

---

# **scikit-uplift**

***Release 0.1.2***

**May 02, 2020**



---

## Contents

---

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Project info</b>	<b>5</b>
2.1	Installation . . . . .	5
2.2	Quick Start . . . . .	5
2.3	API . . . . .	7
2.4	Tutorials . . . . .	19
2.5	Release History . . . . .	20
2.6	Hall of Fame . . . . .	21
<b>3</b>	<b>Papers and materials</b>	<b>23</b>
<b>4</b>	<b>Tags</b>	<b>25</b>
<b>5</b>	<b>Indices and tables</b>	<b>27</b>
	<b>Python Module Index</b>	<b>29</b>
	<b>Index</b>	<b>31</b>



**scikit-uplift (sklift)** is a Python module for basic approaches of uplift modeling built on top of scikit-learn.

Uplift prediction aims to estimate the causal impact of a treatment at the individual level.

More about uplift modelling problem read in russian on habr.com: [Part 1](#) and [Part 2](#).



- Comfortable and intuitive style of modelling like scikit-learn;
- Applying any estimator adheres to scikit-learn conventions;
- All approaches can be used in sklearn.pipeline (see example ([EN](#) , [RU](#) ))
- Almost all implemented approaches solve both the problem of classification and regression;
- A lot of metrics (Such as *Area Under Uplift Curve* or *Area Under Qini Curve*) are implemented to evaluate your uplift model;
- Useful graphs for analyzing the built model.

**The package currently supports the following methods:**

1. Solo Model (aka Treatment Dummy) approach
2. Class Transformation (aka Class Variable Transformation or Revert Label) approach
3. Two Models (aka naïve approach, or difference score method, or double classifier approach) approach, including Dependent Data Representation

**And the following metrics:**

1. Uplift@k
2. Area Under Uplift Curve
3. Area Under Qini Curve





- GitHub repository: <https://github.com/maks-sh/scikit-uplift>
- Github examples: <https://github.com/maks-sh/scikit-uplift/tree/master/notebooks>
- License: MIT

## 2.1 Installation

**Install** the package by the following command from [PyPI](#):

```
pip install scikit-uplift
```

Or install from [source](#):

```
git clone https://github.com/maks-sh/scikit-uplift.git
cd scikit-uplift
python setup.py install
```

## 2.2 Quick Start

See the **RetailHero tutorial notebook** ([EN](#) , [RU](#) ) for details.

**Train and predict your uplift model**

```
# import approaches
from sklift.models import SoloModel, ClassTransformation, TwoModels
# import any estimator adheres to scikit-learn conventions.
from catboost import CatBoostClassifier

# define approach
```

(continues on next page)

(continued from previous page)

```
sm = SoloModel(CatBoostClassifier(verbose=100, random_state=777))
# fit model
sm = sm.fit(X_train, y_train, treat_train, estimator_fit_params={'plot': True})

# predict uplift
uplift_sm = sm.predict(X_val)
```

