**BG-NBD Model for Customer Base Analysis**

**Introduction**

<http://benalexkeen.com/bg-nbd-model-for-customer-base-analysis-in-python/>

In this post we will use python to replicate the BG-NBD (Beta Geometric Negative Binomial Distribution) model that is described in the paper [“Counting Your Customers” the Easy Way: An Alternative to the Pareto/NBD Model](http://brucehardie.com/papers/018/fader_et_al_mksc_05.pdf) by Fader et al. in 2005.

We will use the Excel worksheet that was explored in the original research paper, which can be found [here](http://brucehardie.com/notes/004/).

The model can be used to determine the expected repeat visits for customers in order to determine a customers lifetime value. It can also be used to determine whether a customer has churned or is likely to churn soon.

The model is already implemented in python in the [lifetimes](http://lifetimes.readthedocs.io/en/latest/) package from Cameron Davidson Pilon at Shopify but this post aims to break this open and explore the steps we go through when using this model.

**Assumptions**

This model has a number of assumptions associated with it. These are as follows:

*1. While active, transactions made by a customer in time period*t*is Poisson distributed with mean*λt

Let’s look at the probability distribution of one customer with  λ=5:

import matplotlib.pyplot as plt

from scipy.stats import poisson

probability\_arr = []

distribution = poisson(5)

for transactions in range(0,20):

probability\_arr.append(distribution.pmf(transactions))

plt.figure(figsize=(8,5))

plt.ylabel('Probability')

plt.xlabel('Number of Transactions')

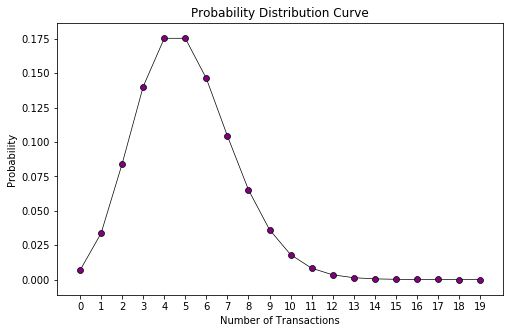
plt.xticks(range(0, 20))

plt.title('Probability Distribution Curve')

plt.plot(probability\_arr, color='black', linewidth=0.7, zorder=1)

plt.scatter(range(0, 20), probability\_arr, color='purple', edgecolor='black', linewidth=0.7, zorder=2)

plt.show()



*2. Differences in transaction rate between customers follows a gamma distribution with shape*r*and scale*α

Let’s take a look at how the probability distributions would look for 100 customers with r=9.0 and α=0.5:

import numpy as np

plt.figure(figsize=(8,5))

for customer in range(0, 100):

distribution = poisson(np.random.gamma(shape=9, scale=0.5))

probability\_arr = []

for transactions in range(0,20):

probability\_arr.append(distribution.pmf(transactions))

plt.plot(probability\_arr, color='black', linewidth=0.7, zorder=1)

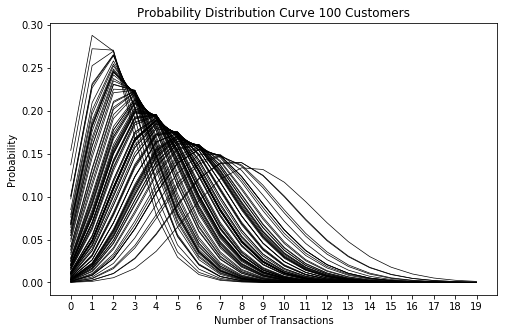
plt.ylabel('Probability')

plt.xlabel('Number of Transactions')

plt.xticks(range(0, 20))

plt.title('Probability Distribution Curve 100 Customers')

plt.show()



*3. Each customer becomes inactive after each transaction with probability*p

*4. Differences in*p*follows a beta distribution with shape parameters*a*and*b

Let’s apply a random drop-off with probability p that is beta distributed with parameters a=1.0a=1.0 and b=2.5 to each of our 100 customers after each transaction and see what this does to our probability distribution curves:

import numpy as np

plt.figure(figsize=(8,5))

for customer in range(0, 100):

distribution = poisson(np.random.gamma(shape=9, scale=0.5))

probability\_arr = []

beta = np.random.beta(a=1.0, b=2.5)

cumulative\_beta = 0

for transactions in range(0,20):

proba = distribution.pmf(transactions)

cumulative\_beta = beta + cumulative\_beta - (beta \* cumulative\_beta)

inactive\_probability = 1 - cumulative\_beta

proba \*= inactive\_probability

probability\_arr.append(proba)

probability\_arr = np.array(probability\_arr)

probability\_arr /= probability\_arr.sum()

plt.plot(probability\_arr, color='black', linewidth=0.7, zorder=1)

