analyze-ab-test-results-notebook-zh-0 Last Checkpoint: 31分钟前 (unsaved changes)

1 分析A/B测试结果

2 目录

- 简介
- I - 概率
- II - A/B 测试
- III - 回归

2.1 大家好，此次我带了一份自己完成的报告，该报告包括模拟零假设抽样建立正态分布、假设性检验验证我的零假设与备择假设、逻辑回归检测变量间的相关性。

2.2 简介

数学来源于生活，同样也服务于生活，个人非常乐意做到这些用所学的知识服务社会。A/B测试通常是很多领域需要做的，它可以帮我们改善现存的状态，很大程度的帮助公司、机构、政府作出不同的决策。也正因为这样，本人对这个非常的感兴趣

首先，对于这个项目，该项目是电子商务网站运行的 A/B 测试的结果。我的目标是通过这个 notebook 来帮助公司弄清楚他们是否应该使用新的页面，保留旧的页面，或者应该将测试时间延长，之后再做出决定。

In [69]:
```python
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
#We are setting the seed to assure you get the same answers on quizzes as we set up
random.seed(42)
```

In [70]:
```python
df = pd.read_csv('ab_data.csv')
df.head()
```

Out[70]:

	user_id	timestamp	group	landing_page	converted
0	851104	2017-01-21 22:11:48.556739	control	old_page	0
1	804228	2017-01-12 08:01:45.159739	control	old_page	0
2	661590	2017-01-11 16:55:06.154213	treatment	new_page	0
3	853541	2017-01-08 18:28:03.143765	treatment	new_page	0
4	864975	2017-01-21 01:52:26.210827	control	old_page	1

b. 使用下面的单元格来查找数据集中的行数。

In [71]:
```python
# 原始数据集行数
df.shape[0]
```

Out[71]: 294478

In [72]:
```python
# 独立用户数量
len(df.user_id.unique())
```

Out[72]: 290584

c. 数据集中独立用户的数量。

In [73]:
```python
# 用户转化（付费）的比例
from __future__ import division
len(df.query('converted == 1'))/len(df)
```

Out[73]: 0.11965919355605512

d. 用户转化的比例。

In [74]:
```python
df_new = df.query('landing_page == "new_page"')
df_new.query('group != "treatment"').shape[0]
```

Out[74]: 1928

e. `new_page` 与 `treatment` 不一致的次数。理由：因为我们要让新页面(new_page)对应的全部为测试用户(treatment),所以查看数据集中是否有新页面(new_page)对应的控制用户(control),这些字段的存在会干扰分析结果，所以假如有，删掉它们，结果我需要删掉这1928行

