# causallift Documentation

***Release 1.0.7***

**Yusuke Minami**

**May 13, 2023**

# CONTENTS

CHAPTER

ONE

# CAUSALLIFT: PYTHON PACKAGE FOR UPLIFT MODELING IN REAL-WORLD BUSINESS; APPLICABLE FOR BOTH A/B TESTING AND OBSERVATIONAL DATA

CHAPTER

TWO

# INTRODUCTION

If you are simply building a Machine Learning model and executing promotion campaigns to the customers who are predicted to buy a product, for example, it is not efficient.

Some customers will buy a product anyway even without promotion campaigns (called “Sure things”).

It is even possible that the campaign triggers some customers to churn (called “Do Not Disturbs” or “Sleeping Dogs”).

The solution is Uplift Modeling.

CHAPTER

THREE

# WHAT IS UPLIFT MODELING?

Uplift Modeling is a Machine Learning technique to find which customers (individuals) should be targeted (”treated”) and which customers should not be targeted.

Uplift Modeling is also known as persuasion modeling, incremental modeling, treatment effects modeling, true lift modeling, or net modeling.

Applications of Uplift Modeling for business include:

- Increase revenue by finding which customers should be targeted for advertising/marketing campaigns and which customers should not.
- Retain revenue by finding which customers should be contacted to prevent churn and which customers should not.

CHAPTER

FOUR

# A FAMOUS USE CASE?

The most famous use case of Uplift Modeling would be the 44th US president Barack Obama's 2nd presidential campaign in 2012. Obama's team used Uplift Modeling to find which voters could be persuaded to vote for him. Here are some articles.

- <What is 'Persuasion Modeling', and how did it help Obama to win the elections?>
- <How Obama's Team Used Big Data to Rally Voters>
- <How uplift modeling helped Obama's campaign – and can aid marketers>

CHAPTER

FIVE

# HOW DOES UPLIFT MODELING WORK?

Uplift Modeling estimates uplift scores (a.k.a. CATE: Conditional Average Treatment Effect or ITE: Individual Treatment Effect). Uplift score is how much the estimated conversion rate will increase by the campaign.

Suppose you are in charge of a marketing campaign to sell a product, and the estimated conversion rate (probability to buy a product) of a customer is 50 % if targeted and the estimated conversion rate is 40 % if not targeted, then the uplift score of the customer is (50-40) = +10 % points. Likewise, suppose the estimated conversion rate if targeted is 20 % and the estimated conversion rate if not targeted is 80%, the uplift score is (20-80) = -60 % points (negative value).

The range of uplift scores is between -100 and +100 % points (-1 and +1). It is recommended to target customers with high uplift scores and avoid customers with negative uplift scores to optimize the marketing campaign.

CHAPTER

SIX

# WHAT ARE THE ADVANTAGES OF “CAUSALLIFT” PACKAGE?

- CausalLift works with both A/B testing results and observational datasets.
- CausalLift can output intuitive metrics for evaluation.

CHAPTER

SEVEN

# WHY CAUSALLIFT WAS DEVELOPED?

In a word, to use for real-world business.

- Existing packages for Uplift Modeling assumes the dataset is from A/B Testing (a.k.a. Randomized Controlled Trial). In real-world business, however, observational datasets in which treatment (campaign) targets were not chosen randomly are more common especially in the early stage of evidence-based decision making. CausalLift supports observational datasets using a basic methodology in Causal Inference called "Inverse Probability Weighting" based on the assumption that propensity to be treated can be inferred from the available features.
- There are 2 challenges of Uplift Modeling; explainability of the model and evaluation. CausalLift utilizes a basic methodology of Uplift Modeling called Two Models approach (training 2 models independently for treated and untreated samples to compute the CATE (Conditional Average Treatment Effects) or uplift scores) to address these challenges.
  - [Explainability of the model] Since it is relatively simple, it is less challenging to explain how it works to stakeholders in the business.
  - [Explainability of evaluation] To evaluate Uplift Modeling, metrics such as Qini and AUUC (Area Under the Uplift Curve) are used in research, but these metrics are difficult to explain to the stakeholders. For business, a metric that can estimate how much more profit can be earned is more practical. Since CausalLift adopted the Two-Model approach, the 2 models can be reused to simulate the outcome of following the recommendation by the Uplift Model and can estimate how much conversion rate (the proportion of people who took the desired action such as buying a product) will increase using the uplift model.

CHAPTER

EIGHT

# INSTALLATION

- [Option 1] To install the latest release from the PyPI:

```
$ pip install causallift
```

- [Option 2] To install the latest pre-release:

```
$ pip install git+https://github.com/Minyus/causallift.git
```

- [Option 3] To install the latest pre-release without need to reinstall even after modifying the source code:

```
$ git clone https://github.com/Minyus/causallift.git
$ cd pipelinex
$ python setup.py develop
```

## 8.1 Dependencies:

- numpy
- pandas
- scikit-learn<0.22 (sklearn==0.22 may not work.)
- easydict
- kedro>=0.15.0

## 8.2 Optional:

- matplotlib
- xgboost
- scikit-optimize

## 8.3 Optional for visualization of the pipeline:

- kedro-viz

CHAPTER

NINE

# HOW IS THE DATA PIPELINE IMPLEMENTED BY CAUSALLIFT?

## 9.1 Step 0: Prepare data

Prepare the following columns in 2 pandas DataFrames, train and test (validation).