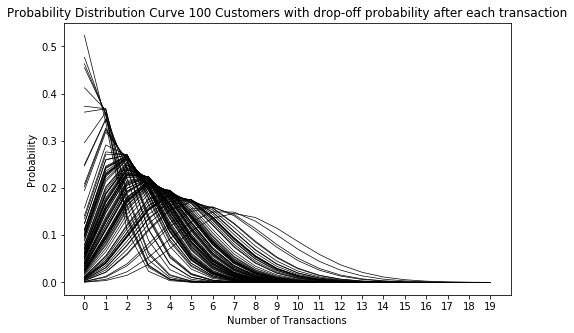
plt.ylabel('Probability')

plt.xlabel('Number of Transactions')

plt.xticks(range(0, 20))

plt.title('Probability Distribution Curve 100 Customers with drop-off probability after each transaction')

plt.show()



We can see that this transformation moves the distribution to the left. This makes sense as there is an increased likelihood of fewer transactions because after each transaction there is the probability that the customer doesn’t return and this accumulates with each additional transaction.

*5. Transaction rate and dropout probability vary independently between customers*

**Loading Data**

Let’s start by loading in the raw data from the Excel file into a pandas DataFrame.

import pandas as pd

df = pd.read\_excel('bgnbd.xls', sheet\_name='Raw Data').set\_index('ID')

df.head()

|  | **x** | **t\_x** | **T** |
| --- | --- | --- | --- |
| **ID** |  |  |  |
| **1** | **2** | **30.428571** | **38.857143** |
| **2** | **1** | **1.714286** | **38.857143** |
| **3** | **0** | **0.000000** | **38.857143** |
| **4** | **0** | **0.000000** | **38.857143** |
| **5** | **0** | **0.000000** | **38.857143** |

In this dataset:

x is the number of repeat purchases from this customer since the first purchase (frequency)  
tx is the date of the most recent purchase in weeks since the customer’s first purchase (recency)  
T is the time to be considered in weeks since the customer’s first purchase

**Optimising likelihood function parameters**

Given the above assumptions, the likelihood a customer makes xx purchases in a given time period T is:

L(λ,p|X=x,T)=(1−p)xλxe−λT+δx>0p(1−p)x−1λxe−λtxL(λ,p|X=x,T)=(1−p)xλxe−λT+δx>0p(1−p)x−1λxe−λtx

However, this depends on knowing the variables λλ and pp, which are both unobserved quantities.

We can, instead, write the likelihood function of a randomly-chosen individual with purchase history X=x,tx,TX=x,tx,T as:

L(r,α,a,b|X=x,tx,T)=A1A2(A3+δx>0A4)L(r,α,a,b|X=x,tx,T)=A1A2(A3+δx>0A4)

Where:

A1=Γ(r+x)αrΓ(r)A1=Γ(r+x)αrΓ(r)

A2=Γ(a+b)Γ(b+x)Γ(b)+Γ(a+b+x)A2=Γ(a+b)Γ(b+x)Γ(b)+Γ(a+b+x)

A3=1α+Tr+xA3=1α+Tr+x

A4=(ab+x−1)(1α+tx)r+xA4=(ab+x−1)(1α+tx)r+x

We can then optimise the paramaters of the likelihood function by optimising the log-likelihood function, such that the log-likelihood function is:

ln[L(r,α,a,b|X=x,tx,T)=ln(A1)ln(A2)ln(eln(A3)+δx>0eln(A4))]ln⁡[L(r,α,a,b|X=x,tx,T)=ln⁡(A1)ln⁡(A2)ln⁡(eln⁡(A3)+δx>0eln⁡(A4))]