### Evaluate your uplift model

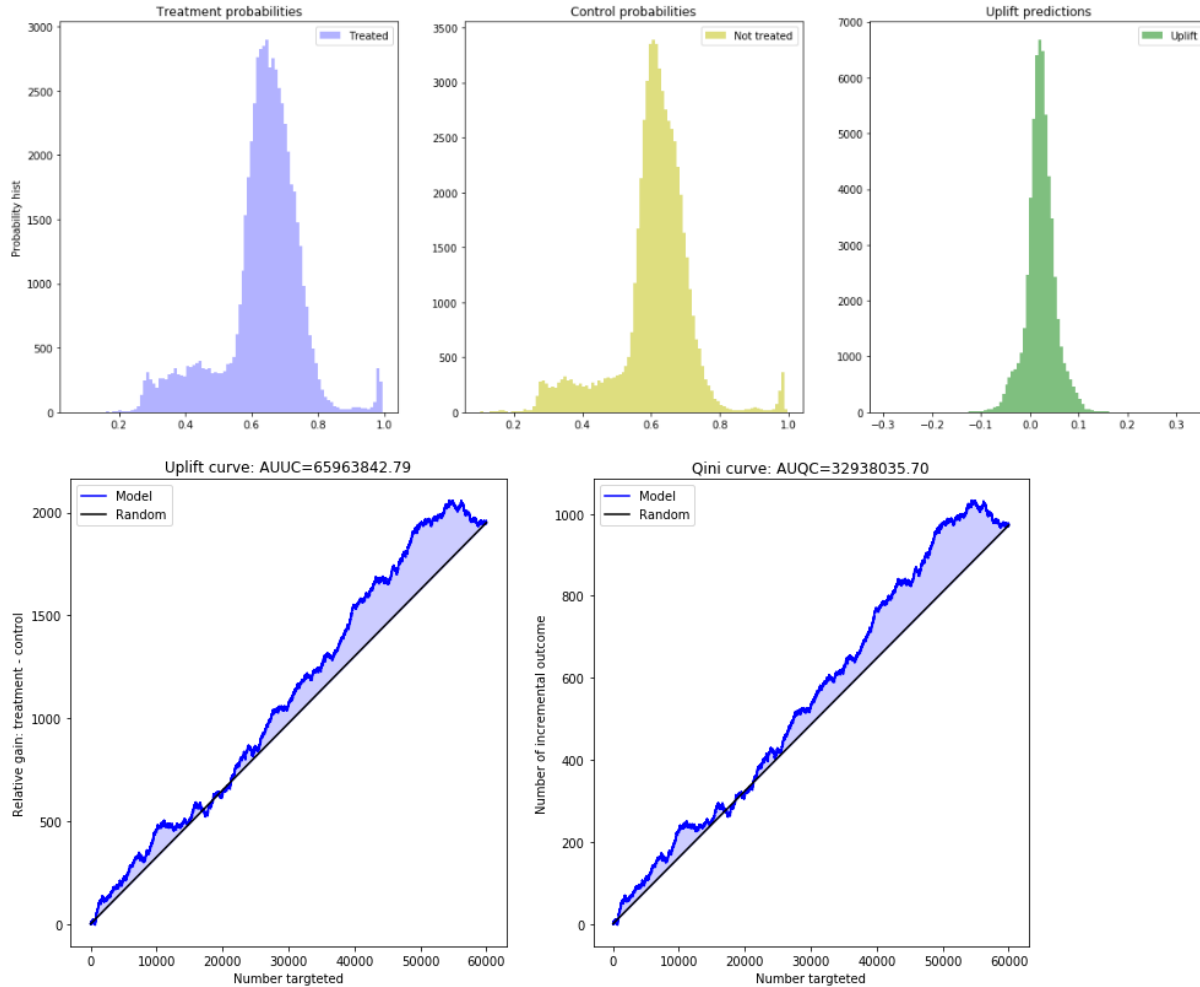
```
# import metrics to evaluate your model
from sklift.metrics import qini_auc_score, uplift_auc_score, uplift_at_k
# Uplift@30%
sm_uplift_at_k = uplift_at_k(y_true=y_val, uplift=uplift_sm, treatment=treat_val, k=0.
↪3)
# Area Under Qini Curve
sm_qini_auc_score = qini_auc_score(y_true=y_val, uplift=uplift_sm, treatment=treat_
↪val)
# Area Under Uplift Curve
sm_uplift_auc_score = uplift_auc_score(y_true=y_val, uplift=uplift_sm,
↪treatment=treat_val)
```

### Vizualize the results

```
# import vizualisation tools
from sklift.viz import plot_uplift_preds, plot_uplift_qini_curves

# get conditional predictions (probabilities) of performing a target action
# with interaction for each object
sm_trmnt_preds = sm.trmnt_preds_
# get conditional predictions (probabilities) of performing a target action
# without interaction for each object
sm_ctrl_preds = sm.ctrl_preds_

# draw probability distributions and their difference (uplift)
plot_uplift_preds(trmnt_preds=sm_trmnt_preds, ctrl_preds=sm_ctrl_preds);
# draw Uplift and Qini curves
plot_uplift_qini_curves(y_true=y_val, uplift=uplift_sm, treatment=treat_val);
```



## 2.3 API

### 2.3.1 Models (sklift.models)

#### 1. Approaches with the same model

##### 1.1 One model with treatment as feature

The simplest and most intuitive solution: the model is trained on union of two groups, with the binary communication flag acting as an additional feature. Each object from the test sample is scored twice: with the communication flag equal to 1 and equal to 0. Subtracting the probabilities for each observation, we get the required uplift.

The training process:

$$\text{fit} \left( \begin{array}{ccc|c|c} x_{11} & \cdots & x_{1k} & w_1 & y_1 \\ \vdots & \ddots & \vdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nk} & w_n & y_n \end{array} \right)$$

$X_{\text{train}} \quad W_{\text{train}} \quad Y_{\text{train}}$

The process of applying the model:

$$\text{predict} \left( \begin{array}{ccc|c} x_{11} & \cdots & x_{1k} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mk} & 1 \end{array} \right) - \text{predict} \left( \begin{array}{ccc|c} x_{11} & \cdots & x_{1k} & 0 \\ \vdots & \ddots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mk} & 0 \end{array} \right) = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix}$$

$X_{\text{test}} \quad W_1 \quad X_{\text{test}} \quad W_0 \quad \text{uplift}$

**class** sklift.models.models.SoloModel (estimator)

aka Treatment Dummy approach, or Single model approach, or S-Learner.

Fit solo model on whole dataset with 'treatment' as an additional feature.

For each test example calculate predictions on new set twice: with treatment == '1' and with treatment == '0'.  
After that calculate uplift as a delta between these predictions.

Return delta of predictions for each example.

See more details about SoloModel in [documentation](#).

**Parameters estimator** (estimator object implementing 'fit') – The object to use to fit the data.

**trmnt\_preds\_**

Estimator predictions on samples when treatment.

**Type** array-like, shape (n\_samples, )

**ctrl\_preds\_**

Estimator predictions on samples when control.