- Features
  - a.k.a independent variables, explanatory variables, covariates
  - e.g. customer gender, age range, etc.
  - Note: Categorical variables need to be one-hot coded so propensity can be estimated using logistic regression. pandas.get_dummies can be used.
- Outcome: binary (0 or 1)
  - a.k.a dependent variable, target variable, label
  - e.g. whether the customer bought a product, clicked a link, etc.
- Treatment: binary (0 or 1)
  - a variable you can control and want to optimize for each individual (customer)
  - a.k.a intervention
  - e.g. whether an advertising campaign was executed, whether a discount was offered, etc.
  - Note: if you cannot find a treatment column, you may need to ask stakeholders to get the data, which might take hours to years.
- [Optional] Propensity: continuous between 0 and 1
  - propensity (or probability) to be treated for observational datasets (not needed for A/B Testing results)
  - If not provided, CausalLift can estimate from the features using logistic regression.

## 9.2 Step 1: Prepare for Uplift modeling and optionally estimate propensity scores using a supervised classification model

If the `train_df` is from observational data (not A/B Test), you can set `enable_ipw`=True so IPW (Inverse Probability Weighting) can address the issue that treatment should have been chosen based on a different probability (propensity score) for each individual (e.g. customer, patient, etc.)

If the `train_df` is from A/B Test or RCT (Randomized Controlled Trial), set `enble_ipw`=False to skip estimating propensity score.

## 9.3 Step 2: Estimate CATE by 2 supervised classification models

Train 2 supervised classification models (e.g. XGBoost) for treated and untreated samples independently and compute estimated CATE (Conditional Average Treatment Effect), ITE (Individual Treatment Effect), or uplift score.

This step is the Uplift Modeling consisting of 2 sub-steps:

1. Training using train_df (Note: `Treatment` and `Outcome` are used)
2. Prediction of CATE for train_df and test_df (Note: Neither `Treatment` nor `Outcome` is used.)

## 9.4 Step 3 [Optional] Estimate impact by following recommendation based on CATE

Estimate how much conversion rate will increase by selecting treatment (campaign) targets as recommended by the uplift modeling.

You can optionally evaluate the predicted CATE for train_df and test_df (Note: `CATE`, `Treatment` and `Outcome` are used.)

This step is *optional*; you can skip if you want only CATE and you do not find this evaluation step useful.

CHAPTER

# TEN

# HOW TO USE CAUSALLIFT?

There are 2 ways:

- [Deprecated option] Use `causallift.CausalLift` class interface
- [Recommended option] Use `causallift.nodes` subpackage with `PipelineX` package

## 10.1 [Deprecated option] Use `causallift.CausalLift` class interface

Please see the demo code in Google Colab (free cloud CPU/GPU environment):

To run the code, navigate to "Runtime" >> "Run all".

To download the notebook file, navigate to "File" >> "Download .ipynb".

Here are the basic steps to use.

```
from causallift import CausalLift

""" Step 1. """
cl = CausalLift(train_df, test_df, enable_ipw=True)

""" Step 2. """
train_df, test_df = cl.estimate_cate_by_2_models()

""" Step 3. """
estimated_effect_df = cl.estimate_recommendation_impact()
```

## 10.2 [Recommended option] Use `causallift.nodes` subpackage with `PipelineX` package

Please see PipelineX package and use PipelineX Causallift example project.

CHAPTER

# ELEVEN

# HOW TO RUN INFERENCE (PREDICTION OF CATE FOR NEW DATA WITH `TREATMENT` AND `OUTCOME` UNKNOWN)?

Use the whole historical data (A/B Test data or observational data) as train_df instead of splitting into `tran_df` and `test_df`, and use the new data with `Treatment` and `Outcome` unknown as `test_df`.

This is possible because `Treatment` and `Outcome` are not used for prediction of CATE after Uplift Model is trained using `Treatment` and `Outcome`.

Please note that valid evaluation for `test_df` will not be available as valid `Treatment` and `Outcome` are not available.

CHAPTER

# TWELVE

# DETAILS ABOUT THE PARAMETERS

Please see [CausalLift API document].

CHAPTER

THIRTEEN

# SUPPORTED PYTHON VERSIONS

- Python 3.5
- Python 3.6 (Tested and recommended)
- Python 3.7

CHAPTER

FOURTEEN

# RELATED PYTHON PACKAGES

- "pylift" [documentation]

  Uplift Modeling based on Transformed Outcome method for A/B Testing data and visualization of metrics such as Qini.

- "EconML" (ALICE: Automated Learning and Intelligence for Causation and Economics) [documentation]

  Several advanced methods to estimate CATE from observational data.

- "DoWhy" [documentation]

  Visualization of steps in Causal Inference for observational data.

- "pymatch"

  Propensity Score Matching for observational data.

- "Ax" [documentation]

  Platform for adaptive experiments, powered by BoTorch, a library built on PyTorch

CHAPTER

FIFTEEN

# RELATED R PACKAGES

- "uplift"

  Uplift Modeling.

- "tools4uplift" [paper]

  Uplift Modeling and utility tools for quantization of continuous variables, visualization of metrics such as Qini, and automatic feature selection.

- "matching"

  Propensity Score Matching for observational data.

- "CausalImpact" [documentation]

  Causal inference using Bayesian structural time-series models

CHAPTER

SIXTEEN

# REFERENCES


- Gutierrez, Pierre. and G´erardy, Jean-Yves. Causal inference and uplift modelling: A review of the literature. In International Conference on Predictive Applications and APIs, pages 1-13, 2017.
- Athey, Susan and Imbens, Guido W. Machine learning methods for estimating heterogeneous causal effects. Stat, 2015.
- Yi, Robert. and Frost, Will. (n.d.). Pylift: A Fast Python Package for Uplift Modeling. Retrieved April 3, 2019, from https://tech.wayfair.com/2018/10/pylift-a-fast-python-package-for-uplift-modeling/

CHAPTER

SEVENTEEN

# INTRODUCTORY RESOURCES ABOUT UPLIFT MODELING

- <Medium article: Uplift Models for better marketing campaigns (Part 1)>
- <Medium article: Simple Machine Learning Techniques To Improve Your Marketing Strategy: Demystifying Uplift Models>
- <Wikipedia: Uplift_modelling>

CHAPTER

EIGHTEEN

# LICENSE

BSD 2-clause License.