Where:

ln(A1)=ln[Γ(r+x)]–ln[Γ(r)]+rln(α)ln⁡(A1)=ln⁡[Γ(r+x)]–ln⁡[Γ(r)]+rln⁡(α)

ln(A2)=ln[Γ(a+b)]+ln[Γ(b+x)]–ln[Γ(b)]–ln[Γ(a+b+x)]ln⁡(A2)=ln⁡[Γ(a+b)]+ln⁡[Γ(b+x)]–ln⁡[Γ(b)]–ln⁡[Γ(a+b+x)]

ln(A3)=−(r+x)ln(α+T)ln⁡(A3)=−(r+x)ln⁡(α+T)

ln(A4)={ln(a)–ln(b+x−1)–(r+x)ln(α+tx)0if x>0otherwiseln⁡(A4)={ln⁡(a)–ln⁡(b+x−1)–(r+x)ln⁡(α+tx)if x>00otherwise

Let’s define this as a negative log-likelihood in python:

from scipy.special import gammaln

def negative\_log\_likelihood(params, x, t\_x, T):

if np.any(np.asarray(params) <= 0):

return np.inf

r, alpha, a, b = params

ln\_A\_1 = gammaln(r + x) - gammaln(r) + r \* np.log(alpha)

ln\_A\_2 = (gammaln(a + b) + gammaln(b + x) - gammaln(b) -

gammaln(a + b + x))

ln\_A\_3 = -(r + x) \* np.log(alpha + T)

ln\_A\_4 = x.copy()

ln\_A\_4[ln\_A\_4 > 0] = (

np.log(a) -

np.log(b + ln\_A\_4[ln\_A\_4 > 0] - 1) -

(r + ln\_A\_4[ln\_A\_4 > 0]) \* np.log(alpha + t\_x)

)

delta = np.where(x>0, 1, 0)

log\_likelihood = ln\_A\_1 + ln\_A\_2 + np.log(np.exp(ln\_A\_3) + delta \* np.exp(ln\_A\_4))

**return** -log\_likelihood.sum()

Now we can optimise our parameters. We’ll use the Nelder-Mead Simplex algorithm, which is a heuristic, non-gradient search method to minimise our negative log likelihood cost function.

from scipy.optimize import minimize

scale = 1 / df['T'].max()

scaled\_recency = df['t\_x'] \* scale

scaled\_T = df['T'] \* scale

def \_func\_caller(params, func\_args, function):

return function(params, \*func\_args)

current\_init\_params = np.array([1.0, 1.0, 1.0, 1.0])

output = minimize(

\_func\_caller,

method="Nelder-Mead",

tol=0.0001,

x0=current\_init\_params,

args=([df['x'], scaled\_recency, scaled\_T], negative\_log\_likelihood),

options={'maxiter': 2000}

)

r = output.x[0]

alpha = output.x[1]

a = output.x[2]

b = output.x[3]

alpha /= scale

**print**("r = {}".format(r))

**print**("alpha = {}".format(alpha))

**print**("a = {}".format(a))

**print**("b = {}".format(b))

r = 0.242594123569

alpha = 4.41358813135

a = 0.792935471652

b = 2.42595536972

**Expected Sales Forecasting**

So now that we have our optimised parameters rr, αα, aa and bb, we can use these to compute our repeat sales forecast for our cohort of customers across time period of length tt using the following formula for any given individual:

E(X(t)|r,α,a,b)=a+b−1a−1[1–(αα+t)r2F1(r,b;a+b−1;tα+t)]E(X(t)|r,α,a,b)=a+b−1a−1[1–(αα+t)r2F1(r,b;a+b−1;tα+t)]

Where 2F12F1 is the Gaussian hypergeometric function, which takes the form:

2F1(a,b;c;z)=∑j=0∞uj2F1(a,b;c;z)=∑j=0∞uj

where uj=(a)j(b)j(c)j)zjj!where uj=(a)j(b)j(c)j)zjj!

This can be expanded to the series:

ujuj−1=(a+j−1)(b+j−1)(c+j−1)jzujuj−1=(a+j−1)(b+j−1)(c+j−1)jz

And we can substitute the values above for these aa, bb, cc and zz values of 2F12F1

from scipy.special import hyp2f1

def expected\_sales\_to\_time\_t(t):

hyp2f1\_a = r

hyp2f1\_b = b

hyp2f1\_c = a + b - 1

hyp2f1\_z = t / (alpha + t)

hyp\_term = hyp2f1(hyp2f1\_a, hyp2f1\_b, hyp2f1\_c, hyp2f1\_z)

return ((a + b - 1) / (a - 1)) \* (1-(((alpha / (alpha+t)) \*\* r) \* hyp\_term))

So now we can test out our expectes sales for any given individual in our cohort, let’s say we want to see how many purchases we can expect from an individual across a period of one year (52 weeks):

expected\_sales\_to\_time\_t(52)

1.444010643699092

So we would expect 1.44 sales for any given individual across the next year.