**Type** array-like, shape (n\_samples, )

Example:

```
# import approach
from sklift.models import SoloModel
# import any estimator adheres to scikit-learn conventions
from catboost import CatBoostClassifier

sm = SoloModel(CatBoostClassifier(verbose=100, random_state=777)) # define_
→ approach
sm = sm.fit(X_train, y_train, treat_train, estimator_fit_params={'plot': True})
→ # fit the model
uplift_sm = sm.predict(X_val) # predict uplift
```

## References

Lo, Victor. (2002). The True Lift Model - A Novel Data Mining Approach to Response Modeling in Database Marketing. SIGKDD Explorations. 4. 78-86.

**fit** (*X*, *y*, *treatment*, *estimator\_fit\_params=None*)

Fit the model according to the given training data.

For each test example calculate predictions on new set twice: by the first and second models. After that calculate uplift as a delta between these predictions.

Return delta of predictions for each example.

### Parameters

- **X** (*array-like*, *shape* (*n\_samples*, *n\_features*)) – Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.
- **y** (*array-like*, *shape* (*n\_samples*,)) – Target vector relative to X.
- **treatment** (*array-like*, *shape* (*n\_samples*,)) – Binary treatment vector relative to X.
- **estimator\_fit\_params** (*dict*, *optional*) – Parameters to pass to the fit method of the estimator.

**Returns** self

**Return type** object

**predict** (*X*)

Perform uplift on samples in X.

**Parameters** **X** (*array-like*, *shape* (*n\_samples*, *n\_features*)) – Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**Returns** uplift

**Return type** array (*shape* (*n\_samples*,))

## 1.2 Class Transformation

**Warning:** This approach is only suitable for classification problem

Quite an interesting and mathematically confirmed approach to the construction of the model, presented back in 2012. The method is to predict a slightly changed target:

$$z_i = y_i * w_i + (1 - y_i) * (1 - w_i),$$

- $z_i$  - new target for  $i$  customer;
- $y_i$  - old target  $i$  customer;
- $w_i$  - treatment flag  $i$  customer.

In other words, the new class is 1 if we know that on a particular observation, the result in the interaction would be as good as in the control group if we could know the result in both groups:

$$z_i = \begin{cases} 1, & \text{if } w_i = 1 \text{ and } y_i = 1 \\ 1, & \text{if } w_i = 0 \text{ and } y_i = 0 \\ 0, & \text{otherwise} \end{cases}$$

Let's describe in more detail what is the probability of a new target variable:

$$\begin{aligned}
 P(Z = 1|X_1, \dots, X_m) &= \\
 &= P(Z = 1|X_1, \dots, X_m, W = 1) * P(W = 1|X_1, \dots, X_m,) + \\
 &+ P(Z = 1|X_1, \dots, X_m, W = 0) * P(W = 0|X_1, \dots, X_m,) = \\
 &= P(Y = 1|X_1, \dots, X_m, W = 1) * P(W = 1|X_1, \dots, X_m,) + \\
 &+ P(Y = 0|X_1, \dots, X_m, W = 0) * P(W = 0|X_1, \dots, X_m,).
 \end{aligned}$$

We assume that  $W$  does not depend on the attributes of  $X_1, \dots, X_m$ , because otherwise the experiment design is not very well designed. Taking this, we have:  $P(W|X_1, \dots, X_m,) = P(W)$  and

$$\begin{aligned}
 P(Z = 1|X_1, \dots, X_m) &= \\
 &= P^T(Y = 1|X_1, \dots, X_m) * P(W = 1) + \\
 &+ P^C(Y = 0|X_1, \dots, X_m) * P(W = 0).
 \end{aligned}$$

Also assume that  $P(W = 1) = P(W = 0) = \frac{1}{2}$ , i.e. during the experiment, the control and treatment groups were divided in equal proportions. Then we get the following:

$$\begin{aligned}
 P(Z = 1|X_1, \dots, X_m) &= \\
 &= P^T(Y = 1|X_1, \dots, X_m) * \frac{1}{2} + P^C(Y = 0|X_1, \dots, X_m) * \frac{1}{2} \Rightarrow \\
 &\Rightarrow 2 * P(Z = 1|X_1, \dots, X_m) = \\
 &= P^T(Y = 1|X_1, \dots, X_m) + P^C(Y = 0|X_1, \dots, X_m) = \\
 &= P^T(Y = 1|X_1, \dots, X_m) + 1 - P^C(Y = 1|X_1, \dots, X_m) \Rightarrow \\
 &\Rightarrow P^T(Y = 1|X_1, \dots, X_m) - P^C(Y = 1|X_1, \dots, X_m) = \\
 &= UPLIFT = 2 * P(Z = 1|X_1, \dots, X_m) - 1
 \end{aligned}$$

Thus, by doubling the forecast of the new target and subtracting one from it, we get the value of the uplift itself, i.e.

$$UPLIFT = 2 * P(Z = 1) - 1$$

Based on the assumption described above:  $P(W = 1) = P(W = 0) = \frac{1}{2}$ , this approach should be used only in cases where the number of clients with whom we have communicated is equal to the number of clients with whom there was no communication.