CHAPTER

# NINETEEN

# TO-DOS

- Improve documentation
- Clarify the model summary output including visualization
- Add examples of applying uplift modeling to more publicly available datasets (such as Lending Club Loan Data as pymatch did.
- Support for multiple treatments

CHAPTER

TWENTY

# CONTRIBUTING

Any feedback is welcome!

Please create an issue for questions, suggestions, and feature requests. Please open pull requests to improve documentation, usability, and features against `develop` branch.

Separate pull requests for each improvement are appreciated rather than a big pull request. It is encouraged to use:

- Google-style docstrings
- PEP 484 comment-style type annotation although Python 2 is not supported.
- An intelligent IDE such as PyCharm or VS Code

If you could write a review about CausalLift in any natural languages (English, Chinese, Japanese, etc.) or implement similar features in any programming languages (R, SAS, etc.), please let me know. I will add the link here.

CHAPTER

# TWENTYONE

## KEYWORDS TO SEARCH

[English] Causal Inference, Counterfactual, Propensity Score, Econometrics

[] , , ,

[] , , ,

CHAPTER

# TWENTYTWO

# ARTICLE ABOUT CAUSALLIST IN JAPANESE

- https://qiita.com/Minyus86/items/07ce57a8bddc49c2bbf5

CHAPTER

# TWENTYTHREE

# AUTHOR:

Yusuke Minami

- @Minyus
- https://www.linkedin.com/in/yusukeminami/
- https://twitter.com/Minyus86

CHAPTER

# TWENTYFOUR

## CONTRIBUTORS:

@farismosman

CHAPTER

# TWENTYFIVE

# CAUSALLIFT

## 25.1 causallift package

CausalLift

### 25.1.1 Subpackages

#### 25.1.1.1 causallift.nodes package

##### Submodules

##### causallift.nodes.estimate_propensity module

causallift.nodes.estimate_propensity.**estimate_propensity**(*args*, *df*, *model*)

causallift.nodes.estimate_propensity.**fit_propensity**(*args*, *df*)

causallift.nodes.estimate_propensity.**schedule_propensity_scoring**(*args*, *df*)

##### causallift.nodes.model_for_each module

**class** causallift.nodes.model_for_each.**ModelForTreated**(**posargs*, ***kwargs*)

Bases: *ModelForTreatedOrUntreated*

**__init__**(**posargs*, ***kwargs*)

**class** causallift.nodes.model_for_each.**ModelForTreatedOrUntreated**(*treatment_val=1.0*)

Bases: object

**__init__**(*treatment_val=1.0*)

**fit**(*args*, *df_*)

**predict_proba**(*args*, *df_*, *models_dict*)

**simulate_recommendation**(*args*, *df_*, *models_dict*)

**class** causallift.nodes.model_for_each.**ModelForUntreated**(**posargs*, ***kwargs*)

Bases: *ModelForTreatedOrUntreated*

__init__(*posargs*, ***kwargs*)

causallift.nodes.model_for_each.**bundle_treated_and_untreated_models**(*treated_model*, *untreated_model*)

causallift.nodes.model_for_each.**model_for_treated_fit**(**posargs*, ***kwargs*)

causallift.nodes.model_for_each.**model_for_treated_predict_proba**(**posargs*, ***kwargs*)

causallift.nodes.model_for_each.**model_for_treated_simulate_recommendation**(**posargs*, ***kwargs*)

causallift.nodes.model_for_each.**model_for_untreated_fit**(**posargs*, ***kwargs*)

causallift.nodes.model_for_each.**model_for_untreated_predict_proba**(**posargs*, ***kwargs*)

causallift.nodes.model_for_each.**model_for_untreated_simulate_recommendation**(**posargs*, ***kwargs*)

### causallift.nodes.utils module

causallift.nodes.utils.**add_cate_to_df**(*args*, *df*, *cate_estimated*, *proba_treated*, *proba_untreated*)

causallift.nodes.utils.**apply_method**(*obj*, *method*, ***kwargs*)

causallift.nodes.utils.**bundle_train_and_test_data**(*args*, *train_df*, *test_df*)

causallift.nodes.utils.**compute_cate**(*proba_treated*, *proba_untreated*)

causallift.nodes.utils.**concat_train_test**(*args*, *train*, *test*)
Concatenate train and test series. Use series.xs('train') or series.xs('test') to split

causallift.nodes.utils.**concat_train_test_df**(*args*, *train*, *test*)
Concatenate train and test data frames. Use df.xs('train') or df.xs('test') to split.

causallift.nodes.utils.**conf_mat_df**(*y_true*, *y_pred*)

causallift.nodes.utils.**estimate_effect**(*args*, *sim_treated_df*, *sim_untreated_df*)

causallift.nodes.utils.**gain_tuple**(*df_*, *r_*)

causallift.nodes.utils.**get_cols_features**(*df*, *non_feature_cols=['Treatment', 'Outcome', 'TransformedOutcome', 'Propensity', 'Recommendation']*)

causallift.nodes.utils.**impute_cols_features**(*args*, *df*)

causallift.nodes.utils.**initialize_model**(*args*, *model_key='uplift_model_params'*, *default_estimator='sklearn.linear_model.LogisticRegression'*)

