lifetimes Documentation

Release 0.11.1

Cameron Davidson-Pilon

Mar 08, 2019

Contents:


LIFETIMES

CHAPTER 1

Introduction

Lifetimes can be used to analyze your users based on a few assumption:

1. Users interact with you when they are "alive".
2. Users under study may "die" after some period of time.

I've quoted "alive" and "die" as these are the most abstract terms: feel free to use your own definition of "alive" and "die" (they are used similarly to "birth" and "death" in survival analysis). Whenever we have individuals repeating occurrences, we can use Lifetimes to help understand user behaviour.

1.1 Applications

If this is too abstract, consider these applications:

- Predicting how often a visitor will return to your website. (Alive = visiting. Die = decided the website wasn't for them)
- Understanding how frequently a patient may return to a hospital. (Alive = visiting. Die = maybe the patient moved to a new city, or became deceased.)
- Predicting individuals who have churned from an app using only their usage history. (Alive = logins. Die = removed the app)
- Predicting repeat purchases from a customer. (Alive = actively purchasing. Die = became disinterested with your product)
- Predicting the lifetime value of your customers

1.2 Specific Application: Customer Lifetime Value

As emphasized by P. Fader and B. Hardie, understanding and acting on customer lifetime value (CLV) is the most important part of your business's sales efforts. And (apparently) everyone is doing it wrong. *Lifetimes* is a Python library to calculate CLV for you.

CHAPTER 2

Installation

```
pip install lifetimes
```

CHAPTER 3

Documentation and tutorials

Official documentation

CHAPTER 4

Questions? Comments? Requests?

Please create an issue in the lifetimes repository.

CHAPTER 5

More Information

1. Roberto Medri did a nice presentation on CLV at Etsy.
2. Papers, lots of papers.
3. R implementation is called BTYD (for, *Buy 'Til You Die*).

5.1 Quickstart

For the following examples, we'll use a dataset from an ecommerce provider to analyze their customers' repeat purchases. The examples below are using the `cdnow_customers.csv` located in the `datasets/` directory.

```
from lifetimes.datasets import load_cdnow_summary
data = load_cdnow_summary(index_col=[0])

print(data.head())
"""
     frequency   recency      T
ID
1    2           30.43       38.86
2    1            1.71       38.86
3    0            0.00       38.86
4    0            0.00       38.86
5    0            0.00       38.86
"""
```

5.1.1 The shape of your data

For all models, the following nomenclature is used:

- `frequency` represents the number of *repeat* purchases the customer has made. This means that it's one less than the total number of purchases. This is actually slightly wrong. It's the count of time periods the customer had a purchase in. So if using days as units, then it's the count of days the customer had a purchase on.

- `T` represents the age of the customer in whatever time units chosen (weekly, in the above dataset). This is equal to the duration between a customer's first purchase and the end of the period under study.
- `recency` represents the age of the customer when they made their most recent purchases. This is equal to the duration between a customer's first purchase and their latest purchase. (Thus if they have made only 1 purchase, the recency is 0.)

If your data is not in the format (very common), there are *utility functions* in lifetimes to transform your data to look like this

5.1.2 Basic Frequency/Recency analysis using the BG/NBD model

We'll use the **BG/NBD model** first. There are other models which we will explore in these docs, but this is the simplest to start with.

```
from lifetimes import BetaGeoFitter

# similar API to scikit-learn and lifelines.
bgf = BetaGeoFitter(penalizer_coef=0.0)
bgf.fit(data['frequency'], data['recency'], data['T'])
print(bgf)
"""
<lifetimes.BetaGeoFitter: fitted with 2357 subjects, a: 0.79, alpha: 4.41, b: 2.43,␣
↪r: 0.24>
"""

bgf.summary
"""
             coef  se(coef)  lower 95% bound  upper 95% bound
r        0.242593  0.012557         0.217981         0.267205
alpha    4.413532  0.378221         3.672218         5.154846
a        0.792886  0.185719         0.428877         1.156895
b        2.425752  0.705345         1.043276         3.808229
"""
```

After fitting, we have lots of nice methods and properties attached to the fitter object, like `param_` and `summary`.

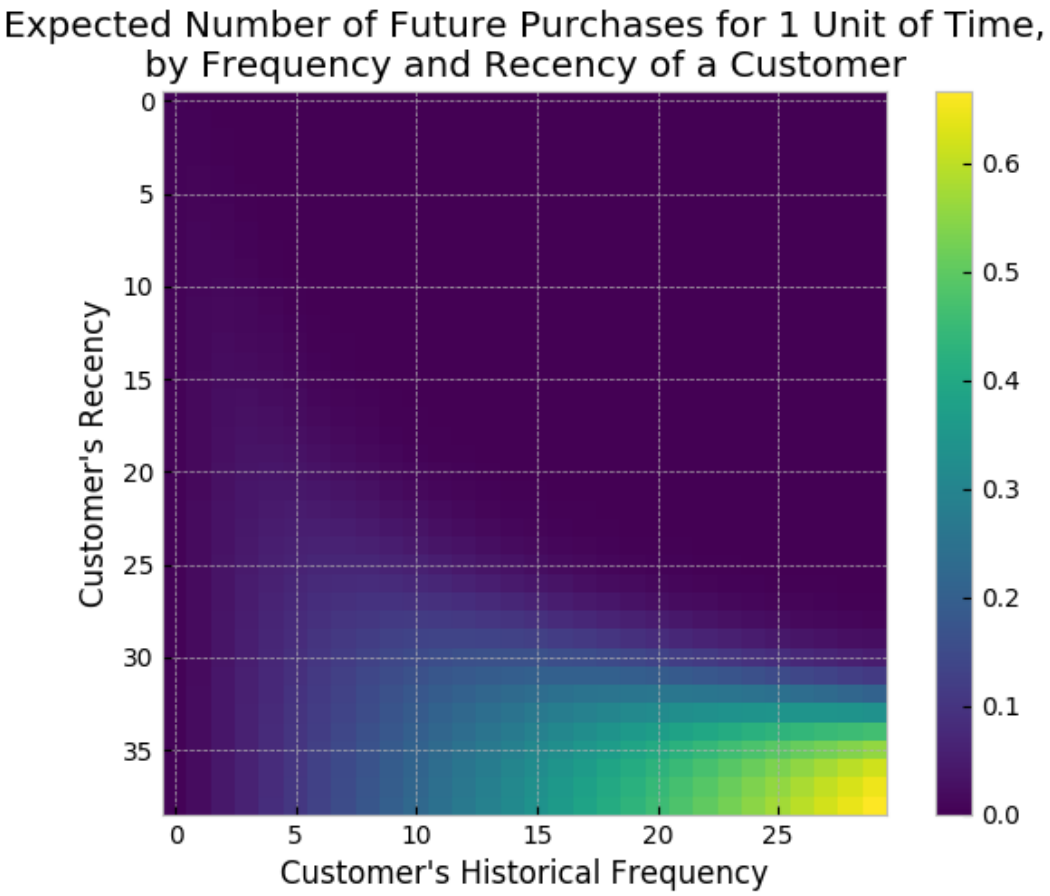
For small samples sizes, the parameters can get implausibly large, so by adding an l2 penalty the likelihood, we can control how large these parameters can be. This is implemented as setting as positive `penalizer_coef` in the initialization of the model. In typical applications, penalizers on the order of 0.001 to 0.1 are effective.

Visualizing our Frequency/Recency Matrix

Consider: a customer bought from you every day for three weeks straight, and we haven't heard from them in months. What are the chances they are still "alive"? Pretty small. On the other hand, a customer who historically buys from you once a quarter, and bought last quarter, is likely still alive. We can visualize this relationship using the **Frequency/Recency matrix**, which computes the expected number of transactions a artificial customer is to make in the next time period, given his or her recency (age at last purchase) and frequency (the number of repeat transactions he or she has made).

```
from lifetimes.plotting import plot_frequency_recency_matrix

plot_frequency_recency_matrix(bgf)
```



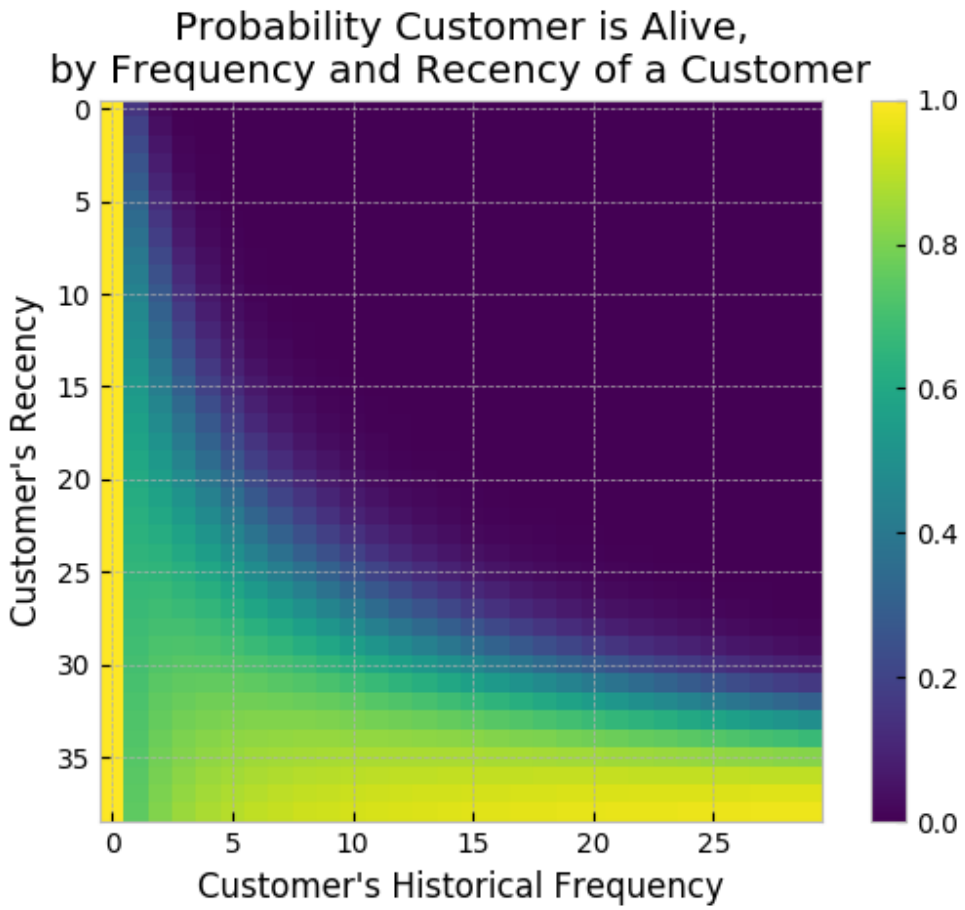
We can see that if a customer has bought 25 times from you, and their latest purchase was when they were 35 weeks old (given the individual is 35 weeks old), then they are your best customer (bottom-right). Your coldest customers are those that are in the top-right corner: they bought a lot quickly, and we haven't seen them in weeks.