In [75]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 294478 entries, 0 to 294477
Data columns (total 5 columns):
user_id         294478 non-null int64
timestamp       294478 non-null object
group           294478 non-null object
landing_page    294478 non-null object
converted       294478 non-null int64
dtypes: int64(2), object(3)
memory usage: 11.2+ MB
```

f. 查看数据集是否存在缺失值，结果无任何的缺失值

`2.`

a. 现在，我将新 dataframe 存储在 **df2** 中。

In [76]:
```python
lists = df_new.query('group != "treatment"').index
df2 = df.drop(lists)
```

In [77]:
```python
df2.shape[0]
```

Out[77]: 290585

重新查看下行数

In [78]:
```python
# Double Check all of the correct rows were removed - this should be 0
df2[((df2['group'] == 'treatment') == (df2['landing_page'] == 'new_page')) == False].shape[0]
```

Out[78]: 0

检查是否有遗漏

In [79]:
```python
df2.user_id.value_counts().count()
```

Out[79]: 290584

`3.`

a. **df2** 中有290584个唯一的 **user_id**,删掉重复id

In [80]:
```python
df2 = df2[~df2.user_id.duplicated()]
```

c. 删除含有重复的 **user_id** 的行。

3 以上对原始数据的初步清洗基本完成

In [81]:
```python
df2.head(3)
```

放上三行数据便于查找变量

`4.`

a. 不管用户收到什么页面，单个用户的转化率约为 `0.119597`

In [82]:
```python
df2.query('converted == 1').shape[0] / df2.shape[0]
```

Out[82]: 0.11959708724499628

b. 假定一个用户处于 `control` 组中，他的转化率约为 `0.06`

In [83]:
```python
df_control = df2.query('group == "control"')
df_control.query('converted == 1').shape[0] / df2.shape[0]
```

Out[83]: 0.06018569501417834

c. 假定一个用户处于 `treatment` 组中，他的转化率约为 `0.0594`

In [84]:
```python
df_treatment = df2.query('group == "treatment"')
df_treatment.query('converted == 1').shape[0] / df2.shape[0]
```

Out[84]: 0.05941139223081794

3.1 疑问

d. 因为网站的新旧页面是随机分发给用户的（为模拟真实场景的随机性），那么一个用户收到新页面的概率是多少？

4 到现在为止，我现在还不能说明究竟是新页面可以让用户更可能的付费，或者是旧页面

5 下面我将开始运用假设性检验来验证新页面是否有效果

`1.` 现在，我需要先做出假设

6 下面我将给出我的原假设与备择假设

$$H_0: p_{old} = p_{new}$$

$$H_1: p_{old} < p_{new}$$

`2.` 假定在零假设中，不管是新页面还是旧页面，p_{new} and p_{old} 都具有等于 **转化** 成功率的"真"成功率，也就是说，p_{new} 与 p_{old} 是相等的。此外，假设它们都等于原数据的转化率，新旧页面都是如此。

我需要执行两次页面之间 **转化** 差异的抽样分布，计算零假设中10000次迭代计算的估计值。

a. 在零假设中，p_{new} 的 **convert rate（转化率）** 为

In [86]:
```python
df2.query('converted == 1').shape[0] / df2.shape[0]
```

Out[86]: 0.11959708724499628

Out[87]: 0.11959708724499628

c. n_{new} 为

In [88]:
```python
df2.query('landing_page == "new_page"').shape[0]
```

Out[88]: 145310

d. n_{old}为

In [89]:
```python
df2.query('landing_page == "old_page"').shape[0]
```

Out[89]: 145274

e. 在零假设中，使用 p_{new} 转化率模拟 n_{new} 交易，并将这些 n_{new} 1's 与 0's 存储在 **new_page_converted** 中。也就是说对新页面用与原数据相等的转化率（概率），来自动生成 0 和 1，0代表未转化，1代表转化

In [90]:
```python
p = df2.query('converted == 1').shape[0] / df2.shape[0]
df2_new = df2.query('landing_page == "new_page"')
new_page_converted = np.random.choice(2,df2_new.shape[0],p=[1-p,p])
new_page_converted
```

Out[90]: array([0, 0, 1, ..., 0, 0, 0])

f. 在零假设中，使用 p_{old} 转化率模拟 n_{old} 交易，并将这些 n_{old} 1's 与 0's 存储在 **old_page_converted** 中。（同理，这次是对旧页面模拟）

In [91]:
```python
df2_old = df2.query('landing_page == "old_page"')
old_page_converted = np.random.choice(2,df2_old.shape[0],p=[1-p,p])
old_page_converted
```

Out[92]: -0.0015989220200194193

h.下面我将用上面的流程，用同样的p值随机抽样，计算10,000个p_{new} - p_{old} 值，并将这 10,000 个值存储在 **p_diffs**列表中

In [93]:
```python
p_diffs = []
p = df2.query('converted == 1').shape[0]/df2.shape[0]
for i in range(10000):
    new_page_converted = np.random.choice(2,df2_new.shape[0],p=[1-p,p])
    old_page_converted = np.random.choice(2,df2_old.shape[0],p=[1-p,p])
    p_diffs.append((new_page_converted == 1).mean() - (old_page_converted == 1).mean())
```

i. 绘制一个 **p_diffs** 直方图。

In [95]:
```python
plt.hist(p_diffs);
```

红线为原数据中p_{old} - p_{new}的值

In [97]:
```python
vlu = (new_page_converted == 1).mean() - (old_page_converted == 1).mean()
p_diff = np.array(p_diffs)
(p_diff>vlu).mean()
```

Out[97]: 0.90110000000000001

得出的p值为 `0.9011`，所以不能拒绝原假设，即 p_{old} = p_{new}，所以转化率在新旧页面无差别

l. 我们也可以使用一个内置程序（built-in）来实现类似的结果。尽管使用内置程序可能更易于编写代码，但上面的内容是对正确思考统计显著性至关重要的思想的一个预排。使用 `n_old` 与 `n_new` 分别引证与旧页面和新页面关联的行数。 下面是计算个页面的转化次数，以及每个页面的访问人数。

In [98]:
```python
df2.head(3)
```

In [99]:
```python
df_old = df2.query('landing_page == "old_page"')
df_new = df2.query('landing_page == "new_page"')
```

In [100]:
```python
import statsmodels.api as sm