**class** `sklift.models.models.ClassTransformation` (*estimator*)  
aka Class Variable Transformation or Revert Label approach.

Redefine target variable, which indicates that treatment make some impact on target or did target is negative without treatment.

$$Z = Y * W + (1 - Y)(1 - W),$$

where  $Y$  - target,  $W$  - communication flag.

Then,  $Uplift \sim 2 * (Z == 1) - 1$

Returns only uplift predictions.

See more details about [ClassTransformation in documentation](#).

**Parameters** `estimator` (*estimator object implementing 'fit'*) – The object to use to fit the data.

Example:

```
# import approach
from sklift.models import ClassTransformation
# import any estimator adheres to scikit-learn conventions
from catboost import CatBoostClassifier

ct = ClassTransformation(CatBoostClassifier(verbose=100, random_state=777)) #_
↪define approach
ct = ct.fit(X_train, y_train, treat_train, estimator_fit_params={'plot': True})
↪# fit the model
uplift_ct = ct.predict(X_val) # predict uplift
```

## References

Maciej Jaskowski and Szymon Jaroszewicz. Uplift modeling for clinical trial data. ICML Workshop on Clinical Data Analysis, 2012.

**fit** (*X*, *y*, *treatment*, *estimator\_fit\_params=None*)  
Fit the model according to the given training data.

### Parameters

- **X** (*array-like*, *shape* (*n\_samples*, *n\_features*)) – Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.
- **y** (*array-like*, *shape* (*n\_samples*,)) – Target vector relative to X.
- **treatment** (*array-like*, *shape* (*n\_samples*,)) – Binary treatment vector relative to X.
- **estimator\_fit\_params** (*dict*, *optional*) – Parameters to pass to the fit method of the estimator.

**Returns** *self*

**Return type** *object*

**predict** (*X*)  
Perform uplift on samples in X.

**Parameters** **X** (*array-like*, *shape* (*n\_samples*, *n\_features*)) – Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**Returns** *uplift*

**Return type** *array* (*shape* (*n\_samples*,))

## 2. Approaches with two models

The two-model approach can be found in almost any uplift modeling work, and is often used as a baseline. However, the use of two models can lead to some unpleasant consequences: if the training will be used fundamentally different models or the nature of the data of the test and control groups will be very different, then the returned models will not be comparable with each other. As a result, the calculation of the uplift will not be completely correct. To avoid this effect, it is necessary to calibrate the models so that their scores can be interpolated as probabilities. Calibration of model probabilities is well described in [the scikit-learn documentation](#).

## 2.1 Two independent models

---

**Hint:** In skift this approach corresponds to the *TwoModels* class and the **vanilla** method.

---

As the name implies, the approach is to model the conditional probabilities of the treatment and control groups separately. The articles argue that this approach is rather weak, since both models focus on predicting the result separately and can therefore skip the “weaker” differences in the samples.

The training process:

$$\begin{array}{c}
 model^T = fit \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{p1} & \cdots & x_{pk} \end{array}, \begin{array}{c} y_1 \\ \cdots \\ y_p \end{array} \right), \quad model^C = fit \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{q1} & \cdots & x_{qk} \end{array}, \begin{array}{c} y_1 \\ \cdots \\ y_q \end{array} \right) \\
 \begin{array}{ccc} X_{train\_treat} & Y_{train\_treat} & \\ & & X_{train\_control} \quad Y_{train\_control} \end{array}
 \end{array}$$

The process of applying the model:

$$\begin{array}{c}
 model^T \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mk} \end{array} \right) - model^C \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mk} \end{array} \right) = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} \\
 \begin{array}{ccc} X_{test} & & \\ & X_{test} & uplift \end{array}
 \end{array}$$

## 2.2 Two dependent models

The dependent data representation approach is based on the classifier chain method originally developed for multi-class classification problems. The idea is that if there are  $L$  different labels, you can build  $L$  different classifiers, each of which solves the problem of binary classification and in the learning process, each subsequent classifier uses the predictions of the previous ones as additional features. The authors of this method proposed to use the same idea to solve the problem of uplift modeling in two stages.

---

**Hint:** In skift this approach corresponds to the *TwoModels* class and the **ddr\_control** method.

---

At the beginning we train the classifier based on control data:

$$P^C = P(Y = 1 | X, W = 0),$$

then we will perform the  $P_C$  predictions as a new feature for training the second classifier on test data, thus effectively introducing a dependency between the two data sets:

$$P^T = P(Y = 1 | X, P_C(X), W = 1)$$

To get the uplift for each observation, calculate the difference:

$$uplift(x_i) = P^T(x_i, P_C(x_i)) - P^C(x_i)$$



Intuitively, the second classifier studies the difference between the expected result in the test and the control, i.e. the uplift itself.