There's also that beautiful "tail" around (5,25). That represents the customer who buys infrequently, but we've seen him or her recently, so they *might* buy again - we're not sure if they are dead or just between purchases.

Another interesting matrix to look at is the probability of still being *alive*:

```
from lifetimes.plotting import plot_probability_alive_matrix

plot_probability_alive_matrix(bgf)
```



Ranking customers from best to worst

Let's return to our customers and rank them from "highest expected purchases in the next period" to lowest. Models expose a method that will predict a customer's expected purchases in the next period using their history.

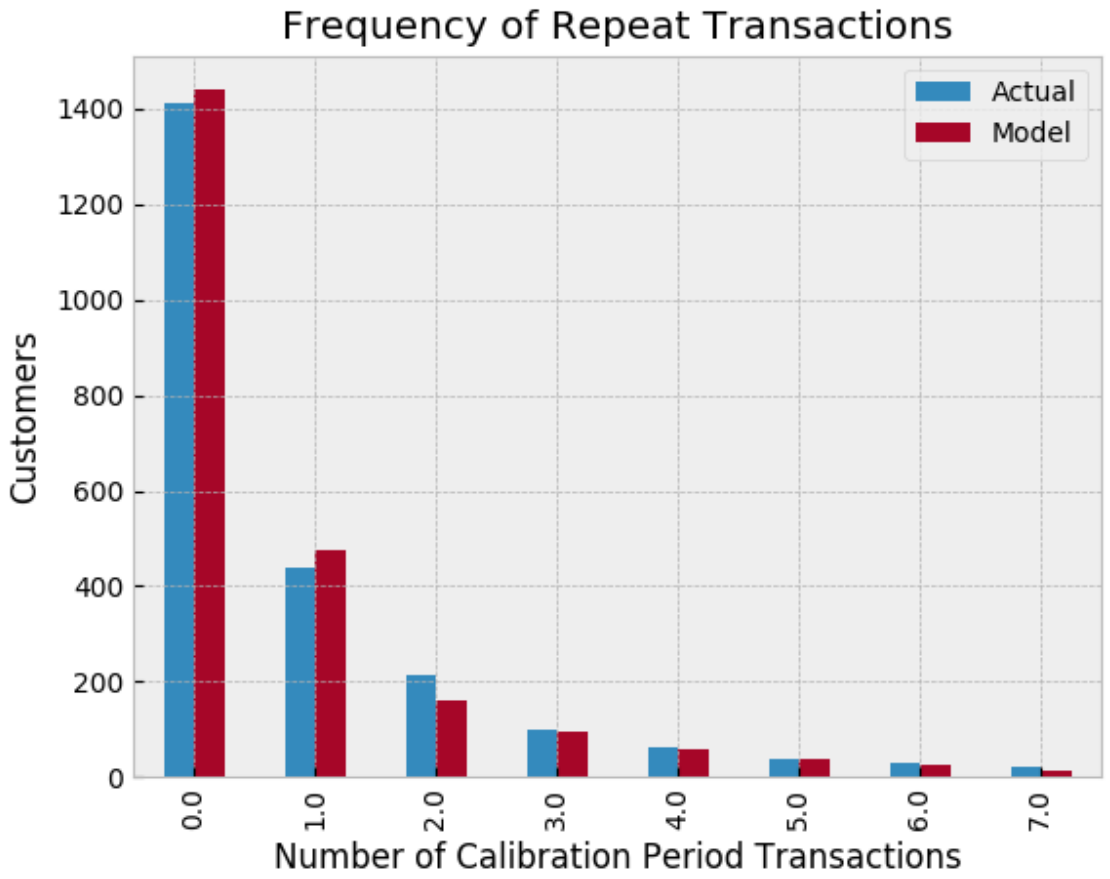
```
t = 1
data['predicted_purchases'] = bgf.conditional_expected_number_of_purchases_up_to_
↪time(t, data['frequency'], data['recency'], data['T'])
data.sort_values(by='predicted_purchases').tail(5)
"""
       frequency  recency      T        predicted_purchases
ID
509    18         35.14        35.86    0.424877
841    19         34.00        34.14    0.474738
1981   17         28.43        28.86    0.486526
157    29         37.71        38.00    0.662396
1516   26         30.86        31.00    0.710623
"""
```

Great, we can see that the customer who has made 26 purchases, and bought very recently from us, is probably going to buy again in the next period.

Assessing model fit

Ok, we can predict and we can visualize our customers' behaviour, but is our model correct? There are a few ways to assess the model's correctness. The first is to compare your data versus artificial data simulated with your fitted model's parameters.

```
from lifetimes.plotting import plot_period_transactions
plot_period_transactions(bgf)
```



We can see that our actual data and our simulated data line up well. This proves that our model doesn't suck.

Example using transactional datasets

Most often, the dataset you have at hand will be at the transaction level. Lifetimes has some utility functions to transform that transactional data (one row per purchase) into summary data (a frequency, recency and age dataset).

```
from lifetimes.datasets import load_transaction_data
from lifetimes.utils import summary_data_from_transaction_data

transaction_data = load_transaction_data()
print(transaction_data.head())
"""
                  date  id
0  2014-03-08 00:00:00   0
```

(continues on next page)

(continued from previous page)

```
1  2014-05-21 00:00:00   1
2  2014-03-14 00:00:00   2
3  2014-04-09 00:00:00   2
4  2014-05-21 00:00:00   2
"""

summary = summary_data_from_transaction_data(transaction_data, 'id', 'date',␣
↪observation_period_end='2014-12-31')

print(summary.head())
"""
frequency  recency      T
id
0          0.0      0.0  298.0
1          0.0      0.0  224.0
2          6.0    142.0  292.0
3          0.0      0.0  147.0
4          2.0      9.0  183.0
"""

bgf.fit(summary['frequency'], summary['recency'], summary['T'])
# <lifetimes.BetaGeoFitter: fitted with 5000 subjects, a: 1.85, alpha: 1.86, b: 3.18,␣
↪r: 0.16>
```

More model fitting

With transactional data, we can partition the dataset into a calibration period dataset and a holdout dataset. This is important as we want to test how our model performs on data not yet seen (think cross-validation in standard machine learning literature). Lifetimes has a function to partition our dataset like this:

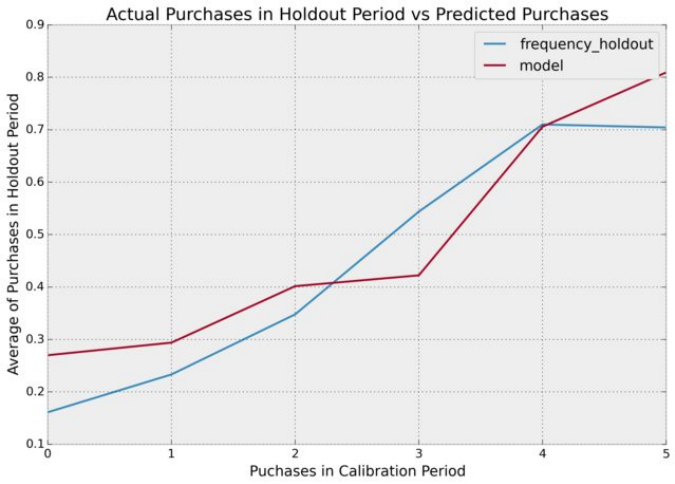
```
from lifetimes.utils import calibration_and_holdout_data

summary_cal_holdout = calibration_and_holdout_data(transaction_data, 'id', 'date',
                                        calibration_period_end='2014-09-01',
                                        observation_period_end='2014-12-31' )
print(summary_cal_holdout.head())
"""
    frequency_cal  recency_cal  T_cal  frequency_holdout  duration_holdout
id
0             0.0          0.0  177.0                0.0               121
1             0.0          0.0  103.0                0.0               121
2             6.0        142.0  171.0                0.0               121
3             0.0          0.0   26.0                0.0               121
4             2.0          9.0   62.0                0.0               121
"""
```

With this dataset, we can perform fitting on the `_cal` columns, and test on the `_holdout` columns:

```
from lifetimes.plotting import plot_calibration_purchases_vs_holdout_purchases

bgf.fit(summary_cal_holdout['frequency_cal'], summary_cal_holdout['recency_cal'],␣
↪summary_cal_holdout['T_cal'])
plot_calibration_purchases_vs_holdout_purchases(bgf, summary_cal_holdout)
```



Customer Predictions

Based on customer history, we can predict what an individuals future purchases might look like:

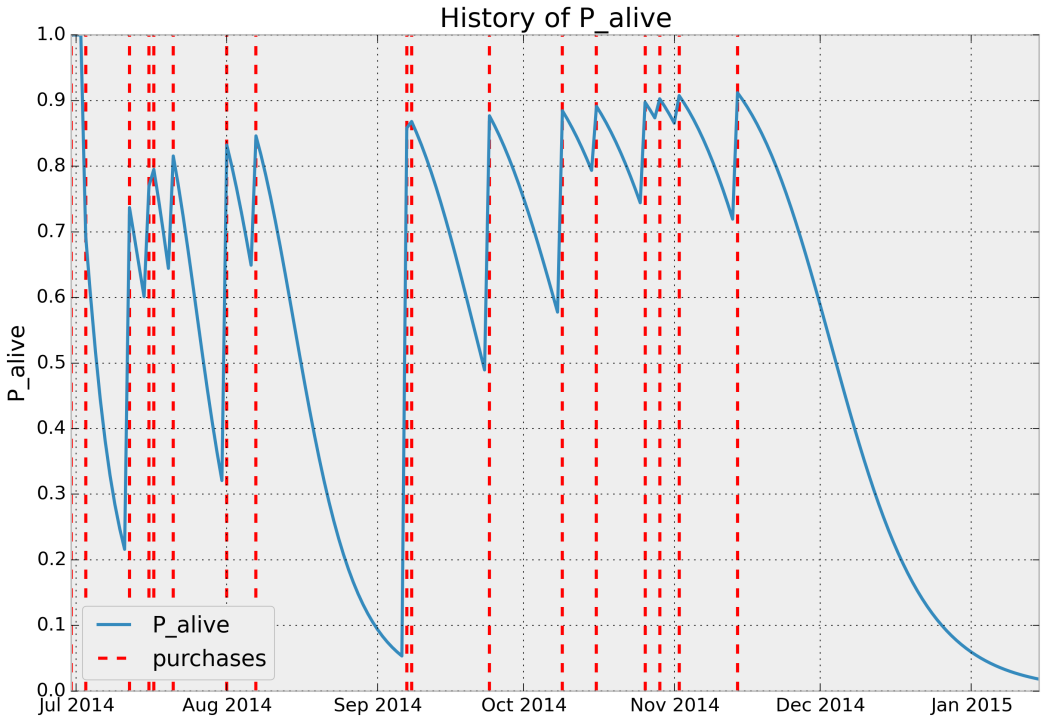
```
t = 10 #predict purchases in 10 periods
individual = summary.iloc[20]
# The below function is an alias to `bfg.conditional_expected_number_of_purchases_up_
↪to_time`
bgf.predict(t, individual['frequency'], individual['recency'], individual['T'])
# 0.0576511
```