**Return type**
Type[BaseEstimator]

causallift.nodes.utils.**len_o**(*df*, *outcome=1.0*, *col_outcome='Outcome'*)

causallift.nodes.utils.**len_t**(*df*, *treatment=1.0*, *col_treatment='Treatment'*)

causallift.nodes.utils.**len_to**(*df*, *treatment=1.0*, *outcome=1.0*, *col_treatment='Treatment'*, *col_outcome='Outcome'*)

causallift.nodes.utils.**outcome_fraction_**(*df*, *col_outcome='Outcome'*)

causallift.nodes.utils.**overall_uplift_gain_**(*df*, *treatment=1.0*, *outcome=1.0*, *col_treatment='Treatment'*, *col_outcome='Outcome'*)

causallift.nodes.utils.**recommend_by_cate**(*args*, *df*, *treatment_fractions*)

causallift.nodes.utils.**score_df**(*y_train*, *y_test*, *y_pred_train*, *y_pred_test*, *average='binary'*)

causallift.nodes.utils.**treatment_fraction_**(*df*, *col_treatment='Treatment'*)

causallift.nodes.utils.**treatment_fractions_**(*args*, *df*)

**Return type**
Type[EasyDict]

#### 25.1.1.2 causallift.context package

### Submodules

### causallift.context.base_context module

This module provides context for Kedro project.

**class** causallift.context.base_context.**BaseKedroContext**

Bases: ABC

KedroContext is the base class which holds the configuration and Kedro's main functionality. Project-specific context class should extend this abstract class and implement the all abstract methods.

Example:

```
from kedro.context import KedroContext
from kedro.pipeline import Pipeline

class ProjectContext(KedroContext):
    @property
    def pipeline(self) -> Pipeline:
        return Pipeline([])
```

**__init__**()

**property catalog**

Read-only property referring to Kedro's DataCatalog for this context.

**Return type**
DataCatalog

**Returns**
DataCatalog defined in *catalog.yml*.

**property io**

Read-only alias property referring to Kedro's DataCatalog for this context.

**Return type**
DataCatalog

**Returns**
DataCatalog defined in *catalog.yml*.

**abstract property pipeline**

Abstract property for Pipeline getter.

**Return type**
`Pipeline`

**Returns**
Defined pipeline.

**run**(*tags=None*, *runner=None*, *node_names=None*, *from_nodes=None*, *to_nodes=None*)

Runs the pipeline with a specified runner.

**Parameters**

- **tags** (`Optional[Iterable[str]]`) – An optional list of node tags which should be used to filter the nodes of the `Pipeline`. If specified, only the nodes containing *any* of these tags will be run.
- **runner** (`Optional[AbstractRunner]`) – An optional parameter specifying the runner that you want to run the pipeline with.
- **node_names** (`Optional[Iterable[str]]`) – An optional list of node names which should be used to filter the nodes of the `Pipeline`. If specified, only the nodes with these names will be run.
- **from_nodes** (`Optional[Iterable[str]]`) – An optional list of node names which should be used as a starting point of the new `Pipeline`.
- **to_nodes** (`Optional[Iterable[str]]`) – An optional list of node names which should be used as an end point of the new `Pipeline`.

**Raises**
*KedroContextError* – If the resulting `Pipeline` is empty or incorrect tags are provided.

**Return type**
`Dict[str, Any]`

**Returns**
Any node outputs that cannot be processed by the `DataCatalog`. These are returned in a dictionary, where the keys are defined by the node outputs.

**exception** causallift.context.base_context.**KedroContextError**

Bases: `Exception`

Error occurred when loading project and running context pipeline.

### causallift.context.flexible_context module

Application entry point.

**class** causallift.context.flexible_context.**FlexibleKedroContext**(*runner=None*, *only_missing=False*, ***kwargs*)

Bases: *ProjectContext2*

Overwrite the default runner and only_missing option for the run.

**__init__**(*runner=None*, *only_missing=False*, ***kwargs*)

**run**(*tags=None*, *runner=None*, *node_names=None*, *only_missing=False*)

Runs the pipeline wi th a specified runner.

**Parameters**

- **tags** (Optional[Iterable[str]]) – An optional list of node tags which should be used to filter the nodes of the Pipeline. If specified, only the nodes containing *any* of these tags will be run.
- **runner** (Optional[AbstractRunner]) – An optional parameter specifying the runner that you want to run the pipeline with.
- **node_names** (Optional[Iterable[str]]) – An optional list of node names which should be used to filter the nodes of the Pipeline. If specified, only the nodes with these names will be run.
- **only_missing** (bool) – An option to run only missing nodes.

**Raises**

*KedroContextError* – If the resulting Pipeline is empty or incorrect tags are provided.

**Return type**

Dict[str, Any]

**Returns**

Any node outputs that cannot be processed by the DataCatalog. These are returned in a dictionary, where the keys are defined by the node outputs.

**class** causallift.context.flexible_context.**ProjectContext**

Bases: *BaseKedroContext*

Users can override the remaining methods from the parent class here, or create new ones (e.g. as required by plugins)

**property pipeline**

Abstract property for Pipeline getter.

**Return type**

Pipeline

**Returns**

Defined pipeline.

**project_name = 'CausalLift'**

**project_version = '0.15.0'**

**run**(*tags=None*, *runner=None*, *node_names=None*, *only_missing=False*)

Runs the pipeline wi th a specified runner.

**Parameters**

- **tags** (Optional[Iterable[str]]) – An optional list of node tags which should be used to filter the nodes of the Pipeline. If specified, only the nodes containing *any* of these tags will be run.
- **runner** (Optional[AbstractRunner]) – An optional parameter specifying the runner that you want to run the pipeline with.
- **node_names** (Optional[Iterable[str]]) – An optional list of node names which should be used to filter the nodes of the Pipeline. If specified, only the nodes with these names will be run.

