AI in Digital Marketing

Leveraging Power of Reinforcement learning in Digital Marketing

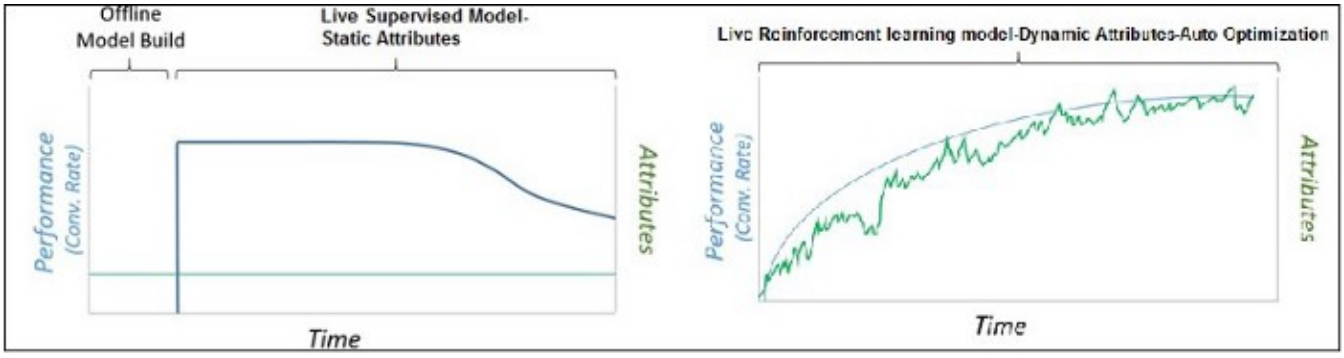
Sourish Dey Jun 26, 2019 · 11 min read ★

In my previous article, I discussed an advanced analytics solution to increase campaign ROI or *Return on Marketing Investment* (ROMI) thorough propensity modeling techniques. While supervised learning is mostly used (at least so far) method in the predictive analytics industry, it has few limitations.

Firstly as supervised learning uses static lists of attributes, which if not refreshed in regular intervals, becomes irrelevant in the course of time. That means these techniques are not prone adaptive learning/self-learning (discovering something new intelligence apart from the rule learned during training), i.e. when there is a change in the new data IV(independent variable) or population characteristics, the algorithm's performance degrades. We tend to see the models are most predictive the closer they are launched to the learning period. Over time external factors (such as shifts in customers' purchasing behaviors) reduce the predictive power of the model leading to the need for an offline rebuild process. Secondly, it is very hard to make it great with bad data — 'garbage in and garbage out'.

Here reinforcement learning/online learning(foundation stone of AI) comes to solve these problems in a great extent. While supervised learning predicts a class and is trained on class, reinforcement algorithm learns from the reward/punishment and updates itself, this continues over time. So while Supervised learning is stateless, RL has a temporal dimension, it looks not only at the past state but also at the state now. So as there is always a real-time update in an algorithm, optimizations occur close to real-time with dynamic attribute lists, minimizing the impact of external factors. So here the model performance characteristic is just the opposite of supervised learning- initially, the performance is sub-optimal and when the algorithm sees more data it learns from the reward/punishment structure its predictive power increases with updated guiding principle.

Performance Characteristics over time



Supervised Model vs. Reinforcement Learning

In this article we will solve a business problem in the digital marketing arena using Upper Confidence Bound(UCB)- a breed of *Reinforcement Learning family.*

Business scenario and RL framework

As the Credit card companies and banks are aspiring and competing to be fully digital with a limited marketing budget and deadlines to realize the tangible $ impact, it is imperative to shift from conventional test and target strategy to next-gen approach. So far in digital marketing and web analytics A/B testing is the prevalent method to compare digital campaigns,chose the winning ad and to decide targeting strategy.

Although it is easy to design and implement, however, this hypothesis based classical approach has few drawbacks.

1) No memory: It is an independent test every time, not linked with knowledge gained about earlier campaigns. A previous knowledge(prior probability) generally strengthen the confidence of posterior probability and hence the decision-making process.

2) Slow and expensive: As it is a two-step process, first it explores the opportunities through a test campaign and then based on the comparative analysis it exploits the winning digital ad or creative to reap the $ benefit. On the other way, the reinforcement learning algorithm acts as a self-learning engine and optimizes both exploration and exploitation at the same time.

3) No guiding principle: There is no self-learning and feedback system to continually update outcomes. Just because creative X performed better over creative Y one year ago does not mean that it will still perform better now.

In the next part of article, we will discuss the business case and reinforcement learning based solution including implementation in python and R.