Customer Probability Histories

Given a customer transaction history, we can calculate their historical probability of being alive, according to our trained model. For example:

```
from lifetimes.plotting import plot_history_alive

id = 35
days_since_birth = 200
sp_trans = transaction_data.loc[transaction_data['id'] == id]
plot_history_alive(bgf, days_since_birth, sp_trans, 'date')
```



5.1.3 Estimating customer lifetime value using the Gamma-Gamma model

For this whole time we didn't take into account the economic value of each transaction and we focused mainly on transactions' occurrences. To estimate this we can use the Gamma-Gamma submodel. But first we need to create summary data from transactional data also containing economic values for each transaction (i.e. profits or revenues).

```
from lifetimes.datasets import load_cdnow_summary_data_with_monetary_value

summary_with_money_value = load_cdnow_summary_data_with_monetary_value()
summary_with_money_value.head()
returning_customers_summary = summary_with_money_value[summary_with_money_value[
↪'frequency']>0]

print(returning_customers_summary.head())
"""
             frequency  recency      T  monetary_value
customer_id
1                    2    30.43  38.86           22.35
2                    1     1.71  38.86           11.77
6                    7    29.43  38.86           73.74
7                    1     5.00  38.86           11.77
9                    2    35.71  38.86           25.55
"""
```

The Gamma-Gamma model and the independence assumption

The model we are going to use to estimate the CLV for our userbase is called the Gamma-Gamma submodel, which relies upon an important assumption. The Gamma-Gamma submodel, in fact, assumes that there is no relationship between the monetary value and the purchase frequency. In practice we need to check whether the Pearson correlation between the two vectors is close to 0 in order to use this model.

```
returning_customers_summary[['monetary_value', 'frequency']].corr()
"""
                monetary_value  frequency
monetary_value        1.000000   0.113884
frequency             0.113884   1.000000
"""
```

At this point we can train our Gamma-Gamma submodel and predict the conditional, expected average lifetime value of our customers.

```
from lifetimes import GammaGammaFitter

ggf = GammaGammaFitter(penalizer_coef = 0)
ggf.fit(returning_customers_summary['frequency'],
        returning_customers_summary['monetary_value'])
print(ggf)
"""
<lifetimes.GammaGammaFitter: fitted with 946 subjects, p: 6.25, q: 3.74, v: 15.45>
"""
```

We can now estimate the average transaction value:

```
print(ggf.conditional_expected_average_profit(
        summary_with_money_value['frequency'],
        summary_with_money_value['monetary_value']
    ).head(10))
"""
customer_id
1     24.658619
2     18.911489
3     35.170981
4     35.170981
5     35.170981
6     71.462843
7     18.911489
8     35.170981
9     27.282408
10    35.170981
dtype: float64
"""

print("Expected conditional average profit: %s, Average profit: %s" % (
    ggf.conditional_expected_average_profit(
        summary_with_money_value['frequency'],
        summary_with_money_value['monetary_value']
    ).mean(),
    summary_with_money_value[summary_with_money_value['frequency']>0]['monetary_value
↪'].mean()
))
"""
```

(continues on next page)

(continued from previous page)

```
Expected conditional average profit: 35.2529588256, Average profit: 35.078551797
"""
```

While for computing the total CLV using the DCF method (https://en.wikipedia.org/wiki/Discounted_cash_flow) adjusting for cost of capital:

```
# refit the BG model to the summary_with_money_value dataset
bgf.fit(summary_with_money_value['frequency'], summary_with_money_value['recency'],␣
↪summary_with_money_value['T'])

print(ggf.customer_lifetime_value(
    bgf, #the model to use to predict the number of future transactions
    summary_with_money_value['frequency'],
    summary_with_money_value['recency'],
    summary_with_money_value['T'],
    summary_with_money_value['monetary_value'],
    time=12, # months
    discount_rate=0.01 # monthly discount rate ~ 12.7% annually
).head(10))
"""
customer_id
1       140.096211
2        18.943467
3        38.180574
4        38.180574
5        38.180574
6      1003.868107
7        28.109683
8        38.180574
9       167.418216
10       38.180574
Name: clv, dtype: float64
"""
```

5.2 Saving and loading model

When you have lots of data and training takes a lot of time option with saving and loading model could be useful. First you need to fit the model, then save it and load.

5.2.1 Fit model

```
from lifetimes import BetaGeoFitter
from lifetimes.datasets import load_cdnow_summary

data = load_cdnow_summary(index_col=[0])
bgf = BetaGeoFitter()
bgf.fit(data['frequency'], data['recency'], data['T'])
bgf
"""<lifetimes.BetaGeoFitter: fitted with 2357 subjects, a: 0.79, alpha: 4.41, b: 2.43,
↪ r: 0.24>"""
```

5.2.2 Saving model

Model will be saved with dill to pickle object. Optional parameters `save_data` and `save_generate_data_method` are present to reduce final pickle object size for big dataframes. Optional parameters:

- `save_data` is used for saving data from model or not (default: `True`).
- `save_generate_data_method` is used for saving `generate_new_data` method from model or not (default: `True`)

```
bgf.save_model('bgf.pkl')
```

or to save only model with minumum size without `data` and `generate_new_data`:

```
bgf.save_model('bgf_small_size.pkl', save_data=False, save_generate_data_method=False)
```

5.2.3 Loading model

Before loading you should initialize the model first and then use method `load_model`

```
bgf_loaded = BetaGeoFitter()
bgf_loaded.load_model('bgf.pkl')
bgf_loaded
"""<lifetimes.BetaGeoFitter: fitted with 2357 subjects, a: 0.79, alpha: 4.41, b: 2.43,
↪ r: 0.24>"""
```

5.3 More Examples and recipes

5.3.1 Example SQL statement to transform transactional data into RFM data

Let's review what our variables mean:

- `frequency` represents the number of *repeat* purchases the customer has made. This means that it's one less than the total number of purchases. This is actually slightly wrong. It's the count of distinct time periods the customer had a purchase in. So if using days as units, then it's the count of distinct days the customer had a purchase on.
- `T` represents the age of the customer in whatever time units chosen. This is equal to the duration between a customer's first purchase and the end of the period under study.
- `recency` represents the age of the customer when they made their most recent purchases. This is equal to the duration between a customer's first purchase and their latest purchase. (Thus if they have made only 1 purchase, the recency is 0.)

Thus, executing a query against a transactional dataset, called `orders`, in a SQL-store may look like:

```
SELECT
  customer_id,
  COUNT(distinct date(transaction_at)) - 1 as frequency,
  datediff('day', MIN(transaction_at), MAX(transaction_at)) as recency,
  datediff('day', CURRENT_DATE, MIN(transaction_at)) as T
FROM orders
GROUP BY customer_id
```

5.3.2 Create table with RFM summary matrix with holdout

Variables `frequency`, `T` and `recency` have the same meaning as in previous section.

Two variables to set before executing:

- `duration_holdout` - holdout duration in days.
- `CURRENT_DATE` - current date, could be changed to final date of the transactional data.

```
select
    a.*,
    COALESCE(b.frequency_holdout, 0) as frequency_holdout,
    duration_holdout as duration_holdout
from (
    select
        customer_id,
        datediff(max(event_date), min(event_date)) as recency,
        count(*) - 1 as frequency,
        datediff(date_sub(CURRENT_DATE, duration_holdout), min(event_date)) as T
    from orders
    where event_date < date_sub(CURRENT_DATE, duration_holdout)
    group by customer_id
) a
left join (
    select
        customer_id,
        count(*) as frequency_holdout
    from orders
    where event_date >= date_sub(CURRENT_DATE, duration_holdout)
      and event_date < CURRENT_DATE
    group by customer_id
) b
on a.customer_id = b.customer_id
```

5.4 lifetimes package

5.4.1 Subpackages

lifetimes.datasets package

Module contents

`lifetimes.datasets.`**`load_cdnow_summary`**`(**kwargs)`
Load cdnow customers summary pandas DataFrame.

`lifetimes.datasets.`**`load_transaction_data`**`(**kwargs)`
Return a Pandas dataframe of transactional data.

Looks like:

date id

0 2014-03-08 00:00:00 0 1 2014-05-21 00:00:00 1 2 2014-03-14 00:00:00 2 3 2014-04-09 00:00:00 2 4 2014-05-21 00:00:00 2

The data was artificially created using Lifetimes data generation routines. Data was generated between 2014-01-01 to 2014-12-31.

lifetimes.datasets.**load_cdnow_summary_data_with_monetary_value**(***kwargs*)
Load cdnow customers summary with monetary value as pandas DataFrame.

lifetimes.datasets.**load_donations**(***kwargs*)
Load donations dataset as pandas DataFrame.

lifetimes.fitters package

Submodules

lifetimes.fitters.base_fitter module

lifetimes.fitters.beta_geo_beta_binom_fitter module

Beta Geo Beta BinomFitter.

class lifetimes.fitters.beta_geo_beta_binom_fitter.**BetaGeoBetaBinomFitter**(*penalizer_coef=0.0*)
Bases: *lifetimes.fitters.BaseFitter*

Also known as the Beta-Geometric/Beta-Binomial Model[1].

Future purchases opportunities are treated as discrete points in time. In the literature, the model provides a better fit than the Pareto/NBD model for a nonprofit organization with regular giving patterns.

The model is estimated with a recency-frequency matrix with n transaction opportunities.

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: Series – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

[1] Fader, Peter S., Bruce G.S. Hardie, and Jen Shang (2010), "Customer-Base Analysis in a Discrete-Time Noncontractual Setting," Marketing Science, 29 (6), 1086-1108.