- **only_missing** (bool) – An option to run only missing nodes.

**Raises**
*KedroContextError* – If the resulting Pipeline is empty or incorrect tags are provided.

**Return type**
Dict[str, Any]

**Returns**
Any node outputs that cannot be processed by the DataCatalog. These are returned in a dictionary, where the keys are defined by the node outputs.

class causallift.context.flexible_context.**ProjectContext1**

Bases: *ProjectContext*

Allow to specify runner by string.

**run**(*runner=None*, ***kwargs*)

Runs the pipeline wi th a specified runner.

**Parameters**

- **tags** – An optional list of node tags which should be used to filter the nodes of the Pipeline. If specified, only the nodes containing *any* of these tags will be run.
- **runner** (Union[AbstractRunner, str, None]) – An optional parameter specifying the runner that you want to run the pipeline with.
- **node_names** – An optional list of node names which should be used to filter the nodes of the Pipeline. If specified, only the nodes with these names will be run.
- **only_missing** – An option to run only missing nodes.

**Raises**
*KedroContextError* – If the resulting Pipeline is empty or incorrect tags are provided.

**Return type**
Dict[str, Any]

**Returns**
Any node outputs that cannot be processed by the DataCatalog. These are returned in a dictionary, where the keys are defined by the node outputs.

class causallift.context.flexible_context.**ProjectContext2**

Bases: *ProjectContext1*

Keep the output datasets in the catalog.

**run**(***kwargs*)

Runs the pipeline wi th a specified runner.

**Parameters**

- **tags** – An optional list of node tags which should be used to filter the nodes of the Pipeline. If specified, only the nodes containing *any* of these tags will be run.
- **runner** – An optional parameter specifying the runner that you want to run the pipeline with.
- **node_names** – An optional list of node names which should be used to filter the nodes of the Pipeline. If specified, only the nodes with these names will be run.
- **only_missing** – An option to run only missing nodes.

**Raises**
*KedroContextError* – If the resulting `Pipeline` is empty or incorrect tags are provided.

**Return type**
`Dict[str, Any]`

**Returns**
Any node outputs that cannot be processed by the `DataCatalog`. These are returned in a dictionary, where the keys are defined by the node outputs.

### 25.1.2 Submodules

### 25.1.3 causallift.causal_lift module

class causallift.causal_lift.**CausalLift**(*train_df=None, test_df=None, cols_features=None, col_treatment='Treatment', col_outcome='Outcome', col_propensity='Propensity', col_proba_if_treated='Proba_if_Treated', col_proba_if_untreated='Proba_if_Untreated', col_cate='CATE', col_recommendation='Recommendation', col_weight='Weight', min_propensity=0.01, max_propensity=0.99, verbose=2, uplift_model_params={'cv': 3, 'estimator': 'xgboost.XGBClassifier', 'n_jobs': -1, 'param_grid': {'base_score': [0.5], 'booster': ['gbtree'], 'colsample_bylevel': [1], 'colsample_bytree': [1], 'gamma': [0], 'learning_rate': [0.1], 'max_delta_step': [0], 'max_depth': [3], 'min_child_weight': [1], 'missing': [None], 'n_estimators': [100], 'n_jobs': [-1], 'nthread': [None], 'objective': ['binary:logistic'], 'random_state': [0], 'reg_alpha': [0], 'reg_lambda': [1], 'scale_pos_weight': [1], 'subsample': [1], 'verbose': [0]}, 'return_train_score': False, 'scoring': None, 'search_cv': 'sklearn.model_selection.GridSearchCV'}, enable_ipw=True, enable_weighting=False, propensity_model_params={'cv': 3, 'estimator': 'sklearn.linear_model.LogisticRegression', 'n_jobs': -1, 'param_grid': {'C': [0.1, 1, 10], 'class_weight': [None], 'dual': [False], 'fit_intercept': [True], 'intercept_scaling': [1], 'max_iter': [100], 'multi_class': ['ovr'], 'n_jobs': [1], 'penalty': ['l1', 'l2'], 'random_state': [0], 'solver': ['liblinear'], 'tol': [0.0001], 'warm_start': [False]}, 'return_train_score': False, 'scoring': None, 'search_cv': 'sklearn.model_selection.GridSearchCV'}, index_name='index', partition_name='partition', runner='SequentialRunner', conditionally_skip=False, df_print=<function display>, dataset_catalog={'df_03': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'estimated_effect_df': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'propensity_model': <kedro.extras.datasets.pickle.pickle_dataset.PickleDataSet object>, 'treated__sim_eval_df': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'untreated__sim_eval_df': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'uplift_models_dict': <kedro.extras.datasets.pickle.pickle_dataset.PickleDataSet object>}, logging_config={'disable_existing_loggers': False, 'formatters': {'json_formatter': {'class': 'pythonjsonlogger.jsonlogger.JsonFormatter', 'format': '[%(asctime)s|%(name)s|%(funcName)s|%(levelname)s] %(message)s'}, 'simple': {'format': '[%(asctime)s|%(name)s|%(levelname)s] %(message)s'}}, 'handlers': {'console': {'class': 'logging.StreamHandler', 'formatter': 'simple', 'level': 'INFO', 'stream': 'ext://sys.stdout'}, 'error_file_handler': {'backupCount': 20, 'class': 'logging.handlers.RotatingFileHandler', 'delay': True, 'encoding': 'utf8', 'filename': './errors.log', 'formatter': 'simple', 'level': 'ERROR', 'maxBytes': 10485760}, 'info_file_handler': {'backupCount': 20, 'class': 'logging.handlers.RotatingFileHandler', 'delay': True, 'encoding': 'utf8', 'filename': './info.log', 'formatter': 'simple', 'level': 'INFO', 'maxBytes': 10485760}}, 'loggers': {'anyconfig': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'WARNING', 'propagate': False}, 'causallift': {'handlers':*

Bases: object

Set up datasets for uplift modeling. Optionally, propensity scores are estimated based on logistic regression.