The training process:

$$\begin{aligned}
 model^c &= fit \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{q1} & \cdots & x_{qk} \end{array}, \begin{array}{c} y_1 \\ \vdots \\ y_q \end{array} \right), \\
 &\quad \quad \quad X_{train\_control} \quad Y_{train\_control} \\
 P^c &= \begin{array}{c} model^c \\ predict \\ proba \end{array} \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{p1} & \cdots & x_{pk} \end{array} \right), \quad model^T = fit \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} & P_1^c & y_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{p1} & \cdots & x_{pk} & P_p^c & y_p \end{array}, \dots \right), \\
 &\quad \quad \quad X_{train\_treat} \quad \quad \quad X_{train\_treat} \quad P^c \quad Y_{train\_treat}
 \end{aligned}$$

The process of applying the model:

$$\begin{aligned}
 &\begin{array}{c} model^T \\ predict \\ proba \end{array} \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mk} \end{array} \right) - \begin{array}{c} model^c \\ predict \\ proba \end{array} \left( \begin{array}{ccc} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mk} \end{array} \right) = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} \\
 &\quad \quad \quad X_{test} \quad \quad \quad P^c(X_{test}) \quad \quad \quad X_{test} \quad \quad \quad uplift
 \end{aligned}$$

Similarly, you can first train the  $P_T$  classifier and then use its predictions as a trait for the  $P_C$  classifier.

---

**Hint:** In sklift this approach corresponds to the *TwoModels* class and the **ddr\_treatment** method.

---

**class** sklift.models.models.**TwoModels** (*estimator\_trmnt*, *estimator\_ctrl*, *method*='vanilla')  
 aka naïve approach, or difference score method, or double classifier approach.

Fit two separate models: on the treatment data and on the control data.

See more details about [TwoModels](#) in [documentation](#).

#### Parameters

- **estimator\_trmnt** (*estimator object implementing 'fit'*) – The object to use to fit the treatment data.
- **estimator\_ctrl** (*estimator object implementing 'fit'*) – The object to use to fit the control data.
- **method** (*string, 'vanilla', 'ddr\_control' or 'ddr\_treatment', default='vanilla'*) – Specifies the approach:
  - **'vanilla'**: Two independent models;
  - **'ddr\_control'**: Dependent data representation (First train control estimator).
  - **'ddr\_treatment'**: Dependent data representation (First train treatment estimator).

**trmnt\_preds\_**  
 Estimator predictions on samples when treatment.

**Type** array-like, shape (n\_samples, )

**ctrl\_preds\_**

Estimator predictions on samples when control.

**Type** array-like, shape (n\_samples, )

Example:

```
# import approach
from sklift.models import TwoModels
# import any estimator adheres to scikit-learn conventions
from catboost import CatBoostClassifier

estimator_trmnt = CatBoostClassifier(silent=True, thread_count=2, random_state=42)
estimator_ctrl = CatBoostClassifier(silent=True, thread_count=2, random_state=42)

# define approach
tm_ctrl = TwoModels(
    estimator_trmnt=estimator_trmnt,
    estimator_ctrl=estimator_ctrl,
    method='ddr_control'
)

# fit the models
tm_ctrl = tm_ctrl.fit(
    X_train, y_train, treat_train,
    estimator_trmnt_fit_params={'cat_features': cat_features},
    estimator_ctrl_fit_params={'cat_features': cat_features}
)
uplift_tm_ctrl = tm_ctrl.predict(X_val) # predict uplift
```

**References** Betlei, Artem & Diemert, Eustache & Amini, Massih-Reza. (2018). Uplift Prediction with Dependent Feature Representation in Imbalanced Treatment and Control Conditions: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13–16, 2018, Proceedings, Part V. 10.1007/978-3-030-04221-9\_5.

Zhao, Yan & Fang, Xiao & Simchi-Levi, David. (2017). Uplift Modeling with Multiple Treatments and General Response Types. 10.1137/1.9781611974973.66.

**fit** (*X*, *y*, *treatment*, *estimator\_trmnt\_fit\_params=None*, *estimator\_ctrl\_fit\_params=None*)

Fit the model according to the given training data.

For each test example calculate predictions on new set twice: by the first and second models. After that calculate uplift as a delta between these predictions.

Return delta of predictions for each example.

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Training vector, where n\_samples is the number of samples and n\_features is the number of features.
- **y** (array-like, shape (n\_samples,)) – Target vector relative to X.
- **treatment** (array-like, shape (n\_samples,)) – Binary treatment vector relative to X.
- **estimator\_trmnt\_fit\_params** (dict, optional) – Parameters to pass to the fit method of the treatment estimator.

- **estimator\_ctrl\_fit\_params** (*dict*, *optional*) – Parameters to pass to the fit method of the control estimator.

**Returns** *self*

**Return type** *object*

**predict** (*X*)

Perform uplift on samples in *X*.

**Parameters** *X* (*array-like*, *shape* (*n\_samples*, *n\_features*)) – Training vector, where *n\_samples* is the number of samples and *n\_features* is the number of features.