References

conditional_expected_number_of_purchases_up_to_time(*m_periods_in_future*, *frequency*, *recency*, *n_periods*)

Conditional expected purchases in future time period.

The expected number of future transactions across the next m_periods_in_future transaction opportunities by a customer with purchase history (x, tx, n).

$$E(X(n_{periods}, n_{periods} + m_{periods_in future})|\alpha, \beta, \gamma, \delta, frequency, recency, n_{periods})$$

See (13) in Fader & Hardie 2010.

Parameters **t** (*array_like*) – time n_periods (n+t)

Returns *array_like* – predicted transactions

conditional_probability_alive(*m_periods_in_future*, *frequency*, *recency*, *n_periods*)

Conditional probability alive.

Conditional probability customer is alive at transaction opportunity n_periods + m_periods_in_future.

$$P(aliveatn_periods + m_periods_in future|alpha, beta, gamma, delta, frequency, recency, n_periods)$$

See (A10) in Fader and Hardie 2010.

Parameters **m** (*array_like*) – transaction opportunities

Returns *array_like* – alive probabilities

expected_number_of_transactions_in_first_n_periods(*n*)

Return expected number of transactions in first n n_periods.

Expected number of transactions occurring across first n transaction opportunities. Used by Fader and Hardie to assess in-sample fit.

$$Pr(X(n) = x|\alpha, \beta, \gamma, \delta)$$

See (7) in Fader & Hardie 2010.

Parameters **n** (*float*) – number of transaction opportunities

Returns *DataFrame* – Predicted values, indexed by x

fit(*frequency*, *recency*, *n_periods*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, ***kwargs*)

Fit the BG/BB model.

Parameters

- **frequency** (*array_like*) – Total periods with observed transactions
- **recency** (*array_like*) – Period of most recent transaction
- **n_periods** (*array_like*) – Number of transaction opportunities. Previously called *n*.
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual log-likelihood, the log-likelihood is calculated for each pattern and multiplied by the number of customers with that pattern. Previously called *n_custs*.

- **verbose** (*boolean, optional*) – Set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – Tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – Index for resulted DataFrame which is accessible via self.data
- **kwargs** – Key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *BetaGeoBetaBinomFitter* – fitted and with parameters estimated

lifetimes.fitters.beta_geo_fitter module

Beta Geo Fitter, also known as BG/NBD model.

class lifetimes.fitters.beta_geo_fitter.**BetaGeoFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Also known as the BG/NBD model.

Based on[2], this model has the following assumptions:

1. Each individual, i, has a hidden lambda_i and p_i parameter
2. These come from a population wide Gamma and a Beta distribution respectively.
3. Individuals purchases follow a Poisson process with rate lambda_i*t .
4. After each purchase, an individual has a p_i probability of dieing (never buying again).

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: Series – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

[2] Fader, Peter S., Bruce G.S. Hardie, and Ka Lok Lee (2005a), "Counting Your Customers the Easy Way: An Alternative to the Pareto/NBD Model," Marketing Science, 24 (2), 275-84.

References

conditional_expected_number_of_purchases_up_to_time(*t*, *frequency*, *recency*, *T*)
Conditional expected number of purchases up to time.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population, given they have purchase history (frequency, recency, T)

Parameters

- **t** (*array_like*) – times to calculate the expectation for.
- **frequency** (*array_like*) – historical frequency of customer.
- **recency** (*array_like*) – historical recency of customer.
- **T** (*array_like*) – age of the customer.

Returns *array_like*

conditional_probability_alive(*frequency*, *recency*, *T*)
Compute conditional probability alive.

Compute the probability that a customer with history (frequency, recency, T) is currently alive.

From http://www.brucehardie.com/notes/021/palive_for_BGNBD.pdf

Parameters

- **frequency** (*array or scalar*) – historical frequency of customer.
- **recency** (*array or scalar*) – historical recency of customer.
- **T** (*array or scalar*) – age of the customer.

Returns *array* – value representing a probability

conditional_probability_alive_matrix(*max_frequency=None*, *max_recency=None*)
Compute the probability alive matrix.

Parameters

- **max_frequency** (*float, optional*) – the maximum frequency to plot. Default is max observed frequency.
- **max_recency** (*float, optional*) – the maximum recency to plot. This also determines the age of the customer. Default to max observed age.

Returns *matrix* – A matrix of the form [t_x: historical recency, x: historical frequency]

expected_number_of_purchases_up_to_time(*t*)
Calculate the expected number of repeat purchases up to time t.

Calculate repeat purchases for a randomly choose individual from the population.

Parameters **t** (*array_like*) – times to calculate the expection for

Returns *array_like*

fit(*frequency*, *recency*, *T*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, ***kwargs*)
Fit a dataset to the BG/NBD model.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).

- **recency** (*array_like*) – the recency vector of customers' purchases (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual loglikelihood, the loglikelihood is calculated for each pattern and multiplied by the number of customers with that pattern.
- **initial_params** (*array_like, optional*) – set the initial parameters for the fitter.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *BetaGeoFitter* – with additional properties like `params_` and methods like `predict`

probability_of_n_purchases_up_to_time(*t*, *n*)

Compute the probability of n purchases.

$$P(N(t) = n|\text{model})$$

where N(t) is the number of repeat purchases a customer makes in t units of time.

Parameters

- **t** (*float*) – number units of time
- **n** (*int*) – number of purchases

Returns *float* – Probability to have n purchases up to t units of time

lifetimes.fitters.gamma_gamma_fitter module

class lifetimes.fitters.gamma_gamma_fitter.**GammaGammaFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Fitter for the gamma-gamma model.

It is used to estimate the average monetary value of customer transactions.

This implementation is based on the Excel spreadsheet found in[3]. More details on the derivation and evaluation can be found in[4].

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

[3] http://www.brucehardie.com/notes/025/ The Gamma-Gamma Model of Monetary Value.

[4] Peter S. Fader, Bruce G. S. Hardie, and Ka Lok Lee (2005), "RFM and CLV: Using iso-value curves for customer base analysis", Journal of Marketing Research, 42 (November), 415-430.

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: OrderedDict – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the columns given in the call to *fit*

References

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: Series – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

conditional_expected_average_profit (*frequency=None*, *monetary_value=None*)
Conditional expectation of the average profit.

This method computes the conditional expectation of the average profit per transaction for a group of one or more customers.

Parameters

- **frequency** (*array_like, optional*) – a vector containing the customers' frequencies. Defaults to the whole set of frequencies used for fitting the model.
- **monetary_value** (*array_like, optional*) – a vector containing the customers' monetary values. Defaults to the whole set of monetary values used for fitting the model.

Returns *array_like* – The conditional expectation of the average profit per transaction

customer_lifetime_value (*transaction_prediction_model*, *frequency*, *recency*, *T*, *monetary_value*, *time=12*, *discount_rate=0.01*, *freq='D'*)
Return customer lifetime value.

This method computes the average lifetime value for a group of one or more customers.

Parameters

- **transaction_prediction_model** (*model*) – the model to predict future transactions, literature uses pareto/ndb models but we can also use a different model like beta-geo models
- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).

- **recency** (*the recency vector of customers' purchases*) – (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **monetary_value** (*array_like*) – the monetary value vector of customer's purchases (denoted m in literature).
- **time** (*float, optional*) – the lifetime expected for the user in months. Default: 12
- **discount_rate** (*float, optional*) – the monthly adjusted discount rate. Default: 0.01
- **freq** (*string, optional*) – {"D", "H", "M", "W"} for day, hour, month, week. This represents what unit of time your T is measure in.

Returns *Series* – Series object with customer ids as index and the estimated customer lifetime values as values

fit (*frequency*, *monetary_value*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, *q_constraint=False*, ***kwargs*)
Fit the data to the Gamma/Gamma model.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **monetary_value** (*array_like*) – the monetary value vector of customer's purchases (denoted m in literature).
- **weights** (*None or array_like*) – Number of customers with given frequency/monetary_value, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/monetary_value. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual loglikelihood, the loglikelihood is calculated for each pattern and multiplied by the number of customers with that pattern.
- **initial_params** (*array_like, optional*) – set the initial parameters for the fitter.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **q_constraint** (*bool, optional*) – when q < 1, population mean will result in a negative value leading to negative CLV outputs. If True, we penalize negative values of q to avoid this issue.
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *GammaGammaFitter* – fitted and with parameters estimated

lifetimes.fitters.modified_beta_geo_fitter module

class lifetimes.fitters.modified_beta_geo_fitter.**ModifiedBetaGeoFitter** (*penalizer_coef=0.0*)
Bases: *lifetimes.fitters.beta_geo_fitter.BetaGeoFitter*

Also known as the MBG/NBD model.

Based on[5],[6], this model has the following assumptions: 1) Each individual, `i`, has a hidden `lambda_i` and `p_i` parameter 2) These come from a population wide Gamma and a Beta distribution

respectively.

3. Individuals purchases follow a Poisson process with rate $\lambda_i * t$.
4. At the beginning of their lifetime and after each purchase, an individual has a p_i probability of dieing (never buying again).

References

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: Series – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

conditional_expected_number_of_purchases_up_to_time (*t*, *frequency*, *recency*, *T*)
Conditional expected number of repeat purchases up to time t.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population, given they have purchase history (frequency, recency, T) See Wagner, U. and Hoppe D. (2008).

Parameters

- **t** (*array_like*) – times to calculate the expectation for.
- **frequency** (*array_like*) – historical frequency of customer.
- **recency** (*array_like*) – historical recency of customer.
- **T** (*array_like*) – age of the customer.

Returns *array_like*

conditional_probability_alive (*frequency*, *recency*, *T*)
Conditional probability alive.

Compute the probability that a customer with history (frequency, recency, T) is currently alive. From https://www.researchgate.net/publication/247219660_Empirical_validation_and_comparison_of_models_for_customer_base_analysis Appendix A, eq. (5)

[5] Batislam, E.P., M. Denizel, A. Filiztekin (2007), "Empirical validation and comparison of models for customer base analysis," International Journal of Research in Marketing, 24 (3), 201-209.

[6] Wagner, U. and Hoppe D. (2008), "Erratum on the MBG/NBD Model," International Journal of Research in Marketing, 25 (3), 225-226.

Parameters

- **frequency** (*array or float*) – historical frequency of customer.
- **recency** (*array or float*) – historical recency of customer.
- **T** (*array or float*) – age of the customer.

Returns *array* – value representing probability of being alive

expected_number_of_purchases_up_to_time(*t*)
Return expected number of repeat purchases up to time t.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population.