**Parameters**

- **train_df** (Optional[DataFrame]) – Pandas Data Frame containing samples used for training
- **test_df** (Optional[DataFrame]) – Pandas Data Frame containing samples used for testing
- **cols_features** (Optional[List[str]]) – List of column names used as features. If None (default), all the columns except for outcome, propensity, CATE, and recommendation.
- **col_treatment** (str) – Name of treatment column. 'Treatment' in default.
- **col_outcome** (str) – Name of outcome column. 'Outcome' in default.
- **col_propensity** (str) – Name of propensity column. 'Propensity' in default.
- **col_cate** (str) – Name of CATE (Conditional Average Treatment Effect) column. 'CATE' in default.
- **col_recommendation** (str) – Name of recommendation column. 'Recommendation' in default.
- **col_weight** (str) – Name of weight column. 'Weight' in default.
- **min_propensity** (float) – Minimum propensity score. 0.01 in default.
- **max_propensity** (float) – Maximum propensity score. 0.99 in defualt.
- **verbose** (int) – How much info to show. Valid values are:
  - 0 to show nothing
  - 1 to show only warning
  - 2 (default) to show useful info
  - 3 to show more info
- **uplift_model_params** (Union[Dict[str, List[Any]], Type[BaseEstimator]]) – Parameters used to fit 2 XGBoost classifier models.
  - Optionally use *search_cv* key to specify the Search CV class name.

    e.g. *sklearn.model_selection.GridSearchCV*

    Refer to https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
  - Use *estimator* key to specify the estimator class name.

    e.g. *xgboost.XGBClassifier*

    Refer to https://xgboost.readthedocs.io/en/latest/parameter.html
  - Optionally use *const_params* key to specify the constant parameters to construct the estimator.

  If None (default):

```
dict(
    search_cv="sklearn.model_selection.GridSearchCV",
    estimator="xgboost.XGBClassifier",
    scoring=None,
    cv=3,
    return_train_score=False,
    n_jobs=-1,
    param_grid=dict(
        max_depth=[3],
        learning_rate=[0.1],
        n_estimators=[100],
        verbose=[0],
        objective=["binary:logistic"],
        booster=["gbtree"],
        n_jobs=[-1],
        nthread=[None],
        gamma=[0],
        min_child_weight=[1],
        max_delta_step=[0],
        subsample=[1],
        colsample_bytree=[1],
        colsample_bylevel=[1],
        reg_alpha=[0],
        reg_lambda=[1],
        scale_pos_weight=[1],
        base_score=[0.5],
        missing=[None],
    ),
)
```

Alternatively, estimator model object is acceptable. The object must have the following methods compatible with scikit-learn estimator interface.

- `fit()`
- `predict()`
- `predict_proba()`

- **enable_ipw** (`bool`) – Enable Inverse Probability Weighting based on the estimated propensity score. True in default.
- **enable_weighting** (`bool`) – Enable Weighting. False in default.
- **propensity_model_params** (`Dict[str, List[Any]]`) – Parameters used to fit logistic regression model to estimate propensity score.
  - Optionally use *search_cv* key to specify the Search CV class name.

    e.g. *sklearn.model_selection.GridSearchCV*

    Refer to https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
  - Use *estimator* key to specify the estimator class name.

    e.g. *sklearn.linear_model.LogisticRegression*

Refer to https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

– Optionally use *const_params* key to specify the constant parameters to construct the estimator.

If None (default):

```
dict(
    search_cv="sklearn.model_selection.GridSearchCV",
    estimator="sklearn.linear_model.LogisticRegression",
    scoring=None,
    cv=3,
    return_train_score=False,
    n_jobs=-1,
    param_grid=dict(
        C=[0.1, 1, 10],
        class_weight=[None],
        dual=[False],
        fit_intercept=[True],
        intercept_scaling=[1],
        max_iter=[100],
        multi_class=["ovr"],
        n_jobs=[1],
        penalty=["l1", "l2"],
        solver=["liblinear"],
        tol=[0.0001],
        warm_start=[False],
    ),
)
```

- **index_name** (str) – Index name of the pandas data frame after resetting the index. 'index' in default.

  If None, the index will not be reset.

- **partition_name** (str) – Additional index name to indicate the partition, train or test. 'partition' in default.

- **runner** (str) – If set to 'SequentialRunner' (default) or 'ParallelRunner', the pipeline is run by Kedro sequentially or in parallel, respectively.

  If set to None , the pipeline is run by native Python.

  Refer to https://kedro.readthedocs.io/en/latest/04_user_guide/05_nodes_and_pipelines.html#runners

- **conditionally_skip** (bool) – *[Effective only if runner is set to either 'SequentialRunner' or 'ParallelRunner']*

  Skip running the pipeline if the output files already exist. False in default.

- **df_print** – callable to use to show output data frames. IPython.display.display in default.

- **dataset_catalog** (Dict[str, AbstractDataSet]) – *[Effective only if runner is set to either 'SequentialRunner' or 'ParallelRunner']*

  Specify dataset files to save in Dict[str, kedro.io.AbstractDataSet] format.