Digital Campaign case study:

A multinational bank is planning to acquire customers through a new campaign in digital space, providing a special spend offer for a movie ticket for one of its newly launched credit card product. There is a detailed cost-benefit analysis and decisions about offer structure. However the digital marketing team is exploring six different creatives and based on the reward/return (click-through rate(CTR)) in test campaign, the winning ad will be shown to the home page, social media, and other partner sites. One option is to do multiple A/B tests with large enough samples and choose the best ad, which will be both time consuming and expensive and as a pure exploration method there is no significant exploitation (simultaneously) of the best ad.

Let's see how Upper Confidence Bound (UCB) algorithm, a reinforcement learning technique, which is closer to the field of Artificial Intelligence(AI) helps us doing exploration and exploitation at the same time to maximize our returns.

Fig- Creative versions for Digital Campaign

So In nutshell, our problem has the following steps:

i)we have six creative versions(d) for a digital campaign and each time a user/visitor connect to web page we display one version of this ad.

ii) Each time a user connects to this web page, counts as a round(n).

iii) At each round n, creative d gets a reward rd (n) ϵ {0,1}: rd (n) = 1 if the user clicks on the ad d, 0 if the user didn't.

iv) Our goal is to maximize the total reward we get over many rounds (i)

UCB algorithm and Multi-Armed Bandits problem:

The intuition of UCB algorithm comes from the working principle of casino slot machines in context with famous Multi-Armed Bandits problem(to choose the best slot

machine, among machines with different probability of success, to maximize profit). Clearly, our business problem is no different than Multi-Armed Bandits problem and the following are steps of a simple version of UCB algorithm:

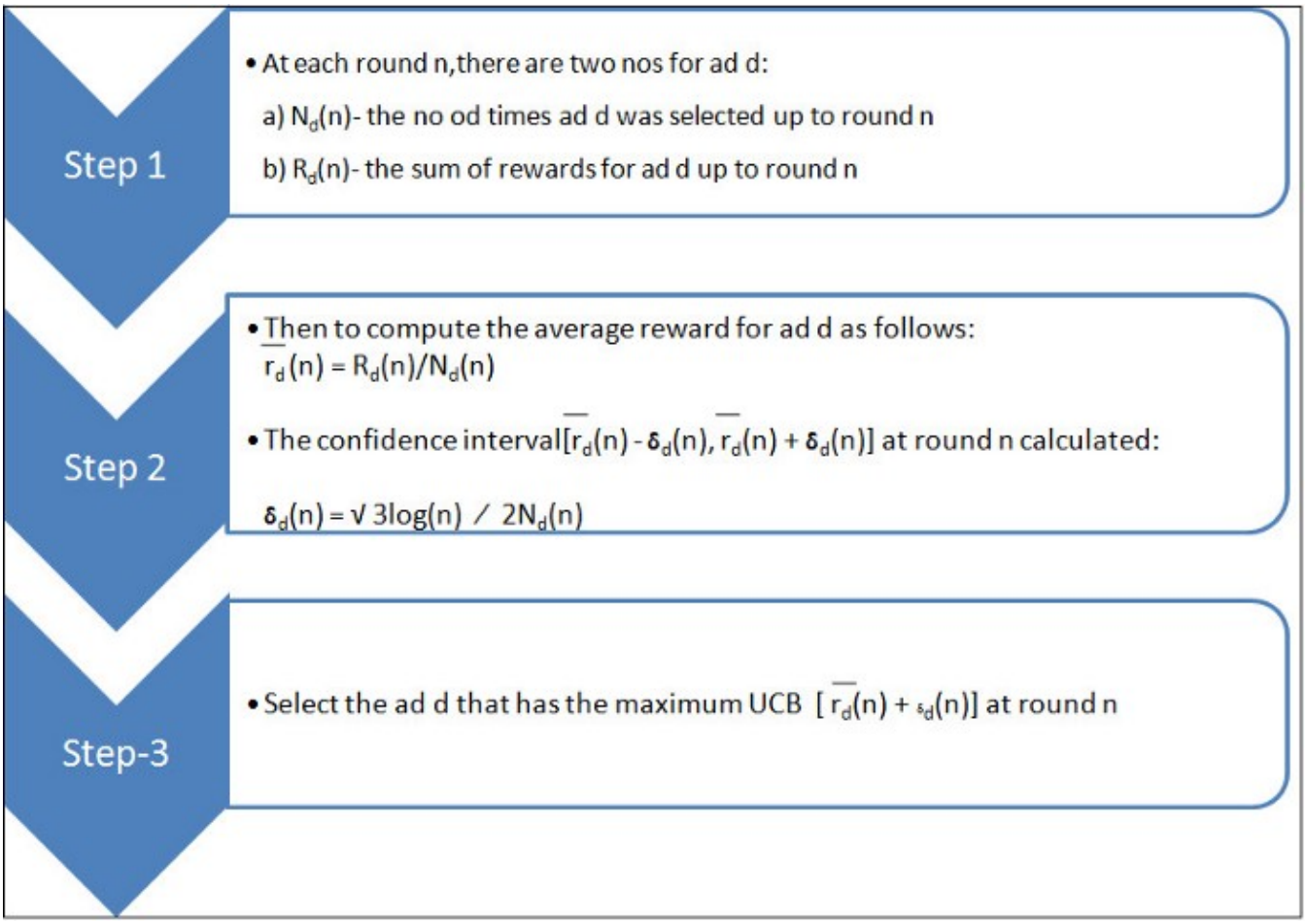


Fig 2: UCB Algorithm and Multi-armed Bandit

In simple words, the algorithm initially assumes some prior information (or some assumption after few rounds) about the options regarding the probability distribution of customer preference and expected returns of these distributions. During the iterations, the algorithm gains intelligence based on user action — reward (clicks)/punishment(no click) and in each round(iteration) the expected value(average reward) as specified in step 2. Then the second thing in step 2 happens is that, with additional information about each option after every round, it becomes more confident about the expected value and with sample size increases it continually updates the confidence interval(refer step 2) by shrinking the confidence bound. After gathering a large sample with thousands of iterations when it is more confident about the overall picture, it further narrowed down the confidence intervals, closer to the expected value and finally, the algorithm finds the best solution or best option. Then it starts exploiting by showing only the optimal ad to maximize ROI. Exactly this is the way to find the best slot machine in Multi-Armed Bandits problem and exploit that slot machine to make money.

Implementation

Now it's time to do some coding to follow the above steps and understand the entire process more clearly. The data set for this exercise can be downloaded from here .(this data set is prepared purely for academic interest and having no linkage between any business data). Here we will see from our data points of only 10k user actions, how UCB chooses the optimal ad and starts exploiting the option to minimize marketing cost(more ads are shown the social media, more cost is incurred). Six different ad versions are shown to the users when users' actions are tagged (clicks then reward = 1 otherwise reward = 0). [Please note this is a simulated data set for illustration purpose. However in real life we start with scratch (no data) and basis the user action in the course of iterations algorithm decides or updates its strategy to show ad version. That is why reinforcement learning is called online learning/interactive learning]

As there is no ready-made package for this algorithm, we will try to build it from scratch. This is a simple version of implementation and of course, there are other ways to implement UCB.

Coding in Python

First, let's see what is the reward w/o adopting any strategy(random selection) and will compare the performance of UCB with random display strategy

```
#importing necessary libraries
import pandas as pd
import os as os
import numpy as np
import matplotlib.pyplot as plt
import random
import math

#Dataset reading
os.chdir('C://practice/Reinforcement/UCB')
os.getcwd()
ucb = pd.read_csv('ucb_data.csv')

# check out first 10 data points
sales_data.head(10)
```


The dataset maps user action. e.g. for user 6 clicks for ad2 but doesn't click any other ad.

Randomly choose ads in different rounds and save the total clicks in 'total_reward_rand' variable and 'ad_selected' list stores different ad versions selected over rounds.

```
#Initialization
user = ucb.user
ucb = ucb.drop(['user'], axis = 1)   #Dropping user field
N = 10000                 # no of iterations
d = 6                     # no of ad versions
total_reward_rand = 0   #variable to store total rewards or clicks
throuout the iterations

#Loop through the iterations and random selection of ad
ad_selected = []       #List to store diff ad versions selected over
iterations
 for i in range(0,N):   #For loop from 0 to 9999
    ad = random.randrange(d)
    ad_selected.append(ad)
    reward = ucb.values[i,ad]
    total_reward_rand = total_reward_rand + reward
```

Will See how many clicks we get without any strategy

```
print('total clicks without any strategy:
{}'.format(total_reward_rand))

#plot- Distribution of ad_selected
plt.hist(ad_selected)
plt.title('Distribution of ads')
plt.xlabel('ad_no')
plt.ylabel('count')
plt.show()
```

total clicks without any strategy: 1287

As expected with random selection different ads are selected almost uniformly and click rate is approx 12.87% (1287/10000).[In python index starts from zero,so Ad1 is represented as 0 and so on]. Also as it's a random process result might be slightly different in another run.