Parameters **t** (*array_like*) – times to calculate the expectation for

Returns *array_like*

fit(*frequency*, *recency*, *T*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, ***kwargs*)
Fit the data to the MBG/NBD model.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **recency** (*array_like*) – the recency vector of customers' purchases (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual log-likelihood, the log-likelihood is calculated for each pattern and multiplied by the number of customers with that pattern.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *ModifiedBetaGeoFitter* – With additional properties and methods like `params_` and `predict`

probability_of_n_purchases_up_to_time(*t*, *n*)
Compute the probability of n purchases up to time t.

$$P(N(t) = n|\text{model})$$

where N(t) is the number of repeat purchases a customer makes in t units of time.

Parameters

- **t** (*float*) – number units of time

- **n** (*int*) – number of purchases

Returns *float* – Probability to have n purchases up to t units of time

lifetimes.fitters.pareto_nbd_fitter module

Pareto/NBD model.

class lifetimes.fitters.pareto_nbd_fitter.**ParetoNBDFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Pareto NBD fitter[7].

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: OrderedDict – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the columns given in the call to *fit*

References

conditional_expected_number_of_purchases_up_to_time(*t*, *frequency*, *recency*, *T*)
Conditional expected number of purchases up to time.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population, given they have purchase history (frequency, recency, T)

Parameters

- **t** (*array_like*) – times to calculate the expectation for.
- **frequency** (*array_like*) – historical frequency of customer.
- **recency** (*array_like*) – historical recency of customer.
- **T** (*array_like*) – age of the customer.

Returns *array_like*

conditional_probability_alive(*frequency*, *recency*, *T*)
Conditional probability alive.

Compute the probability that a customer with history (frequency, recency, T) is currently alive. From paper: http://brucehardie.com/notes/009/pareto_nbd_derivations_2005-11-05.pdf

Parameters

- **frequency** (*float*) – historical frequency of customer.
- **recency** (*float*) – historical recency of customer.
- **T** (*float*) – age of the customer.

[7] **David C. Schmittlein, Donald G. Morrison and Richard Colombo** Management Science,Vol. 33, No. 1 (Jan., 1987), pp. 1-24 "Counting Your Customers: Who Are They and What Will They Do Next,"

Returns *float* – value representing a probability

conditional_probability_alive_matrix(*max_frequency=None*, *max_recency=None*)
Compute the probability alive matrix.

Parameters

- **max_frequency** (*float, optional*) – the maximum frequency to plot. Default is max observed frequency.
- **max_recency** (*float, optional*) – the maximum recency to plot. This also determines the age of the customer. Default to max observed age.

Returns *matrix* – A matrix of the form [t_x: historical recency, x: historical frequency]

conditional_probability_of_n_purchases_up_to_time(*n*, *t*, *frequency*, *recency*, *T*)
Return conditional probability of n purchases up to time t.

Calculate the probability of n purchases up to time t for an individual with history frequency, recency and T (age).

From paper: http://www.brucehardie.com/notes/028/pareto_nbd_conditional_pmf.pdf

Parameters

- **n** (*int*) – number of purchases.
- **t** (*a scalar*) – time up to which probability should be calculated.
- **frequency** (*float*) – historical frequency of customer.
- **recency** (*float*) – historical recency of customer.
- **T** (*float*) – age of the customer.

Returns *array_like*

expected_number_of_purchases_up_to_time(*t*)
Return expected number of repeat purchases up to time t.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population.

Parameters **t** (*array_like*) – times to calculate the expectation for.

Returns *array_like*

fit(*frequency*, *recency*, *T*, *weights=None*, *iterative_fitting=1*, *initial_params=None*, *verbose=False*, *tol=0.0001*, *index=None*, *fit_method='Nelder-Mead'*, *maxiter=2000*, ***kwargs*)
Pareto/NBD model fitter.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **recency** (*array_like*) – the recency vector of customers' purchases (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual log-likelihood, the log-likelihood is calculated for each pattern and multiplied by the number of customers with that pattern.

- **iterative_fitting** (*int, optional*) – perform iterative_fitting fits over random/warm-started initial params
- **initial_params** (*array_like, optional*) – set the initial parameters for the fitter.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **fit_method** (*string, optional*) – fit_method to passing to scipy.optimize.minimize
- **maxiter** (*int, optional*) – max iterations for optimizer in scipy.optimize.minimize will be overwritten if set in kwargs.
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *ParetoNBDFitter* – with additional properties like `params_` and methods like `predict`

Module contents

Base fitter for other classes.

class lifetimes.fitters.**BaseFitter**

Bases: `object`

Base class for fitters.

load_model(*path*)

Load model with dill package.

Parameters **path** (*str*) – From what path load model.

save_model(*path*, *save_data=True*, *save_generate_data_method=True*, *values_to_save=None*)

Save model with dill package.

Parameters

- **path** (*str*) – Path where to save model.
- **save_date** (*bool, optional*) – Whether to save data from fitter.data to pickle object
- **save_generate_data_method** (*bool, optional*) – Whether to save generate_new_data method (if it exists) from fitter.generate_new_data to pickle object.
- **values_to_save** (*list, optional*) – Placeholders for original attributes for saving object. If None will be extended to attr_list length like [None] * len(attr_list)

summary

Summary statistics describing the fit.

Returns **df** (*pd.DataFrame*) – Contains columns coef, se(coef), lower, upper

See also:

`print_summary`

5.4.2 Submodules

5.4.3 lifetimes.estimation module

5.4.4 lifetimes.generate_data module

lifetimes.generate_data.**beta_geometric_beta_binom_model**(*N*, *alpha*, *beta*, *gamma*, *delta*, *size=1*)

Generate artificial data according to the Beta-Geometric/Beta-Binomial Model.

You may wonder why we can have frequency = n_periods, when frequency excludes their first order. When a customer purchases something, they are born, _and in the next **period**_ we start asking questions about their alive-ness. So really they customer has bought frequency + 1, and been observed for n_periods + 1

Parameters

- **N** (*array_like*) – Number of transaction opportunities for new customers.
- **beta, gamma, delta** (*alpha,*) – Parameters in the model. See **[1]**_
- **size** (*int, optional*) – The number of customers to generate

Returns *DataFrame* – with index as customer_ids and the following columns: 'frequency', 'recency', 'n_periods', 'lambda', 'p', 'alive', 'customer_id'

References

lifetimes.generate_data.**beta_geometric_nbd_model**(*T*, *r*, *alpha*, *a*, *b*, *size=1*)

Generate artificial data according to the BG/NBD model.

See [1] for model details

Parameters

- **T** (*array_like*) – The length of time observing new customers.
- **alpha, a, b** (*r,*) – Parameters in the model. See **[1]**_
- **size** (*int, optional*) – The number of customers to generate

Returns *DataFrame* – With index as customer_ids and the following columns: 'frequency', 'recency', 'T', 'lambda', 'p', 'alive', 'customer_id'

References

lifetimes.generate_data.**beta_geometric_nbd_model_transactional_data**(*T*, *r*, *alpha*, *a*, *b*, *observation_period_end='2019-1-1'*, *freq='D'*, *size=1*)

Generate artificial transactional data according to the BG/NBD model.

See [1] for model details

Parameters

- **T** (*int, float or array_like*) – The length of time observing new customers.
- **alpha, a, b** (*r,*) – Parameters in the model. See **[1]**_

- **observation_period_end** (*date_like*) – The date observation ends
- **freq** (*string, optional*) – Default 'D' for days, 'W' for weeks, 'h' for hours
- **size** (*int, optional*) – The number of customers to generate

Returns *DataFrame* – The following columns: 'customer_id', 'date'

References

lifetimes.generate_data.**modified_beta_geometric_nbd_model**(*T*, *r*, *alpha*, *a*, *b*, *size=1*)

Generate artificial data according to the MBG/NBD model.

See[3],[4] for model details

Parameters

- **T** (*array_like*) – The length of time observing new customers.
- **alpha, a, b** (*r,*) – Parameters in the model. See [1]_
- **size** (*int, optional*) – The number of customers to generate

Returns *DataFrame* – with index as customer_ids and the following columns: 'frequency', 'recency', 'T', 'lambda', 'p', 'alive', 'customer_id'

References

lifetimes.generate_data.**pareto_nbd_model**(*T*, *r*, *alpha*, *s*, *beta*, *size=1*)

Generate artificial data according to the Pareto/NBD model.

See [2]_ for model details.

Parameters

- **T** (*array_like*) – The length of time observing new customers.
- **alpha, s, beta** (*r,*) – Parameters in the model. See [1]_
- **size** (*int, optional*) – The number of customers to generate

Returns *obj: DataFrame* – with index as customer_ids and the following columns: 'frequency', 'recency', 'T', 'lambda', 'mu', 'alive', 'customer_id'

References

5.4.5 lifetimes.plotting module

lifetimes.plotting.**plot_period_transactions**(*model*, *max_frequency=7*, *title='Frequency of Repeat Transactions'*, *xlabel='Number of Calibration Period Transactions'*, *ylabel='Customers'*, ***kwargs*)

Plot a figure with period actual and predicted transactions.

Parameters

[3] http://www.brucehardie.com/notes/025/ The Gamma-Gamma Model of Monetary Value.

[4] Peter S. Fader, Bruce G. S. Hardie, and Ka Lok Lee (2005), "RFM and CLV: Using iso-value curves for customer base analysis", Journal of Marketing Research, 42 (November), 415-430.

- **model** (*lifetimes model*) – A fitted lifetimes model.
- **max_frequency** (*int, optional*) – The maximum frequency to plot.
- **title** (*str, optional*) – Figure title
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel
- **kwargs** – Passed into the matplotlib.pyplot.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_calibration_purchases_vs_holdout_purchases**(*model*, *calibration_holdout_matrix*, *kind='frequency_cal'*, *n=7*, ***kwargs*)

Plot calibration purchases vs holdout.

This currently relies too much on the lifetimes.util calibration_and_holdout_data function.

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model.
- **calibration_holdout_matrix** (*pandas DataFrame*) – DataFrame from calibration_and_holdout_data function.
- **kind** (*str, optional*) –

  x-axis :"frequency_cal". Purchases in calibration period, "recency_cal". Age of customer at last purchase, "T_cal". Age of customer at the end of calibration period, "time_since_last_purchase". Time since user made last purchase
- **n** (*int, optional*) – Number of ticks on the x axis

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_frequency_recency_matrix**(*model*, *T=1*, *max_frequency=None*, *max_recency=None*, *title=None*, *xlabel="Customer's Historical Frequency"*, *ylabel="Customer's Recency"*, ***kwargs*)

Plot recency frequecy matrix as heatmap.

