Published in Towards Data Science

Kaushik Sureshkumar Follow
Jan 4, 2021 · 11 min read · · Listen

Bayesian AB Testing — Part I — Conversions

Photo by Raymond Pang on Unsplash

Series Structure

In a previous blog post, I discussed the advantages of using Bayesian AB testing methods rather than frequentist ones. In this series of blog posts, I will be taking a deeper dive into the calculations involved and how to implement them in real world scenarios. The structure of the series will be as follows:

1. Modelling and analysis of conversion based test metrics (rate metrics)
2. Modelling and analysis of revenue based test metrics
3. Calculating test duration
4. Choosing an appropriate prior
5. Running tests with multiple variants

So without further ado, let's jump into how to model, use and analyse conversion based test metrics in bayesian product experiments.

Experiment Context

Following on from the example used in the previous post, let's assume we've recently changed the messaging on an upsell screen and want to AB test it before releasing to our wider user base. We hypothesise that the changes we've made will result in a significantly better conversion rate.

Similar to frequentist methods, we can model each conversion on the upsell screen, X, as Bernoulli random variable with conversion probability λ:

$$
\begin{aligned}
&X \sim Ber(\lambda) \\
&p(X = 1) = \lambda \\
&p(X = 0) = 1 - \lambda
\end{aligned}
$$

While under the frequentist methodology we'd assume that λ is a constant, under the bayesian methodology we model it as a random variable with its own probability distribution. Our first step is to choose an appropriate approximation for this distribution using past data. We call this our prior distribution of λ.

We then set our loss threshold, which is the largest expected loss we're willing to accept if we were to make a mistake. As with any statistical modelling, bayesian experimentation methods are built on approximations of real world data. As such, there is always a chance that we draw the wrong conclusions from the test. This loss threshold allows us to say that even if we did draw the wrong conclusions, we can be confident that the conversion rate wouldn't drop more than an amount we're comfortable with.

Finally, we draw samples in the form of a randomised experiment and use these to update the distribution, and thus our beliefs, about λ under the control and treatment versions of the upsell screen. We can use these posterior distributions to calculate the probability that treatment is better than control and expected loss of wrongly choosing treatment.

So our first step is to choose a prior distribution of λ. To do this we can look at the data we've recently (last few weeks) gathered about this conversion. I've generated a sample prior data set which we can use for this exercise.

```
import pandas as pd

prior_data = pd.read_csv('prior_data.csv')

print(prior_data.head())
print(prior_data.shape)
```

```
                             userId  converted
0  b80c5365834c493197b54a269c1675ef      False
1  e23c53483efe48f39cbed3976b1c4c2e       True
2  ba86c09437214b6fae4fa5043d28979c       True
3  12422332fcb44d4cb8e599a894fef13a       True
4  0a7f0f5939fb45b5951ebe2ba085f8f3      False
(5268, 2)
```

Since this data set is artificially generated, it's already in the ideal format for this exercise. In the real world, it's likely that we'd need to perform some ETL operations in order to get the data in this format. However this is outside the scope of this post.

We see that we have a sample size of 5268 users, and for each user we can see whether they converted on this screen or not. We can go ahead and work out the prior conversion rate.

```
conversion_rate = prior_data['converted'].sum()/prior_data.shape[0]
print(f'Prior Conversion Rate is {round(conversion_rate, 3)}')
```

```
Prior Conversion Rate is 0.321
```

Choosing a Prior Distribution

We see that our prior data gives us a conversion rate of ~30%. We now use this to choose a prior distribution for λ. Choosing a prior is an important aspect of bayesian experimentation methods, and deserves a post on its own. I will be diving into it in more depth in part 4 of this series. However, for the purposes of this post, I will be using a rough method of choosing a prior.

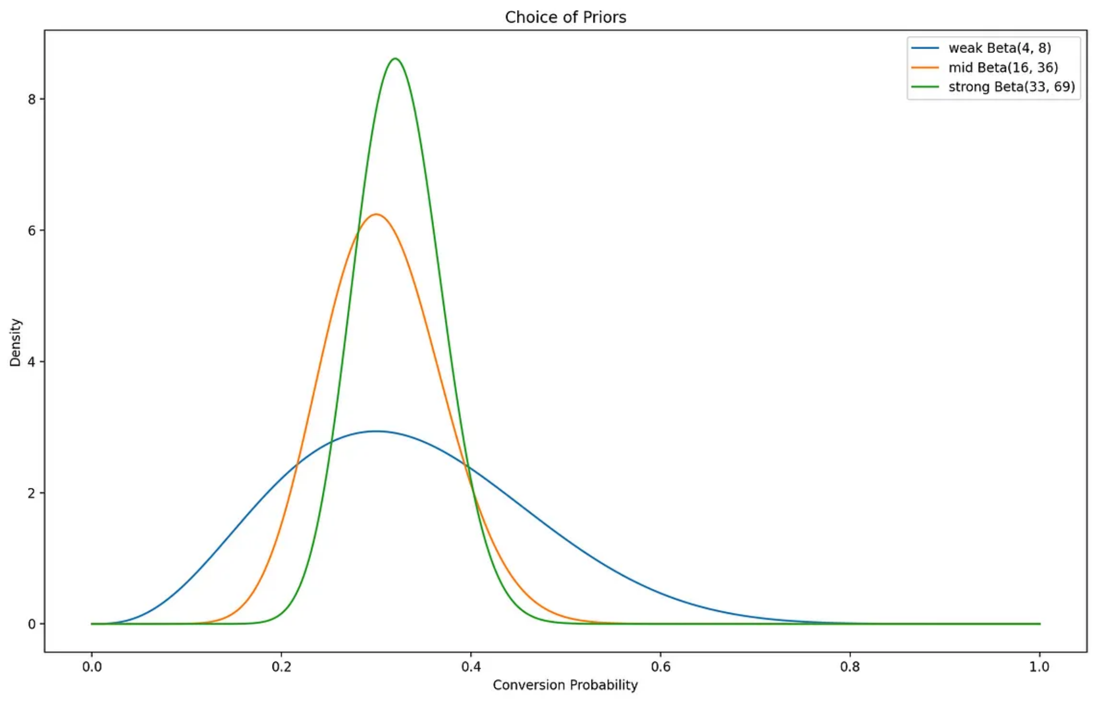
We're going to use the beta distribution to model our conversion rate since it's a flexible distribution over [0,1] and is also a good conjugate prior. It will make our calculations easier when we work out posteriors with experiment data.

When choosing a prior distribution for our conversion, it's best practice to choose a weaker prior than the prior data suggests. Once again I will explore this in more depth in part 4 of this series, but essentially, choosing too strong a prior could result in our posterior distributions being wrong and could therefore lead to wrong calculations and conclusions.

With that in mind, let's look at potential priors of varying strength.

```
import numpy as np
from scipy.stats import beta
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1)

x = np.linspace(0,1,1000)

beta_weak = beta(round(conversion_rate, 1)*10 + 1, 10 + 1 -
round(conversion_rate, 1)*10)
```

```
beta_mid = beta(round(conversion_rate, 1)*50 + 1, 50 + 1 - 
round(conversion_rate, 1)*50)

beta_strong = beta(round(conversion_rate, 2)*100 + 1, 100 + 1 - 
round(conversion_rate, 2)*100)

ax.plot(x, beta_weak.pdf(x), label=f'weak 
Beta({int(round(conversion_rate, 1)*10) + 1}, {10 + 1 - 
int(round(conversion_rate, 1)*10)})')

ax.plot(x, beta_mid.pdf(x), label=f'mid 
Beta({int(round(conversion_rate, 1)*50) + 1}, {50 + 1 - 
int(round(conversion_rate, 1)*50)})')

ax.plot(x, beta_strong.pdf(x), label=f'strong 
Beta({int(round(conversion_rate, 2)*100) + 1}, {100 + 1 - 
int(round(conversion_rate, 2)*100)})')

ax.set_xlabel('Conversion Probability')
ax.set_ylabel('Density')
ax.set_title('Choice of Priors')
ax.legend()
```



Choice of Priors (Image by Author)

Here we can see three prior distributions which have a mean conversion rate of ~30%, similar to our prior data, and are all much weaker than the true distribution of the prior data.

Let’s choose a prior that is in between the weakest and mid priors in the diagram $Beta(7{,}15)$

$$p(\lambda) = \frac{\lambda^6 (1-\lambda)^{14}}{B(7,15)}$$

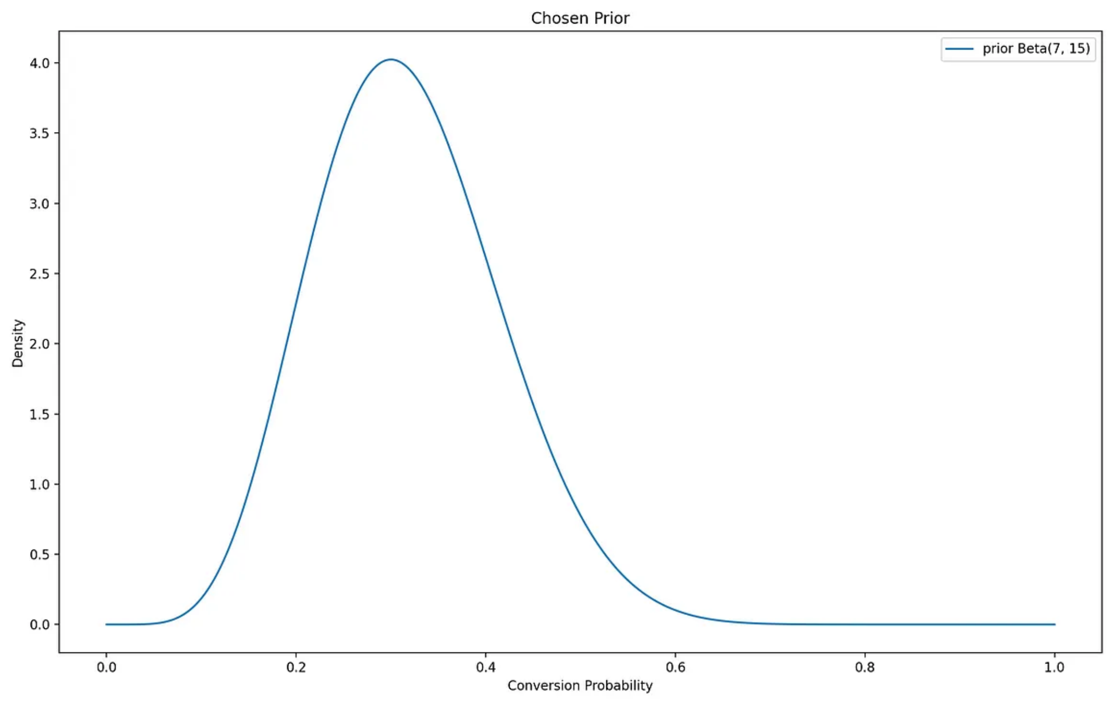
where $B(a{,}b)$ is the beta function defined as

$$B(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}dx$$

and has the property

$$B(a,b) = \frac{(a-1)!(b-1)!}{(a+b-1)!}$$

```
prior_alpha = round(conversion_rate, 1)*20 + 1
prior_beta = 20 + 1 - round(conversion_rate, 1)*20

prior = beta(prior_alpha, prior_beta)

fig, ax = plt.subplots(1, 1)

x = np.linspace(0,1,1000)

ax.plot(x, prior.pdf(x), label=f'prior
Beta({int(round(conversion_rate, 1)*20) + 1}, {20 + 1 -
int(round(conversion_rate, 1)*20)})')
ax.set_xlabel('Conversion Probability')
ax.set_ylabel('Density')
ax.set_title('Chosen Prior')
ax.legend()
```



Chosen Prior (Image by Author)

Setting a loss threshold

Now that we've chosen our prior, we need to choose our ϵ which is the highest expected loss we're willing to accept in the case where we mistakenly choose the wrong variant. Let us assume that this is an important conversion for us so we want to be pretty conservative with this ϵ. We aren't willing to accept a relative expected loss of more than 0.5%. So we set $\epsilon=0.005*0.3=0.0015$.

We have a prior and a threshold for our expected loss, so we can start running our experiment and gathering data from it.

Experiment Results

Let's assume that we've left our experiment running for a couple of weeks and want to check whether we can draw any conclusions from it. In order to do this we need to use our experiment data to calculate our posterior distributions, which we can then use to calculate the probability of each variant being better, and also the expected loss of wrongly choosing each variant.

For the purposes of this exercise, I've generated a sample experiment data set. Let's start off by exploring it and aggregating it to find out the conversion rates of each variant.

```
experiment_data = pd.read_csv('experiment_data.csv')

print(experiment_data.head())
print(experiment_data.shape)
```

```
                             userId      group  converted
0  9db0424f91294a7f98915f00828d64f3    control       True
1  c144e1fa696a459ea6b46aa5d7e3b463  treatment      False
2  240ebd8f5de24b6eb1ef7e0ca6b89af4    control      False
3  c4b40d58a74d49248b8f6d49aecffeed    control       True
4  8fe15a47f7534755ac3375f0ace1e3ec  treatment      False
(5101, 3)
```

We see that the data set is similar to the prior data set with an extra column for specifying which the group the user was allocated to and therefore which variant they saw. Once again it's worth noting that since this data set is artificially generated, it's already in the ideal format for this exercise without the need for additional ETL operations.

We can now go ahead and aggregate the data.

```
results = experiment_data.groupby('group').agg({'userId':
pd.Series.nunique, 'converted': sum})

results.rename({'userId': 'sampleSize'}, axis=1, inplace=True)

results['conversionRate'] =
results['converted']/results['sampleSize']

print(results)
```

```
           sampleSize  converted  conversionRate
group
control          2743        886        0.323004
treatment        2358        821        0.348176
```

We can tell by inspection that treatment had a better conversion rate, but we need to perform further calculations in order to update our beliefs about the respective conversion probabilities λ_c and λ_t of the control and treatment variants.

With our experiment data, we can now calculate our posterior distributions under each variant. But before we do let's explore the maths behind how this is possible from just the information that we have. We're going to use a theorem[1] that states the following:

Suppose we have the prior

$$p(\lambda) = f(\lambda; a, b) = \frac{\lambda^{a-1}(1-\lambda)^{b-1}}{B(a,b)}$$

Suppose a variant was displayed to *n* visitors and *c* converted, then the posterior distribution of the variant is given by

$$p(\lambda|n, c) = f(\lambda; a + c, b + n - c)$$

Let's go ahead and prove this. By bayes theorem we have the following

$$p(\lambda|n, c) = \frac{p(n,c|\lambda)p(\lambda)}{p(n,c)} = \frac{p(n,c|\lambda)p(\lambda)}{\int_0^1 p(n,c|\lambda)p(\lambda)d\lambda} \quad (1)$$

Since we modelled each conversion as a Bernoulli RV with probability λ, given λ we can model the result of showing a variant to *n* visitors as a Binomial RV. So we have

$$p(n, c|\lambda) = \binom{n}{c} \lambda^c (1 - \lambda)^{n-c}$$

And thus, using the definition of our prior

$$p(n, c|\lambda)p(\lambda) = \binom{n}{c} \frac{1}{B(a,b)} \lambda^{a+c-1}(1 - \lambda)^{b+n-c-1} \quad (2)$$

Let's now just consider the coefficients

$$\binom{n}{c} \frac{1}{B(a,b)} = \frac{n!}{c!(n-c)!} \frac{1}{B(a,b)} = \frac{1}{B(c+1,n-c+1)} \frac{1}{B(a,b)} \quad (3)$$

Using the definition of the beta function we can say

$$\begin{aligned} B(c+1, n-c+1) * B(a,b) &= \int_0^1 x^c(1-x)^{n-c}dx * \int_0^1 x^{a-1}(1-x)^{b-1}dx \\ &= \int_0^1 x^{a+c-1}(1-x)^{b+n-c-1}dx = B(a+c, b+n-c) \qquad (4) \end{aligned}$$

Substituting (3) and (4) back into (2)

$$p(n,c|\lambda)p(\lambda) = \frac{1}{B(a+c,b+n-c)}\lambda^{a+c-1}(1-\lambda)^{b+n-c-1} = f(\lambda; a+c, b+n-c) \; (5)$$

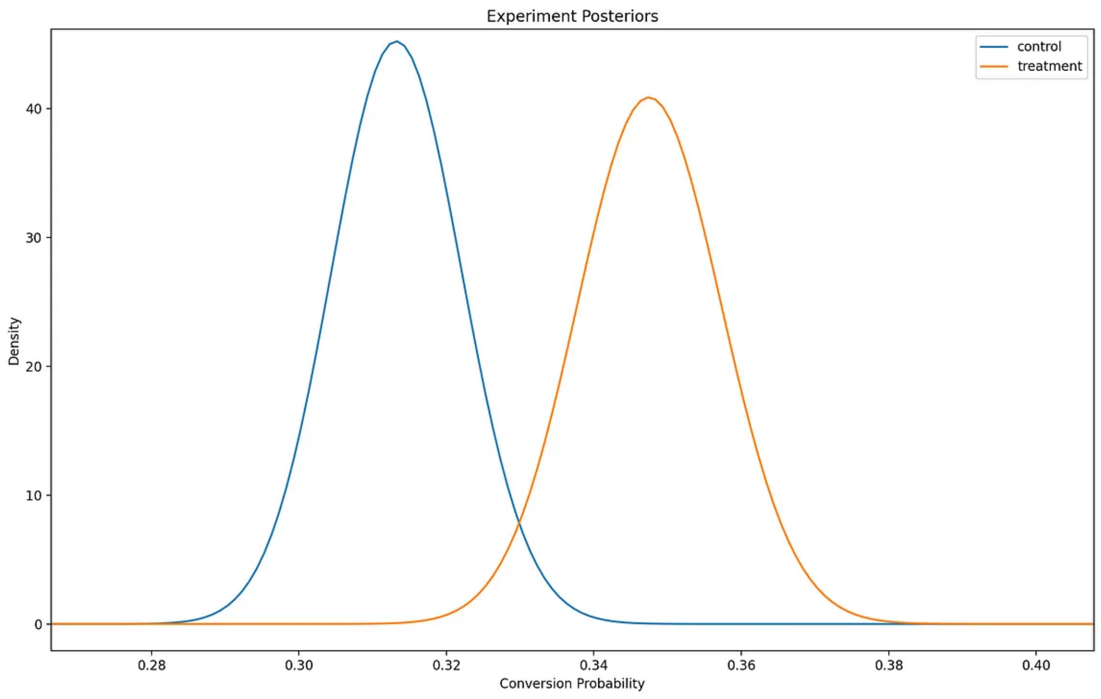
Substituting (5) back into (1)

$$p(\lambda|n,c) = \frac{f(\lambda;a+c,b+n-c)}{\int_0^1 f(\lambda;a+c,b+n-c)d\lambda}$$

Since $f(\lambda;a+c,b+n-c)$ is a distribution over [0,1] the denominator is 1 and we have the result we were after.

Now that we've sufficiently convinced ourselves of the maths, we can calculate our posterior distributions.

```
control = beta(prior_alpha + results.loc['control', 'converted'],
prior_beta + results.loc['control', 'sampleSize'] -
results.loc['control', 'converted'])

treatment = beta(prior_alpha + results.loc['treatment',
'converted'], prior_beta + results.loc['treatment', 'sampleSize'] -
results.loc['treatment', 'converted'])

fig, ax = plt.subplots()

x = np.linspace(0.26,0.42,1000)

ax.plot(x, control.pdf(x), label='control')
ax.plot(x, treatment.pdf(x), label='treatment')
ax.set_xlabel('Conversion Probability')
ax.set_ylabel('Density')
ax.set_title('Experiment Posteriors')
ax.legend()
```



Experiment Posteriors (Image by Author)

Now that we've got our posterior distributions, we can go ahead and calculate the joint posterior. Since randomised experiments are built on the idea of random assignment of a user to a variant, we can assume that these two distributions are independent. Note that this isn't always the case. For example, there might be some cases where network effects might be in play and we'd need to take that into consideration. This assumption is also dependent on the procedure of random assignment working properly.

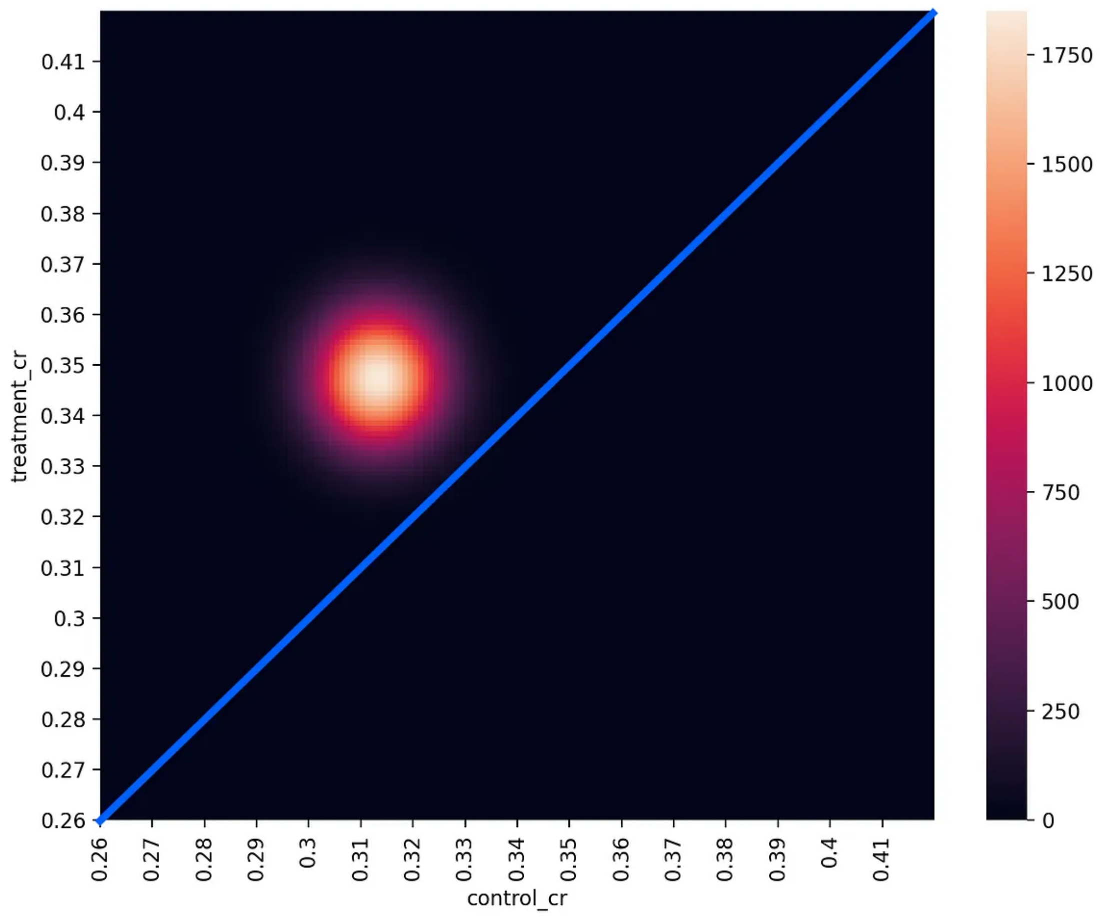
Let us assume that our method of random assignment has worked properly and that there are no network effects. Under this assumption we can say the following:

$$p(\lambda_c, \lambda_t|data) = p(\lambda_c|data) * p(\lambda_t|data)$$

Let's use this to calculate our joint posterior distribution.

```
import seaborn as sns

joint_dist_for_plot = []
for i in np.linspace(0.26,0.42,161):
    for j in np.linspace(0.26,0.42,161):
        joint_dist_for_plot.append([i, j,
control.pdf(i)*treatment.pdf(j)])
```

```
joint_dist_for_plot = pd.DataFrame(joint_dist_for_plot)

joint_dist_for_plot.rename({0: 'control_cr', 1: 'treatment_cr', 2:
'joint_density'}, axis=1, inplace=True)

tick_locations = range(0, 160, 10)
tick_labels = [round(0.26 + i*0.01, 2) for i in range(16)]

heatmap_df = pd.pivot_table(joint_dist_for_plot,
values='joint_density', index='treatment_cr', columns='control_cr')

ax = sns.heatmap(heatmap_df)
ax.set_xticks(tick_locations)
ax.set_xticklabels(tick_labels)
ax.set_yticks(tick_locations)
ax.set_yticklabels(tick_labels)
ax.invert_yaxis()
```



Joint Posterior (Image by Author)

The blue line in this graph is the line that represents *λ_c=λ_t*. Since the joint posterior is above this line, we can use this as a visual indication that the treatment is in fact better. If the joint posterior was below the line then we could be pretty sure that control was better. If any part of the joint posterior was on the line then there would be more uncertainty on which variant was better.

In order to quantify this, we need to calculate *p*(*λ_t*≥*λ_c*) and *E*[*L*](*t*), the expected loss of wrongly choosing treatment.

```
import decimal
decimal.getcontext().prec = 4

control_simulation = np.random.beta(prior_alpha +
results.loc['control', 'converted'], prior_beta +
results.loc['control', 'sampleSize'] - results.loc['control',
'converted'], size=10000)

treatment_simulation = np.random.beta(prior_alpha +
results.loc['treatment', 'converted'], prior_beta +
results.loc['treatment', 'sampleSize'] - results.loc['treatment',
'converted'], size=10000)

treatment_won = [i <= j for i,j in zip(control_simulation,
treatment_simulation)]

chance_of_beating_control = np.mean(treatment_won)

print(f'Chance of treatment beating control is
{decimal.getcontext().create_decimal(chance_of_beating_control)}')
```

```
Chance of treatment beating control is 0.9718
```

From the simulations we see that *p*(*λ_t*≥*λ_c*)=0.9718 so treatment has a 97% chance of being better than control.

Now that we've calculated the probability of treatment being better, we need to calculate *E*[*L*](*t*). The loss function of each variant is given by

$$L(\lambda_c, \lambda_t, t) = max(\lambda_c - \lambda_t, 0)$$
$$L(\lambda_c, \lambda_t, c) = max(\lambda_t - \lambda_c, 0)$$

So the expected loss of each variant is given by

$$E[L](t) = \int_0^1 \int_0^1 L(\lambda_c, \lambda_t, t)p(\lambda_c, \lambda_t | data)d\lambda_t d\lambda_c$$
$$E[L](c) = \int_0^1 \int_0^1 L(\lambda_c, \lambda_t, c)p(\lambda_c, \lambda_t | data)d\lambda_t d\lambda_c$$

We use this to calculate the expected loss[2]

```
decimal.getcontext().prec = 4

loss_control = [max(j - i, 0) for i,j in zip(control_simulation,
treatment_simulation)]

loss_treatment = [max(i - j, 0) for i,j in zip(control_simulation,
treatment_simulation)]

all_loss_control = [int(i)*j for i,j in zip(treatment_won,
loss_control)]

all_loss_treatment = [(1 - int(i))*j for i,j in zip(treatment_won,
loss_treatment)]

expected_loss_control = np.mean(all_loss_control)
expected_loss_treatment = np.mean(all_loss_treatment)

print(f'Expected loss of choosing control:
{decimal.getcontext().create_decimal(expected_loss_control)}.
Expected loss of choosing treatment:
{decimal.getcontext().create_decimal(expected_loss_treatment)}')
```

```
Expected loss of choosing control: 0.02516. Expected loss of choosing treatment: 0.0001369
```

From running simulations we see that:

$E[L](t) = 0.0001369 < 0.0015 = \epsilon$

Since the expected loss of one of the variants is below the threshold that we set at the start of the test, the test has reached significance. We can conclude with high confidence that the treatment is better, and that the expected cost of mistakenly choosing treatment would not be greater than what we're comfortable with. So we can strongly recommend that the treatment variant of the upsell screen should be rolled out to the rest of our user base.

I hope this case study was useful in helping you understand the calculations required to implement bayesian AB testing methods. Watch this space for the next parts of the series!

References

[1] VWO Whitepaper by C. Stucchio

[2] Bayesian A/B testing — a practical exploration with simulations by Blake Arnold — I've used the logic from Blake's code for calculating expected loss

Also found The Power of Bayesian A/B Testing by Michael Frasco very helpful in understanding the technical aspects of bayesian AB testing methods

My code from this post can be found here

Thanks for reading this article! I hope it helped you get a better understanding of how to implement bayesian AB testing methods for conversion metrics.

If you enjoy reading my articles, would like to support my writing and are thinking of getting a Medium subscription, feel free to use my referral link below. I'd get a percentage of your subscription fees.

Ab Testing Statistics Product Management Data Science Experiment