**Returns** *uplift*

**Return type** *array* (*shape* (*n\_samples*,))

## 2.3.2 Metrics (sklift.metrics)

`sklift.metrics.metrics.auqc(y_true, uplift, treatment)`

Compute Area Under the Qini Curve (aka Qini coefficient) from prediction scores.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uplift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.

**Returns** Area Under the Qini Curve.

**Return type** *float*

**Warning:** Metric *auqc* was renamed to *qini\_auc\_score()* in version 0.1.0 and will be removed in 0.2.0

`sklift.metrics.metrics.auuc(y_true, uplift, treatment)`

Compute Area Under the Uplift Curve from prediction scores.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uplift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.

**Returns** Area Under the Uplift Curve.

**Return type** *float*

**Warning:** Metric *auuc* was renamed to *uplift\_auc\_score()* in version 0.1.0 and will be removed in 0.2.0

`sklift.metrics.metrics.qini_auc_score(y_true, uplift, treatment)`

Compute Area Under the Qini Curve (aka Qini coefficient) from prediction scores.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.

- **uplift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.

**Returns** Area Under the Qini Curve.

**Return type** float

`sklift.metrics.metrics.qini_curve(y_true, uplift, treatment)`  
Compute Qini curve.

This is a general function, given points on a curve. For computing the area under the Qini Curve, see `qini_auc_score()`.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uplift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.

**Returns** Points on a curve.

**Return type** array (shape = [ $>2$ ]), array (shape = [ $>2$ ])

**See also:**

`qini_auc_score()`: Compute the area under the Qini curve.

`plot_uplift_qini_curves()`: Plot Uplift and Qini curves.

`sklift.metrics.metrics.response_rate_by_percentile(y_true, uplift, treatment, group, strategy, bins=10)`

Compute response rate and its variance at each percentile.

Response rate is a target mean in the group.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uplift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.
- **group** (*string*, [`'treatment'`, `'control'`]) – Group type for computing response rate: treatment or control.
  - **'treatment'**: Values equal 1 in the treatment column.
  - **'control'**: Values equal 0 in the treatment column.
- **strategy** (*string*, [`'overall'`, `'by_group'`]) – Determines the calculating strategy.
  - **'overall'**: The first step is taking the first k observations of all test data ordered by uplift prediction (overall both groups - control and treatment) and conversions in treatment and control groups calculated only on them. Then the difference between these conversions is calculated.
  - **'by\_group'**: Separately calculates conversions in top k observations in each group (control and treatment) sorted by uplift predictions. Then the difference between these conversions is calculated.
- **bins** (*int*) – Determines number of bins (and relative percentile) in the test data. Default is 10.

**Returns** Response rate at each percentile for control or treatment group array: Variance of the response rate at each percentile

**Return type** array

`sklift.metrics.metrics.treatment_balance_curve(uptift, treatment, winsize)`

Compute the treatment balance curve: proportion of treatment group in the ordered predictions.

#### Parameters

- **uptift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.
- **winsize** (*int*) – Size of the sliding window for calculating the balance between treatment and control.

**Returns** Points on a curve.

**Return type** array (shape = [ $>2$ ]), array (shape = [ $>2$ ])

`sklift.metrics.metrics.uptift_at_k(y_true, uptift, treatment, strategy, k=0.3)`

Compute uplift at first k percentage of the total sample.

#### Parameters

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uptift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.
- **k** (*float or int*) – If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the computation of uplift. If int, represents the absolute number of samples.
- **strategy** (*string, ['overall', 'by\_group']*) – Determines the calculating strategy.
  - **'overall'**: The first step is taking the first k observations of all test data ordered by uplift prediction (overall both groups - control and treatment) and conversions in treatment and control groups calculated only on them. Then the difference between these conversions is calculated.
  - **'by\_group'**: Separately calculates conversions in top k observations in each group (control and treatment) sorted by uplift predictions. Then the difference between these conversions is calculated.

Changed in version 0.1.0: Add supporting absolute values for k parameter Add parameter strategy

- **Returns** Uplift score at first k observations of the total sample.

**Return type** float

`sklift.metrics.metrics.uptift_auc_score(y_true, uptift, treatment)`

Compute Area Under the Uplift Curve from prediction scores.

#### Parameters

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uptift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.

**Returns** Area Under the Uplift Curve.

**Return type** float

`sklift.metrics.metrics.uptlift_curve(y_true, uplift, treatment)`  
 Compute Uplift curve

This is a general function, given points on a curve. For computing the area under the Uplift Curve, see `uptlift_auc_score()`.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uptlift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.

**Returns** Points on a curve.

**Return type** array (shape = [ $>2$ ]), array (shape = [ $>2$ ])

**See also:**

`uptlift_auc_score()`: Compute the area under the Uplift curve.

`plot_uptlift_qini_curves()`: Plot Uplift and Qini curves.

### 2.3.3 Visualization (sklift.viz)

`sklift.viz.base.plot_treatment_balance_curve(uptlift, treatment, random=True, win-size=0.1)`

Plot Treatment Balance curve.

**Parameters**

- **uptlift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.
- **random** (*bool, default True*) – Draw a random curve.
- **winsize** (*float, default 0.1*) – Size of the sliding window to apply. Should be between 0 and 1, extremes excluded.