Now will calculate the cumulative repeat sales for 78 weeks from our cohort of customers.

To calculate cumulative repeat sales for a forecast across our cohort, we can’t just scale up to number of customers, we need to take into account the fact that each customer had a different time of first purchase.

nsns is the number of people that had their first purchase on day ss

If SS is the total possible number of start dates:

Total repeat transactions to t=∑s=0Sδ(t>s)nsE[X(t−s)]Total repeat transactions to t=∑s=0Sδ(t>s)nsE[X(t−s)]

where:

δ(t>s)={10if t>sotherwiseδ(t>s)={1if t>s0otherwise

# Period of consideration is 39 weeks.

# T indicates the length of time since first purchase

n\_s = (39 - df['T']).value\_counts().sort\_index()

n\_s.head()

0.142857 18

0.285714 22

0.428571 17

0.571429 20

0.714286 23

Name: T, dtype: int64

18 people made their first transaction on day 1 (1/7 weeks), 22 on day 2 (2/7 weeks), 17 on day 3 (1/7 weeks) etc.

forecast\_range = np.arange(0, 78, 1/7.0)

**def** cumulative\_repeat\_transactions\_to\_t(t):

expected\_transactions\_per\_customer = (t - n\_s.index).map(**lambda** x: expected\_sales\_to\_time\_t(x) **if** x > 0 **else** 0)

expected\_transactions\_all\_customers = (expected\_transactions\_per\_customer \* n\_s).values

**return** expected\_transactions\_all\_customers.sum()

cum\_rpt\_sales = pd.Series(map(cumulative\_repeat\_transactions\_to\_t, forecast\_range), index=forecast\_range)

cum\_rpt\_sales.tail(10)

76.571429 4109.744742

76.714286 4114.856053

76.857143 4119.961614

77.000000 4125.061441

77.142857 4130.155549

77.285714 4135.243956

77. 77.428571 4140.326675

571429 4145.403724

77.714286 4150.475118

77.857143 4155.540873

dtype: float64

So across the next 78 weeks we would expect to make around 4156 repeat transactions from our cohort of customers

**Conditional Expectation**

If we want to calculate the expected sales from a single customer (how many transacations any single customer will make going forward in time period tt), we can calculate the conditional expectation for that customer.

The expression used to calculate this is:

E(Y(t)|X=x,tx,T,r,α,a,b)=a+b+x–1a−1×[1–(α+Tα+T+t)r+x2F1(r+x,b+x;a+b+x−1;tα+T+t)]1+δ(x>0)ab+x−1(α+Tα+tx)r+xE(Y(t)|X=x,tx,T,r,α,a,b)=a+b+x–1a−1×[1–(α+Tα+T+t)r+x2F1(r+x,b+x;a+b+x−1;tα+T+t)]1+δ(x>0)ab+x−1(α+Tα+tx)r+x

In [11]:

**def** calculate\_conditional\_expectation(t, x, t\_x, T):

first\_term = (a + b + x - 1) / (a-1)

hyp2f1\_a = r + x

hyp2f1\_b = b + x

hyp2f1\_c = a + b + x - 1

hyp2f1\_z = t / (alpha + T + t)

hyp\_term = hyp2f1(hyp2f1\_a, hyp2f1\_b, hyp2f1\_c, hyp2f1\_z)

second\_term = (1 - ((alpha + T) / (alpha + T + t)) \*\* (r + x) \* hyp\_term)

delta = 1 **if** x > 0 **else** 0

denominator = 1 + delta \* (a / (b + x - 1)) \* ((alpha + T) / (alpha + t\_x)) \*\* (r+x)

**return** first\_term \* second\_term / denominator

calculate\_conditional\_expectation(39, 2, 30.43, 38.86)

1.225904664486748

So we would expect a customer with:

x=2x=2  
tx=30.43tx=30.43  
T=38.86T=38.86

to make 1.22 purchases over the next 39 weeks, given that they’re drawn from the same cohort as the others used to calculate our parameters.