Plot a figure of expected transactions in T next units of time by a customer's frequency and recency.

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model.
- **T** (*fload, optional*) – Next units of time to make predictions for
- **max_frequency** (*int, optional*) – The maximum frequency to plot. Default is max observed frequency.
- **max_recency** (*int, optional*) – The maximum recency to plot. This also determines the age of the customer. Default to max observed age.
- **title** (*str, optional*) – Figure title
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel

- **kwargs** – Passed into the matplotlib.imshow command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_probability_alive_matrix**(*model*, *max_frequency=None*, *max_recency=None*, *title='Probability Customer is Alive, \nby Frequency and Recency of a Customer'*, *xlabel="Customer's Historical Frequency"*, *ylabel="Customer's Recency"*, ***kwargs*)

Plot probability alive matrix as heatmap.

Plot a figure of the probability a customer is alive based on their frequency and recency.

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model.
- **max_frequency** (*int, optional*) – The maximum frequency to plot. Default is max observed frequency.
- **max_recency** (*int, optional*) – The maximum recency to plot. This also determines the age of the customer. Default to max observed age.
- **title** (*str, optional*) – Figure title
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel
- **kwargs** – Passed into the matplotlib.imshow command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_expected_repeat_purchases**(*model*, *title='Expected Number of Repeat Purchases per Customer'*, *xlabel='Time Since First Purchase'*, *ax=None*, *label=None*, ***kwargs*)

Plot expected repeat purchases on calibration period .

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model.
- **max_frequency** (*int, optional*) – The maximum frequency to plot.
- **title** (*str, optional*) – Figure title
- **xlabel** (*str, optional*) – Figure xlabel
- **ax** (*matplotlib.AxesSubplot, optional*) – Using user axes
- **label** (*str, optional*) – Label for plot.
- **kwargs** – Passed into the matplotlib.pyplot.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_history_alive**(*model*, *t*, *transactions*, *datetime_col*, *freq='D'*, *start_date=None*, *ax=None*, ***kwargs*)

Draw a graph showing the probability of being alive for a customer in time.

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model.

- **t** (*int*) – the number of time units since the birth we want to draw the p_alive
- **transactions** (*pandas DataFrame*) – DataFrame containing the transactions history of the customer_id
- **datetime_col** (*str*) – The column in the transactions that denotes the datetime the purchase was made
- **freq** (*str, optional*) – Default 'D' for days. Other examples= 'W' for weekly
- **start_date** (*datetime, optional*) – Limit xaxis to start date
- **ax** (*matplotlib.AxesSubplot, optional*) – Using user axes
- **kwargs** – Passed into the matplotlib.pyplot.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_cumulative_transactions**(*model*, *transactions*, *datetime_col*, *customer_id_col*, *t*, *t_cal*, *datetime_format=None*, *freq='D'*, *set_index_date=False*, *title='Tracking Cumulative Transactions'*, *xlabel='day'*, *ylabel='Cumulative Transactions'*, *ax=None*, ***kwargs*)

Plot a figure of the predicted and actual cumulative transactions of users.

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model
- **transactions** (*pandas DataFrame*) – DataFrame containing the transactions history of the customer_id
- **datetime_col** (*str*) – The column in transactions that denotes the datetime the purchase was made.
- **customer_id_col** (*str*) – The column in transactions that denotes the customer_id
- **t** (*float*) – The number of time units since the begining of data for which we want to calculate cumulative transactions
- **t_cal** (*float*) – A marker used to indicate where the vertical line for plotting should be.
- **datetime_format** (*str, optional*) – A string that represents the timestamp format. Useful if Pandas can't understand the provided format.
- **freq** (*str, optional*) – Default 'D' for days, 'W' for weeks, 'M' for months... etc. Full list here: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#dateoffset-objects
- **set_index_date** (*bool, optional*) – When True set date as Pandas DataFrame index, default False - number of time units
- **title** (*str, optional*) – Figure title
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel
- **ax** (*matplotlib.AxesSubplot, optional*) – Using user axes
- **kwargs** – Passed into the pandas.DataFrame.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_incremental_transactions**(*model*, *transactions*, *datetime_col*, *customer_id_col*, *t*, *t_cal*, *datetime_format=None*, *freq='D'*, *set_index_date=False*, *title='Tracking Daily Transactions'*, *xlabel='day'*, *ylabel='Transactions'*, *ax=None*, ***kwargs*)

Plot a figure of the predicted and actual cumulative transactions of users.

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model
- **transactions** (*pandas DataFrame*) – DataFrame containing the transactions history of the customer_id
- **datetime_col** (*str*) – The column in transactions that denotes the datetime the purchase was made.
- **customer_id_col** (*str*) – The column in transactions that denotes the customer_id
- **t** (*float*) – The number of time units since the begining of data for which we want to calculate cumulative transactions
- **t_cal** (*float*) – A marker used to indicate where the vertical line for plotting should be.
- **datetime_format** (*str, optional*) – A string that represents the timestamp format. Useful if Pandas can't understand the provided format.
- **freq** (*str, optional*) – Default 'D' for days, 'W' for weeks, 'M' for months... etc. Full list here: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#dateoffset-objects
- **set_index_date** (*bool, optional*) – When True set date as Pandas DataFrame index, default False - number of time units
- **title** (*str, optional*) – Figure title
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel
- **ax** (*matplotlib.AxesSubplot, optional*) – Using user axes
- **kwargs** – Passed into the pandas.DataFrame.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_transaction_rate_heterogeneity**(*model*, *suptitle='Heterogeneity in Transaction Rate'*, *xlabel='Transaction Rate'*, *ylabel='Density'*, *suptitle_fontsize=14*, ***kwargs*)

Plot the estimated gamma distribution of lambda (customers' propensities to purchase).

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model, for now only for BG/NBD
- **suptitle** (*str, optional*) – Figure suptitle
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel

- **kwargs** – Passed into the matplotlib.pyplot.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

lifetimes.plotting.**plot_dropout_rate_heterogeneity**(*model*, *suptitle='Heterogeneity in Dropout Probability'*, *xlabel='Dropout Probability p'*, *ylabel='Density'*, *suptitle_fontsize=14*, ***kwargs*)

Plot the estimated gamma distribution of p.

p - (customers' probability of dropping out immediately after a transaction).

Parameters

- **model** (*lifetimes model*) – A fitted lifetimes model, for now only for BG/NBD
- **suptitle** (*str, optional*) – Figure suptitle
- **xlabel** (*str, optional*) – Figure xlabel
- **ylabel** (*str, optional*) – Figure ylabel
- **kwargs** – Passed into the matplotlib.pyplot.plot command.

Returns **axes** (*matplotlib.AxesSubplot*)

5.4.6 lifetimes.utils module

Lifetimes utils and helpers.

lifetimes.utils.**calibration_and_holdout_data**(*transactions*, *customer_id_col*, *datetime_col*, *calibration_period_end*, *observation_period_end=None*, *freq='D'*, *datetime_format=None*, *monetary_value_col=None*)

Create a summary of each customer over a calibration and holdout period.

This function creates a summary of each customer over a calibration and holdout period (training and testing, respectively). It accepts transaction data, and returns a DataFrame of sufficient statistics.

Parameters

- **transactions** – a Pandas DataFrame that contains the customer_id col and the datetime col.
- **customer_id_col** (*string*) – the column in transactions DataFrame that denotes the customer_id
- **datetime_col** (*string*) – the column in transactions that denotes the datetime the purchase was made.
- **calibration_period_end** – a period to limit the calibration to, inclusive.
- **observation_period_end** – a string or datetime to denote the final date of the study. Events after this date are truncated. If not given, defaults to the max 'datetime_col'.
- **freq** (*string, optional*) – Default 'D' for days. Other examples: 'W' for weekly.
- **datetime_format** (*string, optional*) – a string that represents the timestamp format. Useful if Pandas can't understand the provided format.

- **monetary_value_col** (*string, optional*) – the column in transactions that denotes the monetary value of the transaction. Optional, only needed for customer lifetime value estimation models.

Returns *obj: DataFrame* – A dataframe with columns frequency_cal, recency_cal, T_cal, frequency_holdout, duration_holdout If monetary_value_col isn't None, the dataframe will also have the columns monetary_value_cal and monetary_value_holdout.

lifetimes.utils.**summary_data_from_transaction_data**(*transactions*, *customer_id_col*, *datetime_col*, *monetary_value_col=None*, *datetime_format=None*, *observation_period_end=None*, *freq='D'*, *freq_multiplier=1*)

Return summary data from transactions.

This transforms a DataFrame of transaction data of the form: customer_id, datetime [, monetary_value]

to a DataFrame of the form: customer_id, frequency, recency, T [, monetary_value]

Parameters

- **transactions** – a Pandas DataFrame that contains the customer_id col and the datetime col.
- **customer_id_col** (*string*) – the column in transactions DataFrame that denotes the customer_id
- **datetime_col** (*string*) – the column in transactions that denotes the datetime the purchase was made.
- **monetary_value_col** (*string, optional*) – the columns in the transactions that denotes the monetary value of the transaction. Optional, only needed for customer lifetime value estimation models.
- **observation_period_end** (*datetime, optional*) – a string or datetime to denote the final date of the study. Events after this date are truncated. If not given, defaults to the max 'datetime_col'.
- **datetime_format** (*string, optional*) – a string that represents the timestamp format. Useful if Pandas can't understand the provided format.
- **freq** (*string, optional*) – Default 'D' for days, 'W' for weeks, 'M' for months... etc. Full list here: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#dateoffset-objects
- **freq_multiplier** (*int, optional*) – Default 1, could be use to get exact recency and T, i.e. with freq='W' row for user id_sample=1 will be recency=30 and T=39 while data in CDNOW summary are different. Exact values could be obtained with freq='D' and freq_multiplier=7 which will lead to recency=30.43 and T=38.86

Returns *obj: DataFrame:* – customer_id, frequency, recency, T [, monetary_value]

lifetimes.utils.**calculate_alive_path**(*model*, *transactions*, *datetime_col*, *t*, *freq='D'*)

Calculate alive path for plotting alive history of user.

Parameters

- **model** – A fitted lifetimes model
- **transactions** (*DataFrame*) – a Pandas DataFrame containing the transactions history of the customer_id

- **datetime_col** (*string*) – the column in the transactions that denotes the datetime the purchase was made
- **t** (*array_like*) – the number of time units since the birth for which we want to draw the p_alive
- **freq** (*string*) – Default 'D' for days. Other examples= 'W' for weekly

Returns *obj: Series* – A pandas Series containing the p_alive as a function of T (age of the customer)

lifetimes.utils.**expected_cumulative_transactions**(*model*, *transactions*, *datetime_col*, *customer_id_col*, *t*, *datetime_format=None*, *freq='D'*, *set_index_date=False*, *freq_multiplier=1*)

Get expected and actual repeated cumulative transactions.