  To find available file formats, refer to https://kedro.readthedocs.io/en/latest/kedro.io.html#data-sets

In default:

```
dict(
    # args_raw = CSVLocalDataSet(filepath='../data/01_raw/args_raw.csv
↪', version=None),
    # train_df = CSVLocalDataSet(filepath='../data/01_raw/train_df.csv
↪', version=None),
    # test_df = CSVLocalDataSet(filepath='../data/01_raw/test_df.csv',
↪ version=None),
    propensity_model  = PickleLocalDataSet(
        filepath='../data/06_models/propensity_model.pickle',
        version=None
    ),
    uplift_models_dict = PickleLocalDataSet(
        filepath='../data/06_models/uplift_models_dict.pickle',
        version=None
    ),
    df_03 = CSVLocalDataSet(
        filepath='../data/07_model_output/df.csv',
        load_args=dict(index_col=['partition', 'index'], float_
↪precision='high'),
        save_args=dict(index=True, float_format='%.16e'),
        version=None,
    ),
    treated__sim_eval_df = CSVLocalDataSet(
        filepath='../data/08_reporting/treated__sim_eval_df.csv',
        version=None,
    ),
    untreated__sim_eval_df = CSVLocalDataSet(
        filepath='../data/08_reporting/untreated__sim_eval_df.csv',
        version=None,
    ),
    estimated_effect_df = CSVLocalDataSet(
        filepath='../data/08_reporting/estimated_effect_df.csv',
        version=None,
    ),
)
```

- **logging_config** (Optional[Dict[str, Any]]) – Specify logging configuration.

  Refer to https://docs.python.org/3.6/library/logging.config.html#logging-config-dictschema

  In default:

```
{'disable_existing_loggers': False,
 'formatters': {
     'json_formatter': {
         'class': 'pythonjsonlogger.jsonlogger.JsonFormatter',
         'format': '[%(asctime)s|%(name)s|%(funcName)s|
↪%(levelname)s] %(message)s',
     },
     'simple': {
         'format': '[%(asctime)s|%(name)s|%(levelname)s] %(message)s
↪',
```

(continues on next page)

(continued from previous page)

```
    },
},
'handlers': {
    'console': {
        'class': 'logging.StreamHandler',
        'formatter': 'simple',
        'level': 'INFO',
        'stream': 'ext://sys.stdout',
    },
   'info_file_handler': {
       'class': 'logging.handlers.RotatingFileHandler',
       'level': 'INFO',
       'formatter': 'simple',
       'filename': './info.log',
       'maxBytes': 10485760, # 10MB
       'backupCount': 20,
       'encoding': 'utf8',
       'delay': True,
   },
    'error_file_handler': {
        'class': 'logging.handlers.RotatingFileHandler',
        'level': 'ERROR',
        'formatter': 'simple',
        'filename': './errors.log',
        'maxBytes': 10485760,  # 10MB
        'backupCount': 20,
        'encoding': 'utf8',
        'delay': True,
    },
},
'loggers': {
    'anyconfig': {
        'handlers': ['console', 'info_file_handler', 'error_file_
↪handler'],
        'level': 'WARNING',
        'propagate': False,
    },
    'kedro.io': {
        'handlers': ['console', 'info_file_handler', 'error_file_
↪handler'],
        'level': 'WARNING',
        'propagate': False,
    },
    'kedro.pipeline': {
        'handlers': ['console', 'info_file_handler', 'error_file_
↪handler'],
        'level': 'INFO',
        'propagate': False,
    },
    'kedro.runner': {
        'handlers': ['console', 'info_file_handler', 'error_file_
↪handler'],
```

(continues on next page)

(continued from previous page)

```
        'level': 'INFO',
        'propagate': False,
    },
    'causallift': {
        'handlers': ['console', 'info_file_handler', 'error_file_
↪handler'],
        'level': 'INFO',
        'propagate': False,
    },
},
'root': {
    'handlers': ['console', 'info_file_handler', 'error_file_handler
↪'],
    'level': 'INFO',
},
'version': 1}
```

**__init__**(*train_df=None, test_df=None, cols_features=None, col_treatment='Treatment', col_outcome='Outcome', col_propensity='Propensity', col_proba_if_treated='Proba_if_Treated', col_proba_if_untreated='Proba_if_Untreated', col_cate='CATE', col_recommendation='Recommendation', col_weight='Weight', min_propensity=0.01, max_propensity=0.99, verbose=2, uplift_model_params={'cv': 3, 'estimator': 'xgboost.XGBClassifier', 'n_jobs': -1, 'param_grid': {'base_score': [0.5], 'booster': ['gbtree'], 'colsample_bylevel': [1], 'colsample_bytree': [1], 'gamma': [0], 'learning_rate': [0.1], 'max_delta_step': [0], 'max_depth': [3], 'min_child_weight': [1], 'missing': [None], 'n_estimators': [100], 'n_jobs': [-1], 'nthread': [None], 'objective': ['binary:logistic'], 'random_state': [0], 'reg_alpha': [0], 'reg_lambda': [1], 'scale_pos_weight': [1], 'subsample': [1], 'verbose': [0]}, 'return_train_score': False, 'scoring': None, 'search_cv': 'sklearn.model_selection.GridSearchCV'}, enable_ipw=True, enable_weighting=False, propensity_model_params={'cv': 3, 'estimator': 'sklearn.linear_model.LogisticRegression', 'n_jobs': -1, 'param_grid': {'C': [0.1, 1, 10], 'class_weight': [None], 'dual': [False], 'fit_intercept': [True], 'intercept_scaling': [1], 'max_iter': [100], 'multi_class': ['ovr'], 'n_jobs': [1], 'penalty': ['l1', 'l2'], 'random_state': [0], 'solver': ['liblinear'], 'tol': [0.0001], 'warm_start': [False]}, 'return_train_score': False, 'scoring': None, 'search_cv': 'sklearn.model_selection.GridSearchCV'}, index_name='index', partition_name='partition', runner='SequentialRunner', conditionally_skip=False, df_print=<function display>, dataset_catalog={'df_03': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'estimated_effect_df': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'propensity_model': <kedro.extras.datasets.pickle.pickle_dataset.PickleDataSet object>, 'treated__sim_eval_df': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'untreated__sim_eval_df': <kedro.extras.datasets.pandas.csv_dataset.CSVDataSet object>, 'uplift_models_dict': <kedro.extras.datasets.pickle.pickle_dataset.PickleDataSet object>}, logging_config={'disable_existing_loggers': False, 'formatters': {'json_formatter': {'class': 'pythonjsonlogger.jsonlogger.JsonFormatter', 'format': '[%(asctime)s|%(name)s|%(funcName)s|%(levelname)s] %(message)s'}, 'simple': {'format': '[%(asctime)s|%(name)s|%(levelname)s] %(message)s'}}, 'handlers': {'console': {'class': 'logging.StreamHandler', 'formatter': 'simple', 'level': 'INFO', 'stream': 'ext://sys.stdout'}, 'error_file_handler': {'backupCount': 20, 'class': 'logging.handlers.RotatingFileHandler', 'delay': True, 'encoding': 'utf8', 'filename': './errors.log', 'formatter': 'simple', 'level': 'ERROR', 'maxBytes': 10485760}, 'info_file_handler': {'backupCount': 20, 'class': 'logging.handlers.RotatingFileHandler', 'delay': True, 'encoding': 'utf8', 'filename': './info.log', 'formatter': 'simple', 'level': 'INFO', 'maxBytes': 10485760}}, 'loggers': {'anyconfig': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'WARNING', 'propagate': False}, 'causallift': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'INFO', 'propagate': False}, 'kedro.io': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'WARNING', 'propagate': False}, 'kedro.pipeline': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'INFO', 'propagate': False}, 'kedro.runner': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'INFO', 'propagate': False}}, 'root': {'handlers': ['console', 'info_file_handler', 'error_file_handler'], 'level': 'INFO'}, 'version': 1}*)

