  |
| --- |
| **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 {{result.train\_set.sample\_rows\_count.value}} 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.

{{if design.partitioned\_model.status != Yes}}

{{if config.is\_saved\_model.value != Yes}}

## Selected Model

{{result.chosen\_algorithm.name}} was finally selected by the user with the optimal set of hyperparameters given in the table below:

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value == Yes}}

{{if result.hyperparameter\_search.status != No}}

## Selected Model

{{endif result.hyperparameter\_search.status}}

The optimal set of hyperparameters for the selected algorithm {{result.chosen\_algorithm.name}} is given in the table below:

{{endif config.is\_saved\_model.value}}

{{result.chosen\_algorithm\_details.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/result.chosen\_algorithm\_details.table}}

See section II.D.2.a) for detailed explanations on the algorithm and its hyperparameters.

{{if config.is\_saved\_model.value == Yes}}

{{if result.hyperparameter\_search.status != No}}

## 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:

{{result.hyperparameter\_search.plot}}

The table below lists all the performed trainings:

{{result.hyperparameter\_search.table}}

|  |  |
| --- | --- |
| Header | Header |
| Content | Content |

{{/result.hyperparameter\_search.table}}

{{endif result.hyperparameter\_search.status}}

{{endif config.is\_saved\_model.value}}

{{if config.is\_saved\_model.value != Yes}}

## Alternative Models

{{if result.hyperparameter\_search.status != No}}

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:

{{result.hyperparameter\_search.plot}}

The table below lists all the performed trainings:

{{result.hyperparameter\_search.table}}

|  |  |
| --- | --- |
| Header | Header |
| Content | Content |

{{/result.hyperparameter\_search.table}}

{{endif result.hyperparameter\_search.status}}

{{endif design.partitioned\_model.status}}

The selected algorithm was compared with other algorithms. The table below gives the performance obtained with the combination of hyperparameters that optimizes the {{design.test\_metrics.name}} metric:

{{result.leaderboard.image}}

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

{{if design.prediction\_type.name == Binary classification}}

|  |  |
| --- | --- |
| Accuracy | {{result.leaderboard.accuracy.image}} |
| Precision | {{result.leaderboard.precision.image}} |
| Recall | {{result.leaderboard.recall.image}} |
| F1 Score | {{result.leaderboard.f1.image}} |
| Cost Matrix Gain | {{result.leaderboard.cost\_matrix.image}} |
| Log Loss | {{result.leaderboard.log\_loss.image}} |
| ROC AUC | {{result.leaderboard.roc\_auc.image}} |
| Average Precision | {{result.leaderboard.average\_precision.image}} |
| Calibration Loss | {{result.leaderboard.calibration\_loss.image}} |
| Lift | {{result.leaderboard.lift.image}} |

{{endif design.prediction\_type.name}}

{{if design.prediction\_type.name == Multiclass classification}}

|  |  |
| --- | --- |
| Accuracy | {{result.leaderboard.accuracy.image}} |
| Precision | {{result.leaderboard.precision.image}} |
| Recall | {{result.leaderboard.recall.image}} |
| F1 Score | {{result.leaderboard.f1.image}} |
| Log Loss | {{result.leaderboard.log\_loss.image}} |
| ROC AUC | {{result.leaderboard.roc\_auc.image}} |
| Average Precision | {{result.leaderboard.average\_precision.image}} |
| Calibration Loss | {{result.leaderboard.calibration\_loss.image}} |

{{endif design.prediction\_type.name}}

{{if design.prediction\_type.name == Regression}}

|  |  |
| --- | --- |
| EVS | {{result.leaderboard.evs.image}} |
| MAPE | {{result.leaderboard.mape.image}} |
| MAE | {{result.leaderboard.mae.image}} |
| MSE | {{result.leaderboard.mse.image}} |
| RMSE | {{result.leaderboard.rmse.image}} |
| RMSLE | {{result.leaderboard.rmsle.image}} |
| R2 score | {{result.leaderboard.r2.image}} |
| Pearson coeff. | {{result.leaderboard.correlation.image}} |