Parameters

- **model** – A fitted lifetimes model
- **transactions** – a Pandas DataFrame containing the transactions history of the customer_id
- **datetime_col** (*string*) – the column in transactions that denotes the datetime the purchase was made.
- **customer_id_col** (*string*) – the column in transactions that denotes the customer_id
- **t** (*int*) – the number of time units since the begining of data for which we want to calculate cumulative transactions
- **datetime_format** (*string, optional*) – a string that represents the timestamp format. Useful if Pandas can't understand the provided format.
- **freq** (*string, optional*) – Default 'D' for days, 'W' for weeks, 'M' for months... etc. Full list here: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#dateoffset-objects
- **set_index_date** (*bool, optional*) – when True set date as Pandas DataFrame index, default False - number of time units
- **freq_multiplier** (*int, optional*) – Default 1, could be use to get exact cumulative transactions predicted by model, i.e. model trained with freq='W', passed freq to expected_cumulative_transactions is freq='D', and freq_multiplier=7.

Returns *obj: DataFrame* – A dataframe with columns actual, predicted

5.4.7 lifetimes.version module

5.4.8 Module contents

All fitters from fitters directory.

class lifetimes.**BetaGeoFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Also known as the BG/NBD model.

Based on [2]_, this model has the following assumptions:

1. Each individual, i, has a hidden lambda_i and p_i parameter

2. These come from a population wide Gamma and a Beta distribution respectively.
3. Individuals purchases follow a Poisson process with rate lambda_i*t .
4. After each purchase, an individual has a p_i probability of dieing (never buying again).

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: Series – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

References

conditional_expected_number_of_purchases_up_to_time (*t*, *frequency*, *recency*, *T*)
Conditional expected number of purchases up to time.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population, given they have purchase history (frequency, recency, T)

Parameters

- **t** (*array_like*) – times to calculate the expectation for.
- **frequency** (*array_like*) – historical frequency of customer.
- **recency** (*array_like*) – historical recency of customer.
- **T** (*array_like*) – age of the customer.

Returns *array_like*

conditional_probability_alive (*frequency*, *recency*, *T*)
Compute conditional probability alive.

Compute the probability that a customer with history (frequency, recency, T) is currently alive.

From http://www.brucehardie.com/notes/021/palive_for_BGNBD.pdf

Parameters

- **frequency** (*array or scalar*) – historical frequency of customer.
- **recency** (*array or scalar*) – historical recency of customer.
- **T** (*array or scalar*) – age of the customer.

Returns *array* – value representing a probability

conditional_probability_alive_matrix(*max_frequency=None*, *max_recency=None*)
Compute the probability alive matrix.

Parameters

- **max_frequency** (*float, optional*) – the maximum frequency to plot. Default is max observed frequency.
- **max_recency** (*float, optional*) – the maximum recency to plot. This also determines the age of the customer. Default to max observed age.

Returns *matrix* – A matrix of the form [t_x: historical recency, x: historical frequency]

expected_number_of_purchases_up_to_time(*t*)
Calculate the expected number of repeat purchases up to time t.

Calculate repeat purchases for a randomly choose individual from the population.

Parameters **t** (*array_like*) – times to calculate the expection for

Returns *array_like*

fit(*frequency*, *recency*, *T*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, ***kwargs*)
Fit a dataset to the BG/NBD model.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **recency** (*array_like*) – the recency vector of customers' purchases (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual loglikelihood, the loglikelihood is calculated for each pattern and multiplied by the number of customers with that pattern.
- **initial_params** (*array_like, optional*) – set the initial parameters for the fitter.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *BetaGeoFitter* – with additional properties like `params_` and methods like `predict`

probability_of_n_purchases_up_to_time(*t*, *n*)
Compute the probability of n purchases.

$$P(N(t) = n|\text{model})$$

where N(t) is the number of repeat purchases a customer makes in t units of time.

Parameters

- **t** (*float*) – number units of time
- **n** (*int*) – number of purchases

Returns *float* – Probability to have n purchases up to t units of time

class lifetimes.**ParetoNBDFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Pareto NBD fitter[7].

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: OrderedDict – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the columns given in the call to *fit*

References

conditional_expected_number_of_purchases_up_to_time(*t*, *frequency*, *recency*, *T*)
Conditional expected number of purchases up to time.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population, given they have purchase history (frequency, recency, T)

Parameters

- **t** (*array_like*) – times to calculate the expectation for.
- **frequency** (*array_like*) – historical frequency of customer.
- **recency** (*array_like*) – historical recency of customer.
- **T** (*array_like*) – age of the customer.

Returns *array_like*

conditional_probability_alive(*frequency*, *recency*, *T*)
Conditional probability alive.

Compute the probability that a customer with history (frequency, recency, T) is currently alive. From paper: http://brucehardie.com/notes/009/pareto_nbd_derivations_2005-11-05.pdf

Parameters

- **frequency** (*float*) – historical frequency of customer.

[7] **David C. Schmittlein, Donald G. Morrison and Richard Colombo** Management Science,Vol. 33, No. 1 (Jan., 1987), pp. 1-24 "Counting Your Customers: Who Are They and What Will They Do Next,"

- **recency** (*float*) – historical recency of customer.
- **T** (*float*) – age of the customer.

Returns *float* – value representing a probability

conditional_probability_alive_matrix(*max_frequency=None*, *max_recency=None*)
Compute the probability alive matrix.

Parameters

- **max_frequency** (*float, optional*) – the maximum frequency to plot. Default is max observed frequency.
- **max_recency** (*float, optional*) – the maximum recency to plot. This also determines the age of the customer. Default to max observed age.

Returns *matrix* – A matrix of the form [t_x: historical recency, x: historical frequency]

conditional_probability_of_n_purchases_up_to_time(*n*, *t*, *frequency*, *recency*, *T*)
Return conditional probability of n purchases up to time t.

Calculate the probability of n purchases up to time t for an individual with history frequency, recency and T (age).

From paper: http://www.brucehardie.com/notes/028/pareto_nbd_conditional_pmf.pdf

Parameters

- **n** (*int*) – number of purchases.
- **t** (*a scalar*) – time up to which probability should be calculated.
- **frequency** (*float*) – historical frequency of customer.
- **recency** (*float*) – historical recency of customer.
- **T** (*float*) – age of the customer.

Returns *array_like*

expected_number_of_purchases_up_to_time(*t*)
Return expected number of repeat purchases up to time t.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population.

Parameters **t** (*array_like*) – times to calculate the expectation for.

Returns *array_like*

fit(*frequency*, *recency*, *T*, *weights=None*, *iterative_fitting=1*, *initial_params=None*, *verbose=False*, *tol=0.0001*, *index=None*, *fit_method='Nelder-Mead'*, *maxiter=2000*, ***kwargs*)
Pareto/NBD model fitter.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **recency** (*array_like*) – the recency vector of customers' purchases (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter

represents the count of customers with a given purchase pattern. Instead of calculating individual log-likelihood, the log-likelihood is calculated for each pattern and multiplied by the number of customers with that pattern.

- **iterative_fitting** (*int, optional*) – perform iterative_fitting fits over random/warm-started initial params
- **initial_params** (*array_like, optional*) – set the initial parameters for the fitter.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **fit_method** (*string, optional*) – fit_method to passing to scipy.optimize.minimize
- **maxiter** (*int, optional*) – max iterations for optimizer in scipy.optimize.minimize will be overwritten if set in kwargs.
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *ParetoNBDFitter* – with additional properties like `params_` and methods like `predict`

class lifetimes.**GammaGammaFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Fitter for the gamma-gamma model.

It is used to estimate the average monetary value of customer transactions.

This implementation is based on the Excel spreadsheet found in[3]. More details on the derivation and evaluation can be found in[4].

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: OrderedDict – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the columns given in the call to *fit*

References

penalizer_coef
float – The coefficient applied to an l2 norm on the parameters

params_
:obj: Series – The fitted parameters of the model

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

conditional_expected_average_profit (*frequency=None*, *monetary_value=None*)
Conditional expectation of the average profit.

This method computes the conditional expectation of the average profit per transaction for a group of one or more customers.

Parameters

- **frequency** (*array_like, optional*) – a vector containing the customers' frequencies. Defaults to the whole set of frequencies used for fitting the model.
- **monetary_value** (*array_like, optional*) – a vector containing the customers' monetary values. Defaults to the whole set of monetary values used for fitting the model.

Returns *array_like* – The conditional expectation of the average profit per transaction

customer_lifetime_value (*transaction_prediction_model*, *frequency*, *recency*, *T*, *monetary_value*, *time=12*, *discount_rate=0.01*, *freq='D'*)
Return customer lifetime value.

This method computes the average lifetime value for a group of one or more customers.

Parameters

- **transaction_prediction_model** (*model*) – the model to predict future transactions, literature uses pareto/ndb models but we can also use a different model like beta-geo models
- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **recency** (*the recency vector of customers' purchases*) – (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **monetary_value** (*array_like*) – the monetary value vector of customer's purchases (denoted m in literature).
- **time** (*float, optional*) – the lifetime expected for the user in months. Default: 12
- **discount_rate** (*float, optional*) – the monthly adjusted discount rate. Default: 0.01
- **freq** (*string, optional*) – {"D", "H", "M", "W"} for day, hour, month, week. This represents what unit of time your T is measure in.

Returns *Series* – Series object with customer ids as index and the estimated customer lifetime values as values

fit (*frequency*, *monetary_value*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, *q_constraint=False*, ***kwargs*)
Fit the data to the Gamma/Gamma model.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **monetary_value** (*array_like*) – the monetary value vector of customer's purchases (denoted m in literature).
- **weights** (*None or array_like*) – Number of customers with given frequency/monetary_value, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/monetary_value. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual loglikelihood, the loglikelihood is calculated for each pattern and multiplied by the number of customers with that pattern.
- **initial_params** (*array_like, optional*) – set the initial parameters for the fitter.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **q_constraint** (*bool, optional*) – when q < 1, population mean will result in a negative value leading to negative CLV outputs. If True, we penalize negative values of q to avoid this issue.
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *GammaGammaFitter* – fitted and with parameters estimated

class lifetimes.**ModifiedBetaGeoFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.beta_geo_fitter.BetaGeoFitter*

Also known as the MBG/NBD model.