**estimate_cate_by_2_models**()

Estimate CATE (Conditional Average Treatment Effect) using 2 XGBoost classifier models.

**Return type**
Tuple[DataFrame, DataFrame]

**estimate_recommendation_impact**(*cate_estimated=None*, *treatment_fraction_train=None*, *treatment_fraction_test=None*, *verbose=None*)

Estimate the impact of recommendation based on uplift modeling.

**Parameters**

- **cate_estimated** (Optional[Type[Series]]) – Pandas series containing the CATE. If None (default), use the ones calculated by estimate_cate_by_2_models method.
- **treatment_fraction_train** (Optional[float]) – The fraction of treatment in train dataset. If None (default), use the ones calculated by estimate_cate_by_2_models method.
- **treatment_fraction_test** (Optional[float]) – The fraction of treatment in test dataset. If None (default), use the ones calculated by estimate_cate_by_2_models method.
- **verbose** (Optional[int]) – How much info to show. If None (default), use the value set in the constructor.

**Return type**
Type[DataFrame]

### 25.1.4 causallift.generate_data module

The original code is at https://github.com/wayfair/pylift/blob/master/pylift/generate_data.py licensed under the BSD 2-Clause "Simplified" License Copyright 2018, Wayfair, Inc.

This code is an enhanced (backward-compatible) version that can simulate observational dataset including "sleeping dogs."

"Sleeping dogs" (people who will "buy" if not treated but will not "buy" if treated) can be simulated by negative values in tau parameter. Observational data which includes confounding can be simulated by non-zero values in propensity_coef parameter. A/B Test (RCT) with a 50:50 split can be simulated by all-zeros values in propensity_coef parameter (default). The first element in each list parameter specifies the intercept.

causallift.generate_data.**generate_data**(*N=1000*, *n_features=3*, *beta=[1, -2, 3, -0.8]*, *error_std=0.5*, *tau=3*, *discrete_outcome=False*)

Generates random data with a ground truth data generating process. Draws random values for features from [0, 1), errors from a 0-centered distribution with std *error_std*, and creates an outcome y.

**Parameters**

- **N** – (Optional[int]) - Number of observations.
- **n_features** – (Optional[int]) - Number of features.
- **beta** – (Optional[List[float]]) - Array of beta coefficients to multiply by X to get y.
- **error_std** – (Optional[float]) - Standard deviation (scale) of distribution from which errors are drawn.
- **tau** – (Union[List[float], float]) - Array of coefficients to multiply by X to get y if treated. More/larger negative values will simulate more "sleeping dogs" If float scalar is input, effect of features is not considered.
- **tau_std** – (Optional[float]) - When not None, draws tau from a normal distribution centered around tau with standard deviation tau_std rather than just using a constant value of tau.
- **discrete_outcome** – (Optional[bool]) - If True, outcomes are 0 or 1; otherwise continuous.
- **seed** – (Optional[int]) - Random seed fed to np.random.seed to allow for deterministic behavior.
- **feature_effect** – (Optional[float]) - Effect of beta on outcome if treated.
- **propensity_coef** – (Optional[List[float]]) - Array of coefficients to multiply by X to get propensity log-odds to be treated.

- **index_name** – (Optional[str]) - Index name in the output DataFrame. If None (default), index name will not be set.

**Returns**

**pd.DataFrame**
A DataFrame containing the generated data.

**Return type**
df

### 25.1.5 causallift.pipeline module

Pipeline construction.

causallift.pipeline.**create_pipeline**(***kwargs*)

Create the project's pipeline.

**Parameters**
**kwargs** – Ignore any additional arguments added in the future.

**Returns**
The resulting pipeline.

**Return type**
Pipeline

### 25.1.6 causallift.run module

CHAPTER

# TWENTYSIX

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## C

# INDEX

## Symbols



## A



## B



## C



## E



## F



## G