convert_old = df_old.query('converted == 1').shape[0]
convert_new = df_new.query('converted == 1').shape[0]
n_old = df_old.shape[0]
n_new = df_new.shape[0]
```

m. 我现在使用 `stats.proportions_ztest` 来计算我的检验统计量与 p-值。

In [101]:
```python
z_score,p_value = sm.stats.proportions_ztest([convert_old,convert_new],[n_old,n_new],alternative='smaller')
```

In [102]:
```python
z_score,p_value
```

Out[102]: (1.3109241984234394, 0.90505831275902449)

In [103]:
```python
from scipy.stats import norm
norm.cdf(z_score)
norm.ppf(1 - 0.05)
```

Out[103]: 1.6448536269514722

刚刚得出的值为p值，p值是在假定零假设为真的情况下（观察到统计量或支持对则假设更多极端）的概率，p值越大越不能拒绝原假设，p值与错误阈值α通常用来判断是否拒绝原假设选择对则假设，在新旧页面中无区别

6.1 III - 回归分析法之一

b. 使用 **statsmodels** 来拟合逻辑回归模型，以查看用户收到的不同页面是否存在显著的转化差异。但是，首先，我需要为这个截距创建一个列， 并为每个用户收到的页面创建一个虚拟变量列。添加一个 **截距** 列，一个 **ab_page** 列，当用户接收 **treatment** 时为1， **control** 时为0。

In [107]:
```python
df2['intercept'] = 1
df2[['control','ab_page']] = pd.get_dummies(df['group'])
```

In [108]:
```python
df2 = df2.drop(['control'],axis = 1)
df2.head()
```

Out[108]:

	user_id	timestamp	group	landing_page	converted	intercept	ab_page
0	851104	2017-01-21 22:11:48.556739	control	old_page	0	1	0
1	804228	2017-01-12 08:01:45.159739	control	old_page	0	1	0
2	661590	2017-01-11 16:55:06.154213	treatment	new_page	0	1	1
3	853541	2017-01-08 18:28:03.143765	treatment	new_page	0	1	1
4	864975	2017-01-21 01:52:26.210827	control	old_page	1	1	0

ab_page下 `0代表control`， `1代表treatment`

c. 使用 **statsmodels** 导入我的回归模型。 实例化该模型，使用 **b.** 中创建的2个列来拟合该模型，用来预测一个用户是否会发生转化。

In [109]:
```python
# 创建分类器并添加变量
logit_mod = sm.Logit(df2['converted'],df2[['intercept','ab_page']])
```

d. 请在下方提供你的模型摘要，并根据需要使用它来回答下面的问题。

In [110]:
```python
# 进行拟合，并生成报告
results = logit_mod.fit()
```

Dep. Variable:	converted	**No. Observations:**	290584
Model:	Logit	**Df Residuals:**	290582
Method:	MLE	**Df Model:**	1
Date:	Mon, 26 Nov 2018	**Pseudo R-squ.:**	8.077e-06
Time:	05:17:45	**Log-Likelihood:**	-1.0639e+05
converged:	True	**LL-Null:**	-1.0639e+05
		LLR p-value:	0.1899

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.9888	0.008	-246.669	0.000	-2.005	-1.973
ab_page	-0.0150	0.011	-1.311	0.190	-0.037	0.007

与ab_page关联的p值为0.19，在 II 中我的零假设与备择假设分别为

$H_0: p_{old} = p_{new}$

$H_1: p_{old} < p_{new}$

而 III 中的零假设与备择假设分别为

$H_0: p_{old} = p_{new}$

$H_1: p_{old} > p_{new}$

两者的备择假设不同，从而方向性不同，所以 p 值出现两极的情况

f. 好的，之前我只是单纯的将转化数据与页面数据放到分类器里面，但并未放入其他的变量（条件），假设用户发生付费与其它的外界因素有关呢？城市、国家、