Based on[5],[6], this model has the following assumptions: 1) Each individual, `i`, has a hidden `lambda_i` and `p_i` parameter 2) These come from a population wide Gamma and a Beta distribution

respectively.

3. Individuals purchases follow a Poisson process with rate $\lambda_i * t$.
4. At the beginning of their lifetime and after each purchase, an individual has a p_i probability of dieing (never buying again).

References

penalizer_coef

float – The coefficient applied to an l2 norm on the parameters

params_

:obj: Series – The fitted parameters of the model

[5] Batislam, E.P., M. Denizel, A. Filiztekin (2007), "Empirical validation and comparison of models for customer base analysis," International Journal of Research in Marketing, 24 (3), 201-209.

[6] Wagner, U. and Hoppe D. (2008), "Erratum on the MBG/NBD Model," International Journal of Research in Marketing, 25 (3), 225-226.

data
:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

conditional_expected_number_of_purchases_up_to_time(*t*, *frequency*, *recency*, *T*)
Conditional expected number of repeat purchases up to time t.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population, given they have purchase history (frequency, recency, T) See Wagner, U. and Hoppe D. (2008).

Parameters

- **t** (*array_like*) – times to calculate the expectation for.
- **frequency** (*array_like*) – historical frequency of customer.
- **recency** (*array_like*) – historical recency of customer.
- **T** (*array_like*) – age of the customer.

Returns *array_like*

conditional_probability_alive(*frequency*, *recency*, *T*)
Conditional probability alive.

Compute the probability that a customer with history (frequency, recency, T) is currently alive. From https://www.researchgate.net/publication/247219660_Empirical_validation_and_comparison_of_models_for_customer_base_analysis Appendix A, eq. (5)

Parameters

- **frequency** (*array or float*) – historical frequency of customer.
- **recency** (*array or float*) – historical recency of customer.
- **T** (*array or float*) – age of the customer.

Returns *array* – value representing probability of being alive

expected_number_of_purchases_up_to_time(*t*)
Return expected number of repeat purchases up to time t.

Calculate the expected number of repeat purchases up to time t for a randomly choose individual from the population.

Parameters **t** (*array_like*) – times to calculate the expectation for

Returns *array_like*

fit(*frequency*, *recency*, *T*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, ***kwargs*)
Fit the data to the MBG/NBD model.

Parameters

- **frequency** (*array_like*) – the frequency vector of customers' purchases (denoted x in literature).
- **recency** (*array_like*) – the recency vector of customers' purchases (denoted t_x in literature).
- **T** (*array_like*) – customers' age (time units since first purchase)
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual log-likelihood, the log-likelihood is calculated for each pattern and multiplied by the number of customers with that pattern.
- **verbose** (*bool, optional*) – set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – index for resulted DataFrame which is accessible via self.data
- **kwargs** – key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *ModifiedBetaGeoFitter* – With additional properties and methods like `params_` and `predict`

probability_of_n_purchases_up_to_time(*t*, *n*)

Compute the probability of n purchases up to time t.

$$P(N(t) = n|\text{model})$$

where N(t) is the number of repeat purchases a customer makes in t units of time.

Parameters

- **t** (*float*) – number units of time
- **n** (*int*) – number of purchases

Returns *float* – Probability to have n purchases up to t units of time

class lifetimes.**BetaGeoBetaBinomFitter**(*penalizer_coef=0.0*)

Bases: *lifetimes.fitters.BaseFitter*

Also known as the Beta-Geometric/Beta-Binomial Model [1]_.

Future purchases opportunities are treated as discrete points in time. In the literature, the model provides a better fit than the Pareto/NBD model for a nonprofit organization with regular giving patterns.

The model is estimated with a recency-frequency matrix with n transaction opportunities.

Parameters **penalizer_coef** (*float*) – The coefficient applied to an l2 norm on the parameters

penalizer_coef

float – The coefficient applied to an l2 norm on the parameters

params_

:obj: Series – The fitted parameters of the model

data

:obj: DataFrame – A DataFrame with the values given in the call to *fit*

variance_matrix_
:obj: DataFrame – A DataFrame with the variance matrix of the parameters.

confidence_intervals_
:obj: DataFrame – A DataFrame 95% confidence intervals of the parameters

standard_errors_
:obj: Series – A Series with the standard errors of the parameters

summary
:obj: DataFrame – A DataFrame containing information about the fitted parameters

References

conditional_expected_number_of_purchases_up_to_time(*m_periods_in_future*, *frequency*, *recency*, *n_periods*)
Conditional expected purchases in future time period.

The expected number of future transactions across the next m_periods_in_future transaction opportunities by a customer with purchase history (x, tx, n).

$$E(X(n_{periods}, n_{periods} + m_{periods_i n_f uture})|\alpha, \beta, \gamma, \delta, frequency, recency, n_{periods})$$

See (13) in Fader & Hardie 2010.

Parameters **t** (*array_like*) – time n_periods (n+t)

Returns *array_like* – predicted transactions

conditional_probability_alive(*m_periods_in_future*, *frequency*, *recency*, *n_periods*)
Conditional probability alive.

Conditional probability customer is alive at transaction opportunity n_periods + m_periods_in_future.

$$P(aliveatn_periods + m_periods_i n_f uture|alpha, beta, gamma, delta, frequency, recency, n_periods)$$

See (A10) in Fader and Hardie 2010.

Parameters **m** (*array_like*) – transaction opportunities

Returns *array_like* – alive probabilities

expected_number_of_transactions_in_first_n_periods(*n*)
Return expected number of transactions in first n n_periods.

Expected number of transactions occurring across first n transaction opportunities. Used by Fader and Hardie to assess in-sample fit.

$$Pr(X(n) = x|\alpha, \beta, \gamma, \delta)$$

See (7) in Fader & Hardie 2010.

Parameters **n** (*float*) – number of transaction opportunities

Returns *DataFrame* – Predicted values, indexed by x

fit(*frequency*, *recency*, *n_periods*, *weights=None*, *initial_params=None*, *verbose=False*, *tol=1e-07*, *index=None*, ***kwargs*)
Fit the BG/BB model.

Parameters

- **frequency** (*array_like*) – Total periods with observed transactions
- **recency** (*array_like*) – Period of most recent transaction
- **n_periods** (*array_like*) – Number of transaction opportunities. Previously called *n*.
- **weights** (*None or array_like*) – Number of customers with given frequency/recency/T, defaults to 1 if not specified. Fader and Hardie condense the individual RFM matrix into all observed combinations of frequency/recency/T. This parameter represents the count of customers with a given purchase pattern. Instead of calculating individual log-likelihood, the log-likelihood is calculated for each pattern and multiplied by the number of customers with that pattern. Previously called *n_custs*.
- **verbose** (*boolean, optional*) – Set to true to print out convergence diagnostics.
- **tol** (*float, optional*) – Tolerance for termination of the function minimization process.
- **index** (*array_like, optional*) – Index for resulted DataFrame which is accessible via self.data
- **kwargs** – Key word arguments to pass to the scipy.optimize.minimize function as options dict

Returns *BetaGeoBetaBinomFitter* – fitted and with parameters estimated

5.5 Changelog

5.5.1 0.11.1

- bump the Pandas requirements to >= 0.24.0. This should have been done in 0.11.0
- suppress some warnings from autograd.

5.5.2 0.11.0

- Move most models (all but Pareto) to autograd for automatic differentiation of their likelihood. This results in faster (at least 3x) and more successful convergence, plus allows for some really exciting extensions (coming soon).
- `GammaGammaFitter`, `BetaGeoFitter`, `ModifiedBetaGeoFitter` and `BetaGeoBetaBinomFitter` have three new attributes: `confidence_interval_`, `variance_matrix_` and `standard_errors_`
- `params_` on fitted models is not longer an OrderedDict, but a Pandas Series
- `GammaGammaFitter` can accept a `weights` argument now.
- `customer_lifelime_value` in `GammaGamma` now accepts a frequency argument.
- fixed a bug that was causing `ParetoNBDFitter` to generate data incorrectly.

5.5.3 0.10.1

- performance improvements to `generate_data.py` for large datasets #195
- performance improvements to `summary_data_from_transaction_data`, thanks @MichaelSchreier

- Previously, `GammaGammaFitter` would have an infinite mean when its `q` parameter was less than 1. This was possible for some datasets. In 0.10.1, a new argument is added to `GammaGammaFitter` to constrain that `q` is greater than 1. This can be done with `q_constraint=True` in the call to `GammaGammaFitter.fit`. See issue #146. Thanks @vruvora
- Stop support of scipy < 1.0.
- Stop support of < Python 3.5.

5.5.4 0.10.0

- `BetaGeoBetaBinomFitter.fit` has replaced `n_custs` with the more appropriately named `weights` (to align with other statisical libraries). By default and if unspecified, `weights` is equal to an array of 1s.
- The `conditional_` methods on `BetaGeoBetaBinomFitter` have been updated to handle exogenously provided recency, frequency and periods.
- Performance improvements in `BetaGeoBetaBinomFitter`. `fit` takes about 50% less time than previously.
- `BetaGeoFitter`, `ParetoNBDFitter`, and `ModifiedBetaGeoFitter` both have a new `weights` argument in their `fit`. This can be used to reduce the size of the data (collapsing subjects with the same recency, frequency, T).

5.5.5 0.9.1

- Added a data generation method, `generate_new_data` to `BetaGeoBetaBinomFitter`. @zscore
- Fixed a bug in `summary_data_from_transaction_data` that was casting values to `int` prematurely. This was solved by including a new param `freq_multiplier` to be used to scale the resulting durations. See #100 for the original issue. @aprotopopov
- Performance and bug fixes in `utils.expected_cumulative_transactions`. @aprotopopov
- Fixed a bug in `utils.calculate_alive_path` that was causing a difference in values compared to `summary_from_transaction_data`. @DaniGate

5.5.6 0.9.0

- fixed many of the numpy warnings as the result of fitting
- added optional `initial_params` to all models
- Added `conditional_probability_of_n_purchases_up_to_time` to `ParetoNBDFitter`
- Fixed a bug in `expected_cumulative_transactions` and `plot_cumulative_transactions`

5.5.7 0.8.1

- adding new `save_model` and `load_model` functions to all fitters. This will save the model locally as a pickle file.
- `observation_period_end` in `summary_data_from_transaction_data` and `calibration_and_holdout_data` now defaults to the max date in the dataset, instead of current time.
- improved stability of estimators.

- improve Runtime warnings.
- All fitters are now in a local file. This doesn't change the API however.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

D

E

F

G

L

M

P

S

V