{{endif design.prediction\_type.name}}

{{endif config.is\_saved\_model.value}}

# Selected Model Results

## Selected Model Metrics

{{if design.prediction\_type.name != Regression}}

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:

{{result.confusion\_matrix.image}}

{{endif design.prediction\_type.name}}

{{if design.partitioned\_model.status != Yes}}

{{if design.prediction\_type.name == Binary classification}}

A classifier produces a probability that a given object belongs to the “positive” class (**{{result.target\_value.positive\_class.value}}**). 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 {{result.classification\_threshold.current.value}}. The optimal value according to the {{result.threshold\_metric.name}} is {{result.classification\_threshold.optimal.value}}.

{{if design.k\_fold\_cross\_testing.value== Yes}}

As K-fold cross-testing has been used, the confusion matrix is computed on the 1st fold only.

{{endif design.k\_fold\_cross\_testing.value}}

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

{{result.confusion\_matrix\_metrics.plot}}

|  |
| --- |
| **Legend*** *Precision*: Proportion of correct predictions among “positive” (**{{result.target\_value.positive\_class.value}}**) predictions.
* *Recall*: Proportion of actually “positive” **({{result.target\_value.positive\_class.value}}**) 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.
 |

{{if design.k\_fold\_cross\_testing.value== Yes}}

As K-fold cross-testing has been used, the displayed metrics have been averaged over all test datasets (folds).

{{endif design.k\_fold\_cross\_testing.value}}

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

|  |
| --- |
| {{result.cost\_matrix.image}} |

{{endif design.prediction\_type.name}}

{{endif design.partitioned\_model.status}}

{{if design.k\_fold\_cross\_testing.value== Yes}}

Finally, the detailed metrics obtained on the test datasets (folds) are given below. As K-fold cross-testing was chosen, the average over the test datasets (folds) is given as well as a confidence interval.

{{endif design.k\_fold\_cross\_testing.value}}

{{if design.k\_fold\_cross\_testing.value!= Yes}}

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

{{endif design.k\_fold\_cross\_testing.value}}

{{result.detailed\_metrics.table}}

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

{{/result.detailed\_metrics.table}}

The ml assertions metrics are given below.

{{result.ml\_assertions.table}}

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

{{/result.ml\_assertions.table}}

The ml override metrics are given below.

{{result.ml\_overrides.metrics.table}}

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

{{/result.ml\_overrides.metrics.table}}

## Selected Model Performance Charts

{{if design.partitioned\_model.status != Yes}}

{{if design.prediction\_type.name == Binary classification}}

### Lift Charts

A binary classifier produces a probability that a given record is “positive” (Here {{result.target\_value.positive\_class.value}}). 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...)

{{result.lift\_curve.plot}}

|  |
| --- |
| **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 {{result.target\_value.positive\_class.value}}).* 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.

{{if design.k\_fold\_cross\_testing.value == Yes}}* The cumulative gain curve shows the actual percentage of actually positive records found by this model. The steeper the curve, the better. As K-fold cross-testing has been chosen, one curve per fold is displayed.

{{endif design.k\_fold\_cross\_testing.value}}­{{if design.k\_fold\_cross\_testing.value != Yes}}}}* The cumulative gain curve shows the actual percentage of actually positives found by this model. The steeper the curve, the better.

{{endif design.k\_fold\_cross\_testing.value}}**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 {{result.target\_value.positive\_class.value}}) 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.

{{result.decision\_chart.plot}}

{{endif design.prediction\_type.name}}

{{if design.prediction\_type.name != Regression}}

### 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.

{{if design.prediction\_type.name != Binary classification}}

There is one ROC curve per class, based on the “one class vs. all other classes” binary classification problem.

{{endif design.prediction\_type.name}}

{{result.roc\_curve.plot}}

### Density Chart

{{if design.prediction\_type.name == Binary classification}}

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 {{result.target\_value.positive\_class.value}}). 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 ({{result.target\_value.positive\_class.value}}) should be entirely on the right.
* The density function of the “negative” class ({{result.target\_value.negative\_class.value}}) should be entirely on the left.

{{endif design.prediction\_type.name}}

{{if design.prediction\_type.name != Binary classification}}

For each class, the density chart illustrates how the model succeeds in recognizing (and separating) the class among the other classes. It shows the probability distribution of the actual classes in the test set given the predicted probability of being of the class. The two density functions show the probability density of rows in the test set that actually belongs to the 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 class should be entirely on the right.
* The density function of the aggregation of the other classes should be entirely on the left.

{{endif design.prediction\_type.name}}

The dotted vertical lines mark the medians.

{{result.density\_chart.plot}}

{{if design.k\_fold\_cross\_testing.value == Yes}}

As K-fold cross-testing was chosen, the results are presented for the first fold only.

{{endif design.k\_fold\_cross\_testing.value}}

### Calibration

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

{{if design.prediction\_type.name != Binary classification}}

There is one calibration curve per class, based on the probability that a row is a given class vs. being of any other class.

{{endif design.prediction\_type.name}}

{{result.calibration.plot}}

|  |
| --- |
| 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). |

{{endif design.prediction\_type.name}}

{{endif design.partitioned\_model.status}}

{{if design.prediction\_type.name == Regression}}

{{if design.partitioned\_model.status != Yes}}

The error distribution table for this regression model is given below as a table with some statistics, as well as a histogram and as a scatter plot.

{{endif design.partitioned\_model.status}}

{{if design.partitioned\_model.status == Yes}}

The error distribution table for this regression model is given below as a table with some statistics.

{{endif design.partitioned\_model.status}}

{{result.error\_distribution.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/result.error\_distribution.table}}

{{if design.partitioned\_model.status != Yes}}

The errors (the difference between predicted and actual values) should be centered around zero, and the distribution should be “narrow”, i.e., the spread of the error should be limited. More generally, the errors should be “normally” distributed around zero (the curve should look like a bell).

To reduce the effect of possible spurious outliers, the error distribution is winsorized (clipped) at the 2nd and 98th percentiles.

{{result.error\_distribution.plot}}

{{result.scatter.plot}}

{{endif design.partitioned\_model.status}}

{{endif design.prediction\_type.name}}

{{if design.partitioned\_model.status != Yes}}

## Sensitivity Testing and Analysis

{{if result.absolute\_importance.status != No}}

Shapley feature importance has been computed representing which features have the strongest impact on the predictions of the algorithm.

{{result.absolute\_importance.plot}}

Feature effects display multiple Shapley values computed per feature.

{{result.feature\_effects.plot}}

{{endif result.absolute\_importance.status}}

{{if result.feature\_importance.status != No}}

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

{{result.feature\_importance.plot}}

{{endif result.feature\_importance.status}}

{{if result.regression\_coefficients.status == Yes}}

The selected algorithm is linear so that its coefficients are particularly meaningful to understand the features that have a significant impact on its performance.

|  |
| --- |
| {{result.regression\_coefficients.image}} |

{{endif result.regression\_coefficients.status}}

{{endif design.partitioned\_model.status}}

## 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.

{{result.diagnostics.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/result.diagnostics.table}}

# Deployment and Monitoring

## Implementation Details

* The backend used by the model is: {{design.backend.name}}
* The model can be found here: {{config.project.link}}
* The name of the generated file is: {{config.output\_file.name}}
* The timing of the training was the following:

{{result.timings.table}}

|  |  |
| --- | --- |
|  |  |
|  |  |

{{/result.timings.table}}

## Version Control

* The model was trained at {{result.training\_date.name}} (In the DSS server time zone).
* The model was trained with the following version of DSS: {{config.dss.version}}
* With the following code environment: {{config.environment.name}}

{{if result.decision\_trees.status == Yes}}

# Annexes

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

{{result.decision\_trees.image}}

{{endif result.decision\_trees.status}}

1. For each pair of features, two new features (A+B and A-B) are generated. [↑](#footnote-ref-1)
2. For each pair of features, one new feature (A\*B) is generated. [↑](#footnote-ref-2)