```python
df_new = countries_df.set_index('user_id').join(df2.set_index('user_id'), how='inner')
```

In [113]:
```python
df_new.head()
```

Out[113]:

	country	timestamp	group	landing_page	converted	intercept	ab_page
user_id							
834778	UK	2017-01-14 23:08:43.304998	control	old_page	0	1	0
928468	US	2017-01-23 14:44:16.387854	treatment	new_page	0	1	1
822059	UK	2017-01-16 14:04:14.719771	treatment	new_page	1	1	1
711597	UK	2017-01-22 03:14:24.763511	control	old_page	0	1	0
710616	UK	2017-01-16 13:14:44.000513	treatment	new_page	0	1	1

这次的数据事先整理过，所以我直接导入了 `country` 变量放到了原数据中

In [117]:
```python
# 查看下该网站注册下的三个国家的注册人数
df_new.country.value_counts()
```

Out[117]:
```
US    203619
UK     72466
CA     14499
Name: country, dtype: int64
```

In [114]:
```python
df_new[['CA','UK','US']] = pd.get_dummies(df_new['country'])
```

继续向之前的那样，建立逻辑回归模型

In [115]:
```python
logit_mod = sm.Logit(df_new['converted'],df_new[['intercept','CA','UK']])
results = logit_mod.fit()
```

Dep. Variable:	converted	**No. Observations:**	290584
Model:	Logit	**Df Residuals:**	290581
Method:	MLE	**Df Model:**	2
Date:	Mon, 26 Nov 2018	**Pseudo R-squ.:**	1.521e-05
Time:	05:38:06	**Log-Likelihood:**	-1.0639e+05
converged:	True	**LL-Null:**	-1.0639e+05
		LLR p-value:	0.1984

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.9967	0.007	-292.314	0.000	-2.010	-1.983
CA	-0.0408	0.027	-1.518	0.129	-0.093	0.012
UK	0.0099	0.013	0.746	0.456	-0.016	0.036

结果出乎意料，CA 与 UK 两者 P 值都大于 alpha 水平，所以不拒绝原假设，差异不具备统计学显著性，对转化无影响

h. 虽然我现在已经查看了国家与页面在转化率上的个体性因素，但现在我要查看页面与国家/地区之间的相互作用，测试其是否会对转化产生重大影响。创建必要的附加列，并拟合一个新的模型。 所以我需要将 `new_page` 与 `CA`， `UK`，分别相乘得到两个新变量放到分类器中

```python
df3['new_CA'] = df3['new_page'] * df3['CA']
df3['new_UK'] = df3['new_page'] * df3['UK']
```

In [121]:
```python
# 分别创建new_page、new_CA、new_UK
df_new[['new_page','old_page']] = pd.get_dummies(df_new['landing_page'])
df_new['new_CA'] = df_new['new_page'] * df_new['CA']
df_new['new_UK'] = df_new['new_page'] * df_new['UK']
```

834778	UK	2017-01-14 23:08:43.304998	control	old_page	0	1	0	0	1	0	0	1	0	0
928468	US	2017-01-23 14:44:16.387854	treatment	new_page	0	1	1	0	0	1	1	0	0	0
822059	UK	2017-01-16 14:04:14.719771	treatment	new_page	1	1	1	0	1	0	1	0	0	1
711597	UK	2017-01-22 03:14:24.763511	control	old_page	0	1	0	0	1	0	0	1	0	0
710616	UK	2017-01-16 13:14:44.000513	treatment	new_page	0	1	1	0	1	0	1	0	0	1

```
Iterations 6
```

Out[123]:

Logit Regression Results

Dep. Variable:	converted	**No. Observations:**	290584
Model:	Logit	**Df Residuals:**	290578
Method:	MLE	**Df Model:**	5
Date:	Mon, 26 Nov 2018	**Pseudo R-squ.:**	3.482e-05
Time:	05:49:08	**Log-Likelihood:**	-1.0639e+05
converged:	True	**LL-Null:**	-1.0639e+05
		LLR p-value:	0.1920

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.9865	0.010	-206.344	0.000	-2.005	-1.968
new_CA	-0.0469	0.054	-0.872	0.383	-0.152	0.059
new_UK	0.0314	0.027	1.181	0.238	-0.021	0.084
ab_page	-0.0206	0.014	-1.505	0.132	-0.047	0.006
CA	-0.0175	0.038	-0.465	0.642	-0.091	0.056
UK	-0.0057	0.019	-0.306	0.760	-0.043	0.031

根据所得结果 p 值大于 alpha水平，不拒绝原假设，页面与国家/地区之间相互作用不显著

In [1]:
```python
import base64
base64.b64decode('sOQzWGgdOEE=')
```

Out[1]: '\xb3D3Xh\x1d8A'

In [ ]: