Steve Hanov's Blog

I know how to make and sell software online, and I can share my tips with you.
Email | Twitter | LinkedIn | Comics | All articles

< **20 lines of code that will beat A/B testing every time** >

Posted five years ago

Zwibbler.com is a drop-in solution that lets users draw on your web site.

A/B testing is used far too often, for something that performs so badly. It is defective by design: Segment users into two groups. Show the A group the old, tried and true stuff. Show the B group the new whiz-bang design with the bigger buttons and slightly different copy. After a while, take a look at the stats and figure out which group presses the button more often. Sounds good, right? The problem is staring you in the face. It is the same dilemma faced by researchers administering drug studies. During drug trials, you can only give half the patients the life saving treatment. The others get sugar water. If the treatment works, group B lost out. This sacrifice is made to get good data. But it doesn't have to be this way.

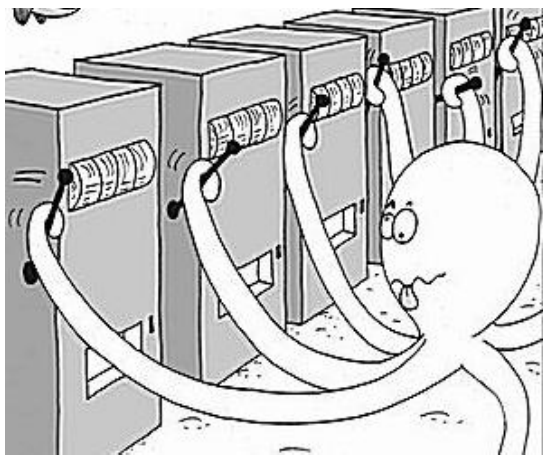
In recent years, hundreds of the brightest minds of modern civilization have been hard at work not curing cancer. Instead, they have been refining techniques for getting you and me to click on banner ads. It has been working. Both Google and Microsoft are focusing on using more information about visitors to predict what to show them. Strangely, anything better than A/B testing is absent from mainstream tools, including Google Analytics, and Google Website optimizer. I hope to change that by raising awareness about better techniques.

With a simple 20-line change to how A/B testing works, **that you can implement today**, you can *always* do better than A/B testing -- sometimes, two or three times better. This method has several good points:

- It can reasonably handle more than two options at once.. Eg, A, B, C, D, E, F, G, Ã¢ï¿½Â¦
- New options can be added or removed at any time.

But the most enticing part is that **you can set it and forget it**. If your time is really worth $1000/hour, you really don't have time to go back and check how every change you made is doing and pick options. You don't have time to write rambling blog entries about how you got your site redesigned and changed this and that and it worked or it didn't work. Let the algorithm do its job. This 20 lines of code automatically finds the best choice quickly, and then uses it until it stops being the best choice.

The Multi-armed bandit problem



Picture from Microsoft Research

The multi-armed bandit problem takes its terminology from a casino. You are faced with a wall of slot machines, each with its own lever. You suspect that some slot machines pay out more frequently than others. How can you learn which machine is the best, and get the most coins in the fewest trials?

Like many techniques in machine learning, the simplest strategy is hard to beat. More complicated techniques are worth considering, but they may eke out only a few hundredths of a percentage point of performance. One strategy that has been shown

Steve Hanov makes a living working on Rhymebrain.com, PriceMonkey.ca, www.websequencediagrams.com, and Zwibbler.com. He lives in Waterloo, Canada.

Post comment

Chase from ReSci edit
five months ago
Love this. It's four years old but some people still dont' use bandit type software.

Now we even have AI to really help marketers crush their A/B tests.

Check out the post I just wrote around A/B testing levels of sophistication and how technology is changing things in 2017:

www.retentionscience.com/ab-testing/

lhpopov edit
eleven months ago
Very interesting article.

I just found out that Kameleoon does it.

"Multi-armed bandit tests

(Adaptive traffic distribution)"

from kameleoon.com/en/pricing-ab-testing-personalization.html

Also conductrics.com/real-time-optimization/ state "Using machine learning methods".

Also vwo.com/features/ with their "Multivariate Testing".

Probably you are not the first one to come up with this idea, but for sure gave it to some of the listed above players :-) Good work!

Eric edit
one year ago
How do you account for this method if you are trying to rank listings on a product page for an online marketplace (i.e. Ebay, etsy, Amazon)?

MoreCowbell edit
one year ago
Hi! I'm a time traveler who has gone back to 2013, the year this article was written. Google Website Optimizer still exists here in in the past, but Google Content Experiments (which does what this article describes) was launched in 2012 as Optimizer's eventual replacement. Live in the future, my friends.

emanuel edit
one year ago
Excellent post, thanks for sharing!

Love the site UX btw. It's super straight forward and works well. Fancyness is overrated.

webxl edit
one year ago
Pretty cool stuff. What the heck is "¢ï¿½Â¦"?

Re. this comment system: The UI is a lot more interesting than most blogs. Everything wraps the blog entry. Very simple and straightforward. I like it. But

Comment spam defeated at last

For years when running this blog, I would have to log in each day and delete a dozen comments due to spam. This was a chore, and I tried many ways to stem the tide.

implement an entire vector graphics API in a few lines of Javascript?

severalwordsstucktogether.

button; you want to see the increase in CTR/conversions/whatever with respect to the control group; but.. I think that somehow your approach is shorter than the one with A/B variant, where you should do at the end a follow-up test running only the winner in order to validate the result.

So.. it seems very nice approach, but, there are some software (saas) that are already doing this, not with 20 lines of code of course (and with a lot of money). There are MVT or AB SaaS that instead of leaving the owner to choose from the possible winners he choose automatically the winner during the test.

One little thing I am afraid is that you want to test different "creatives"/banners/images/html-banners on different sections of a product page for example, you should write this 20 lines of code on each zone that has multiple variants to choose. So, it will be quite messy inside the script source that generates that specific web page.

The approach using SaaS that uses section-divs where you upload creatives/banners/images it's easier than writing 20 lines of codes for each zone that we want to test especially if you are not a programmer, or you cannot hire one for this task. The reports are also nice.. but, this programming approach you are proposing is quite cost effective I suppose.

Anyway it's worthing to explore this solution. Could be more cost effective for many little A/B tests.

Nice article! Thx.

PS. Sorry for my English as I'm not a native English speaking guy.

button tested on Tuesday afternoon. Similarly, if your website gets slashdotted, you may have a sudden spike in visitors who might be either totally uninterested about buying your product (they just want to read a cool thing you wrote) or completely determined in signing up for your cool new service. And then there are seasonal items. Your Halloween themed "buy" button might perform quite well during Halloween, but how long will it remain to the top after Halloween?

3. Fashion and design trends. On the web, the context changes.

By context, I mean overall standards and conventions in web design. That glossy button of yours that has accumulated outstanding conversion rating over time is just not "in" any more. Visitors are now used to more subtle interfaces like the one of Facebook. Unless you have a mechanism to decay the value of clicks over time you will end up with "winning" options that endure when they shouldn't.

The problem here is that for this system to work, it needs to run in a controlled, static environment for a significant period of time. And you really don't have that:

Imagine you are in a casino trying to run your algorithm on a wall of one armed bandits. 5 minutes after you start, a bunch of guys come in and start playing on the same machines as you. Then repairmen come and add twice as many machines. Half of those are a new model. Then they update the OS of 4 of the machines. On and on. Does it still feel like your algorithm would work in that environment? That's far closer to the environment in which most websites operate

Each time a change is introduced, you need a full reset, but this algorithm is useful only when it is allowed to run long enough to reach a statistically significant result, and that makes it poorly suited for website testing, at least for most websites. It might work for huge websites where the numbers of visitors are so

easy as writing blocking code. Instead, developers in node.js need to manage many levels of callbacks. Today, we will examine four different methods of performing the same task asynchronously, in node.js.

of views. After several thousand page views, you can be reasonably certain that your CTR is stable. My point is that if the data is not very well distributed, it could take a much longer time to reach the number of page views needed to make the determination.

This is kinda like quick sort. In most cases, with a random distribution, it will run O(n log (n)), but in the worst case, mathematically, it's O(n^2). I think the algorithm described above could really make great improvements over the current standard A/B testing, but anyone that uses it needs to know the pros and cons so that it can be tweaked properly. In some cases, maybe the 90-10 split could be more optimized at 20-80 or 30-70. It really depends on what kind of data you have and finding a "sweet spot" for it. With careful analysis of the specific application, it could prove to be very powerful... but you do have know what's going on and make accurate assumptions about the data. The situation that I thought of where this would not be optimal is if you have a lot of data initially for on of the tests that doesn't match the eventual CTR after x number of views.

automated saves so much time down the road, and talk about great stats to provide to your clients.

won't show them orange, since we think it
hoose Green. They don't click. The same
nd we end up cycling through the choices.
e click through rate for each option

ge button! Quickly, the browser makes an
'/reward?testname=buy-button"); and our

, he scratches his head. What the F*? The
is tiny! The green button is obviously the
n will always choose it forever now!

is really the suboptimal choice. Since the
l always be shown. That is, until it stops
to look better.

there is one, will have been found, and will
esults based on an actual web site that I
timate of the click through rate for each

rapidly decrease until A starts getting shown again.

All other things being equal, this method will show you which option gives you the best CTR, which is all you really care about anyway.

enough to clear the hurdle presented, it was an 'unnecessary loss.')

I found that unnecessary losses stayed relatively the same for all randomization trials. That was a surprise.

So, try for a low randomization number. That seems to indicate that the randomization is just there in case things change. For a static set of options and static customers, no randomization gets us to the "right" answer fastest with the best average click-through rates.

Anyways, happy to share my spreadsheet with anyone who wants to see it.

control over.

I want to build this! I want to build it now! If anyone else feels the same and can help, I'd love to do it.

Craig.

algorithm works because it's behaviour mirrors user behaviour. Your population of users will, with some probability alpha agree on what is the best looking button or copy-edit or whatever. And with probability 1-alpha they will choose something else. But the space of something else is large, and each user will choose a different thing in that space from the others. So the critical error with this approach is actually its greatest strength.

It marginalizes random user disagreements to the point where they become totally insignificant.

large that statistical significance can be reached before the context changes, with some tweaks taking into account things like the time of day/week/season, running a counter reset on unexpected variations (if button A suddenly converts 3x as much as before when it had already been tested often enough to reach statistical significance, something is up)

AB testing, on the other end, does not suffer as much from this volatile environment because it does not try to favor results before statistical significance is reached. On Sunday or at night, both A & B suffer/benefit equally, whereas with your system, a design which might have been ahead on Friday by a large margin may loose its advance over the week-end, get overtaken on Monday, finally recover towards the end of the day only to tank again overnight. Some potentially interesting options might take a long time to reach statistical significance.

Another advantage of AB testing, and possibly the most important issue here, is that it teaches us and helps us understand what is happening. After a few tries, you can work out some general rules like: "bigger buttons are better", "fewer options are necessary for non qualified visitors, but qualified visitors (from such and such websites) will fill out longer forms", "Pictures can control the attention of visitors", etc. which can then guide your UI evolution.

This can be done to a point with your system, but it's much less reliable: "A" has a conversion rate of 60% and "B" a conversion rate of 22%, but "B" has been tested 600k times in the low season and "A" has only been tested 900 times in the high season. Is "A" really better than "B"? You can't really compare A and B which prevents you from learning as much as you would in AB testing. You are stuck guessing and are unable to extract and verify the rules that work for your website.

Also, AB testing is more hands on which forces you to think with the data: Your Halloween button will never live through Christmas because of the fantastic sales from October/November with AB testing (or your usual "best" button will be shown 90% of the time through Christmas because the new Christmas buttons didn't get a good enough conversion early in December and by the time they recovered from that, the Christmas buying season is over)

There is this old quote - I don't remember where it's from or how it goes exactly, but I think it is pretty applicable here: "When trying to write software that learns by itself, you find out that it doesn't... but you do."

I wasn't quite planning on posting a thousand words in the comments. If you feel like responding to it, I would be happy to hear about it. You can contact me at dev@preptags.com.

6%

art. The randomization of 10% of trials
s. It is a trade-off between trying new
sticking with what it knows will work. There
y strategy. In the epsilon-first strategy, you
ing and once you have a good sample,
an have it decrease the amount of
eedy strategy that I have described is a
ormance. Learning about the other
oration, and methods that take context into
just want something that works.

s?

rstand. People distrust things that they do
st machine learning algorithms, even if
pport this, because then you'd have to
es, and that is hard. Some common

erent rates will skew the results. (No it
of the click through rate for each choice)
itors probably don't change. But if you
, multiply the old reward value by a

hings at once that depend on each-other.

for 30 days so how can I reward it?