**Returns** Object that stores computed values.

`sklift.viz.base.plot_uptlift_by_percentile(y_true, uplift, treatment, strategy, kind='line', bins=10)`

Plot uplift score, treatment response rate and control response rate at each percentile.

Treatment response rate is a target mean in the treatment group. Control response rate is a target mean in the control group. Uplift score is a difference between treatment response rate and control response rate.

**Parameters**

- **y\_true** (*1d array-like*) – Correct (true) target values.
- **uptlift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.
- **strategy** (*string, ['overall', 'by\_group']*) – Determines the calculating strategy.
  - **'overall'**: The first step is taking the first k observations of all test data ordered by uplift prediction (overall both groups - control and treatment) and conversions in

treatment and control groups calculated only on them. Then the difference between these conversions is calculated.

- **'by\_group'**: Separately calculates conversions in top k observations in each group (control and treatment) sorted by uplift predictions. Then the difference between these conversions is calculated.
- **kind**(*string*, [*'line'*, *'bar'*]) – The type of plot to draw. Default is *'line'*.
  - **'line'**: Generates a line plot.
  - **'bar'**: Generates a traditional bar-style plot.
- **bins**(*int*) – Determines number of bins (and relative percentile) in the test data. Default is 10.

**Returns** Object that stores computed values.

`sklift.viz.base.plot_uplift_preds(trmnt_preds, ctrl_preds, log=False, bins=100)`

Plot histograms of treatment, control and uplift predictions.

#### Parameters

- **trmnt\_preds** (*1d array-like*) – Predictions for all observations if they are treatment.
- **ctrl\_preds** (*1d array-like*) – Predictions for all observations if they are control.
- **log** (*bool*, *default False*) – Logarithm of source samples.
- **bins** (*integer or sequence*, *default 100*) – Number of histogram bins to be used. If an integer is given, bins + 1 bin edges are calculated and returned. If bins is a sequence, gives bin edges, including left edge of first bin and right edge of last bin. In this case, bins is returned unmodified.

**Returns** Object that stores computed values.

`sklift.viz.base.plot_uplift_qini_curves(y_true, uplift, treatment, random=True, perfect=False)`

Plot Uplift and Qini curves.

#### Parameters

- **y\_true** (*1d array-like*) – Ground truth (correct) labels.
- **uplift** (*1d array-like*) – Predicted uplift, as returned by a model.
- **treatment** (*1d array-like*) – Treatment labels.
- **random** (*bool*, *default True*) – Draw a random curve.
- **perfect** (*bool*, *default False*) – Draw a perfect curve.

**Returns** Object that stores computed values.

## 2.4 Tutorials

### 2.4.1 Basic

It is better to start scikit-uplift from the basic tutorials.

- **The overview of the basic approaches to solving the Uplift Modeling problem**
  - In English: [nbviewer](#) | [github](#)

- In Russian: [nbviewer](#) | [github](#)
- **Example of usage model from `sklift.models` in `sklearn.pipeline`**
  - In English: [nbviewer](#) | [github](#)
  - In Russian: [nbviewer](#) | [github](#)

## 2.5 Release History

### 2.5.1 Legend for changelogs

- something big that you couldn't do before.
- something that you couldn't do before.
- a miscellaneous minor improvement.
- something that previously didn't work as documented – or according to reasonable expectations – should now work.
- you will need to change your code to have the same effect in the future; or a feature will be removed in the future.

### 2.5.2 Version 0.1.2

#### **sklift.models**

- Fix bugs in `TwoModels` for regression problem.
- Minor code refactoring.

#### **sklift.metrics**

- Minor code refactoring.

#### **sklift.viz**

- Add bar plot in `plot_uplift_by_percentile` by @ElisovaIra.
- Fix bug in `plot_uplift_by_percentile`.
- Minor code refactoring.

### 2.5.3 Version 0.1.1

#### **sklift.viz**

- Add `plot_uplift_by_percentile` by @ElisovaIra.
- Fix bug with import `plot_treatment_balance_curve`.



### sklift.metrics

- Add `response_rate_by_percentile` by @ElisovaIra.
- Fix bug with import `uplift_auc_score` and `qini_auc_score`.
- Fix typos in docstrings.

### Miscellaneous

- Add tutorial “Example of usage model from `sklift.models` in `sklearn.pipeline`”.
- Add link to Release History in main `Readme.md`.

## 2.5.4 Version 0.1.0

### sklift.models

- Fix typo in `TwoModels` docstring by @spiaz.
- Improve docstrings and add references to all approaches.

### sklift.metrics

- Add `treatment_balance_curve` by @spiaz.
- The metrics `auuc` and `auc` are now respectively renamed to `uplift_auc_score` and `qini_auc_score`. So, `auuc` and `auc` will be removed in 0.2.0.
- Add a new parameter `startegy` in `uplift_at_k`.

### sklift.viz

- Add `plot_treatment_balance_curve` by @spiaz.
- fix typo in `plot_uplift_qini_curves` by @spiaz.