Let's see whether UCB algorithm improves the click rate by exploiting the ad, having the highest probability of getting clicked. The code lines are commented and self-explanatory. However, feel free to ask.

```
#Initialization
N = 10000                 # no of iterations
d = 6                     # no of ad versions
ad_selected = []         #List to store diff ad versions selected over
iterations
```

```
no_selection = [0] * d #List to store no of times diff ad versions
selected
sum_reward = [0] * d   #List to store no of clicks diff ad versions
get upto round N
total_reward_ucb = 0   # To store Total No clicks upto round N
```

As mentioned in Fig 2, the inner loop calculates two nos;1)Avg reward for ad version j and 2)Upper confidence bound for that ad version for each iteration. Variable 'ad' stores the ad version with highest UCB and continuously updates its value whenever upper_bnd > max_upper_bnd. max_upper_bnd value is also updated.

```
#loop through no of rounds
for i in range(0,N):
    max_upper_bnd = 0     # Variable to store value of max UCB and
resets at each round
    ad = 0               #temporary Variable to store ad version
having highest UCB and starts with ad1
    for j in range(0,d):  #Here loop starts with 0 index till index
5
        if (no_selection[j] > 0):
           avg_reward = sum_reward[j]/no_selection[j]
           delta_j = (math.sqrt(3/2) * math.log(i +
1)/no_selection[j])    #Refer Step2 for fig2
           upper_bnd = avg_reward + delta_j            #UCB
calculation for ad version j for iteration i
        else:
           upper_bnd = 1e5                  #High value assigned
during initial rounds
        if upper_bnd > max_upper_bnd:
           max_upper_bnd = upper_bnd
           ad = j
    ad_selected.append(ad)                   #List updates with ad
selected at each round
    no_selection[ad] = no_selection[ad] + 1
    reward = ucb.values[i,ad]                #Picks the user action-
click/no click for selected ad at round i
    sum_reward[ad] = sum_reward[ad] + reward     #list updates to
store rewards for diff add version
    total_reward_ucb = total_reward_ucb + reward #Total
rewards/clicks updates after every round
```

However initially we have no_selection = 0 for any ad version j, so for initial 10 rounds, the strategy will be to select each ad version once,e.g. for round 1 ad0, for round 2 ad1 will be displayed and so on.Then when we get some info about the rewards(clicks) about each ad version during initial rounds the UCB algorithm starts working.The initial if-else clause is used to show ad according to UCB algorithm, only when ad version j is selected at least 1. Upper_bnd is assigned with a very high

value(1e5), so that the sequence of ad selection is maintained During initial rounds(you can try with any ad version, not selected once, during initial 10 rounds, i.e. ad 2 for round 3).

That's it.The UCB algorithm is implemented and now time to see the result!

```
print('total clicks with UCB Algorithm:
{}'.format(total_reward_ucb))

#plot- Distribution of ad_selected
plt.hist(ad_selected)
plt.title('Distribution of ads')
plt.xlabel('ad_no')
plt.ylabel('count')
plt.show()
```

total clicks with UCB Algorithm: 2028.

The result is exciting! With UCB we get the click rate of 20.28%, a significant lift of 1.58 times than random strategy.Also it is evident from the histogram of ad selected that Ad 3('MORE SCI-FI FOR LESS') is the clear winner. That means UCB is able to show the ad version, which has the highest probability of getting clicked.

Let's explore further two vectors ad_selected and sum reward

```
#How many times Ad 3 selected after 1000 rounds till end
 no_displayed = pd.Series(ad_selected[1000:N]).value_counts()
 select_rate = no_displayed/len(ad_selected[1000:N])
```

```
#Let's also look click though rate of diff ad versions
sum_reward.sort(reverse = True)
no_selection.sort(reverse = True)
click_probability = pd.Series(sum_reward)/pd.Series(no_selection)
ucb_output = pd.DataFrame({'Ad Version':no_displayed.index, 'No
Displayed' : no_displayed.values, 'Display Rate'
:select_rate.values, 'Click probability(CTR)':click_probability})
ucb_output
```


So from the output even after 1k rounds algorithm fixes its strategy of displaying the winning ad version.We can see almost 98% of instances Ad 3(no 2 as per python indexing standard), which has the highest probability of click(CTR) and algorithm started exploiting the best ad to get increased no of clicks and subsequent more conversion.This is just a miniature data set. In real life, the data set will be humongous and so the realization of dollar benefit.

Implementation in R

You can find the R Code for UCB implementation here. Logic is the same. Run and see whether the result is matching or not.

Road Ahead:

This is just a baby step towards building an intelligent digital marketing ecosystem and there are immense opportunities to utilize the untapped potential of reinforcement learning. The power of big data and efficient analytics combined inspires a whole new wave of intelligent automation that the human race is just beginning to tap into. Digital marketing is one of the many fields that this wave is disrupting. In my next article, I will introduce another AI-based approach- Thompson Sampling, to see whether it has a better solution for this problem. Until then 'stay hungry stay foolish'.