{{design.mltask.name}}

{{mlflow.model\_version.name}}

short line



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

Table of contents

[I. Executive Summary 3](#_Toc117170268)

[II. Selected Model Performance Charts 4](#_Toc117170269)

[1. Lift Charts 4](#_Toc117170270)

[2. Decision Chart 5](#_Toc117170271)

[3. ROC Curve 5](#_Toc117170272)

[4. Density Chart 5](#_Toc117170273)

[5. Calibration 6](#_Toc117170274)

[6. Error Distribution 7](#_Toc117170275)

[B. Sensitivity Testing and Analysis 8](#_Toc117170277)

[C. Diagnostics 8](#_Toc117170278)

[III. Deployment and Monitoring 9](#_Toc117170279)

[A. Implementation Details 9](#_Toc117170280)

[B. Version Control 9](#_Toc117170281)

# Executive Summary

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

# MLflow Import details

Information about how the model was created and imported can be found in this section.

## MLflow Import Environment

The model was imported with the following environment:

* Code environment: {{config.environment.name}}
* Python version: {{mlflow.python\_version.name}}

## Pyfunc Flavor Labels

Below is the list of pyfunc flavor labels describing the environment in which the model was trained and imported.

{{mlflow.pyfunc.table}}

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

{{/mlflow.pyfunc.table}}

## Other Labels

Below is the list of other labels describing the libraries used to create the model.

{{mlflow.labels.table}}

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

{{/mlflow.labels.table}}

## Features

The table below lists the features set by the user when the model was imported:

{{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…) |

{{if mlflow.origin.name == experiment}}

## Original Experiment

The model version {{mlflow.model\_version.name}} has been designed in the run {{mlflow.experiment.runId}} of experiment {{mlflow.experiment.experimentId}}.

{{endif mlflow.origin.name}}

# Model Results

## 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.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}}.

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}}

{{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}}

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

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

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

## Model Performance Charts

{{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.   **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}}

### 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}}

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

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.

{{result.error\_distribution.table}}

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

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

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.prediction\_type.name}}

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

## Sensitivity Testing and Analysis

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}}

# 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}}

## Version Control

* The model was trained at {{mlflow.mlflow\_training\_date.name}} (UTC).
* It was then imported at {{result.mlflow\_import\_date.name}} (in the DSS time zone)
* With the following code environment: {{config.environment.name}}
* And the following python version: {{mlflow.python\_version.name}}