### Miscellaneous

- Remove `sklift.preprocess` submodule.
- Add compatibility of tutorials with colab and add colab buttons by @ElMaxuno.
- Add Changelog.
- Change the documentation structure. Add next pages: [Tutorials](#), [Release History](#) and [Hall of fame](#).

## 2.6 Hall of Fame

Here are the links to the competitions, names of the winners and to their solutions, where scikit-uplift was used.

## 2.6.1 X5 RetailHero Uplift Modeling contest

2. [Kirill Liksakov](#) solution

---

---

### Papers and materials

---

1. **Gutierrez, P., & Gérardy, J. Y.** Causal Inference and Uplift Modelling: A Review of the Literature. In International Conference on Predictive Applications and APIs (pp. 1-13).
  2. **Artem Betlei, Criteo Research; Eustache Diemert, Criteo Research; Massih-Reza Amini, Univ. Grenoble Alpes** Dependent and Shared Data Representations improve Uplift Prediction in Imbalanced Treatment Conditions FAIM'18 Workshop on CausalML.
  3. **Eustache Diemert, Artem Betlei, Christophe Renaudin, and Massih-Reza Amini. 2018.** A Large Scale Benchmark for Uplift Modeling. In Proceedings of AdKDD & TargetAd (ADKDD'18). ACM, New York, NY, USA, 6 pages.
  4. **Athey, Susan, and Imbens, Guido. 2015.** Machine learning methods for estimating heterogeneous causal effects. Preprint, arXiv:1504.01132. Google Scholar.
  5. **Oscar Mesalles Naranjo. 2012.** Testing a New Metric for Uplift Models. Dissertation Presented for the Degree of MSc in Statistics and Operational Research.
  6. **Kane, K., V. S. Y. Lo, and J. Zheng. 2014.** Mining for the Truly Responsive Customers and Prospects Using True-Lift Modeling: Comparison of New and Existing Methods. *Journal of Marketing Analytics* 2 (4): 218–238.
  7. **Maciej Jaskowski and Szymon Jaroszewicz.** Uplift modeling for clinical trial data. ICML Workshop on Clinical Data Analysis, 2012.
  8. **Lo, Victor. 2002.** The True Lift Model - A Novel Data Mining Approach to Response Modeling in Database Marketing. *SIGKDD Explorations*. 4. 78-86.
  9. **Zhao, Yan & Fang, Xiao & Simchi-Levi, David. 2017.** Uplift Modeling with Multiple Treatments and General Response Types. 10.1137/1.9781611974973.66.
-



## CHAPTER 4

---

Tags

---

**EN:** uplift modeling, uplift modelling, causal inference, causal effect, causality, individual treatment effect, true lift, net lift, incremental modeling

**RU:** , Uplift

**ZH:** ,,,,,,

---



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### S

`sklift.metrics.metrics`, [15](#)

`sklift.viz.base`, [18](#)



## A

`auc()` (in module *sklift.metrics.metrics*), 15  
`auuc()` (in module *sklift.metrics.metrics*), 15

## C

`ClassTransformation` (class in *sklift.models.models*), 10  
`ctrl_preds_` (*sklift.models.models.SoloModel* attribute), 8  
`ctrl_preds_` (*sklift.models.models.TwoModels* attribute), 14

## F

`fit()` (*sklift.models.models.ClassTransformation* method), 11  
`fit()` (*sklift.models.models.SoloModel* method), 9  
`fit()` (*sklift.models.models.TwoModels* method), 14

## P

`plot_treatment_balance_curve()` (in module *sklift.viz.base*), 18  
`plot_uplift_by_percentile()` (in module *sklift.viz.base*), 18  
`plot_uplift_preds()` (in module *sklift.viz.base*), 19  
`plot_uplift_qini_curves()` (in module *sklift.viz.base*), 19  
`predict()` (*sklift.models.models.ClassTransformation* method), 11  
`predict()` (*sklift.models.models.SoloModel* method), 9  
`predict()` (*sklift.models.models.TwoModels* method), 15

## Q

`qini_auc_score()` (in module *sklift.metrics.metrics*), 15  
`qini_curve()` (in module *sklift.metrics.metrics*), 16

## R

`response_rate_by_percentile()` (in module *sklift.metrics.metrics*), 16

## S

*sklift.metrics.metrics* (module), 15  
*sklift.viz.base* (module), 18  
*SoloModel* (class in *sklift.models.models*), 8

## T

`treatment_balance_curve()` (in module *sklift.metrics.metrics*), 17  
`trmnt_preds_` (*sklift.models.models.SoloModel* attribute), 8  
`trmnt_preds_` (*sklift.models.models.TwoModels* attribute), 13  
*TwoModels* (class in *sklift.models.models*), 13

## U

`uplift_at_k()` (in module *sklift.metrics.metrics*), 17  
`uplift_auc_score()` (in module *sklift.metrics.metrics*), 17  
`uplift_curve()` (in module *sklift.metrics.metrics*), 18