دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره 2 بخش 2
گروه ...
درس یادگیری تعاملی
پاییز 1400

نام و نام خانوادگی	مهسا تاجیک
شماره دانشجویی	810198126

فهرست

چکیده

در این تمرین به بررسی الگوریتم MAB و مزایا و معایب آن در مقایسه با روش A/B testing و یا گروهی کارشناس میپردازیم و به طور خاص به پیاده سازی الگوریتم epsilon greedy با اپسیلون متغیر برای مدل سازی یک مساله می پردازیم.

سوال الف – سوال تئوری

از نظر بازاریابی، راه‌حل MAB ورژنی هوشمندتر و پیچیده‌تر از A/B testing است که از الگوریتم‌های یادگیری ماشین برای تخصیص پویای ترافیک به تغییراتی که عملکرد خوبی دارند، استفاده می‌کند، در حالی که ترافیک کمتری را به تغییراتی که عملکرد ضعیفی دارند اختصاص می‌دهد. در تئوری، MAB باید نتایج سریع‌تری تولید کند، زیرا نیازی به انتظار برای یک تغییر برنده نیست .اصطلاح multi armed bandit از یک آزمایش فرضی می آید که در آن فرد باید بین چندین عمل (یعنی ماشین های بازی، "راهزنان یک دست") که هر کدام میزان پرداختی ناشناخته دارند، یکی را انتخاب کند. هدف تعیین بهترین یا سودآورترین نتیجه از طریق یک سری انتخاب است. در ابتدای آزمایش، زمانی که شانس ها و پرداخت ها ناشناخته هستند، قمارباز باید تعیین کند که کدام ماشین را به کدام ترتیب و چند بار بکشد. این مسئله multi armed bandit است.

در این مسئله هر کدام از طرح ها یک اکشن است که در هر بار استفاده یکی از آن ها را انتخاب میکند و براساس انتخابی که کاربر انجام داده سود دریافتی app مشخص میشود. پس یک مسئله armed 2 داریم. و این اکشن ها براساس الگوریتم انتخابی با یک سیاستی که کاربر دارد انتخاب میشوند و به تدریج به سمت اکشن با سود بیشتر همگرا میشود.

در روش A/B testing دو نسخه مختلف از یک وب‌سایت یا اپلیکیشن با یکدیگر مقایسه میشوند تا مشخص شود که کدامیک بهتر است. تست A/B روشی است که ضمن مقایسه ی حالات مختلف، بهترین استراتژی تبلیغاتی و بازاریابی آنلاین برای کسب‌وکار را فراهم می کند. در این روش، دو نسخه از یک صفحه وب سایت ایجاد می‌شود که در یک نسخه عنصر مورد آزمایش به شکل A و در نسخه‌ی دیگر به شکل B قرار داده شده است. سپس این صفحات، در یک بازه زمانی معین (معمولاً بین 3 تا 10 روز) به صورت تصادفی به کاربران نمایش داده می‌شود (به این معنی که یک کاربر ممکن است صفحه A و کاربر دیگری صفحه B را به صورت همزمان مشاهده کنند). در نهایت، با مقایسه‌ی عملکرد کلی هر دو نسخه می‌توان نسخه نهایی وب‌سایت را به گونه‌ای طراحی کرد تا بالاترین نرخ تبدیل کاربر به مشتری را دریافت نمایید.

برای تصمیم گیری در مورد استفاده از multi armed bandit یا A/B testing باید exploration-exploitation tradeoff را سنجید.

با A/B testing تعداد محدودی exploration انجام می دهیم که در این exploration ها ترافیک یکسانی به دو ورژن A و B اختصاص می دهیم.بعد از اینکه برنده مشخص شد و اعلام شد همه ی یوزرها به ورژن برنده روی می آورند و یک دوره ی طولانی exploitation داریم. یکی از مشکلات این رویکرد این

است که منابع برای تغییرات ورژن بازنده هدر می رود زمانیکه اطلاعات جمع می کنیم و یاد می گیریم کدامیک برنده است.

با A/B testing تست ها adaptive هستند و دوره های exploration-exploitation را همزمان شامل می شوند و در این روش بجای اینکه مجبور باشیم منتظر بمانیم تا در پایان آزمایش برنده اعلام شود، ترافیک را به تدریج به سمت برنده سوق می دهند. این فرایند سریع تر و کارامدتر است زیرا زمان کمتری برای ارسال ترافیک به inferior variations صرف می شود.

یکی از معایب اصلی MAB پیچیدگی محاسباتی آن است و اجرای آن دشوارتر و نیازمند منابع است. برخی موقعیت های شناخته شده وجود دارد که در آن MAB بهتر کار می کند:

Headlines & short-term campaigns:

هزینه ی منتظر ماندن برای نتایج تست در A/B testing ، الگوریتم های MAB را به گزینه ی بهتری برای محتوای کوتاه مدت مثل تست تیتر برای مقالات جدید تبدیل می کند.

Long-term dynamic changes:

هنگامی که آیتم مورد آزمایش به اندازه کافی تغییر می کند تا نتایج یک A/B testing را در طول زمان باطل کند، MAB جایگزینی برای آزمایش مجدد با exploring مداوم ارائه می دهد.

Targeting:

نمونه ی دیگری از استفاده ی طولانی مدت از الگوریتم های MAB است. اگر تایپ خاصی از کاربران رایج تر باشند نسبت به بقیه، MAB میتواند قوانین targeting آموزش دیده را سریعتر برای کاربرانی که رایج تر هستند اعمال کند و همزمان به آزمایش روی کاربران دیگر ادامه می دهد.

Automation for scale:

اگر چندین مولفه برای بهینه سازی مداوم داریم ، رویکرد MAB به ما چارچوبی می دهد تا فرایند بهینه سازی را برای مسائل کم ریسک تا حدی خودکار کند که تجزیه و تحلیل جداگانه آن می تواند بسیار پرهزینه باشد.

سوال ب - سوال تئوری

از آن جایی که آدم های مختلف، در استیت های مختلفی قرار دارند و دریافت های ذهنی متفاوتی دارند بنابراین تابع یوتیلی ای که هر کارشناس در ذهن دارد ممکن است متفاوت باشد و همچنین ممکن است بایاس های زیادی را وارد تصمیم گیری کند و در انتخاب تصمیم بهینه تاثیرگذار باشند در حالیکه در الگوریتم MAB این بایاس ها وجود ندارند.

سوال ج – سوال پیاده سازی

هدف سوال

هدف این قسمت از سوال پیاده سازی یک ایجنت اپسیلون گریدی با اپسیلون متغیر و بررسی میزان regret در 1000 ترایال و 10 اجرای مختلف است.

توضیح پیاده سازی

در این سوال با یک مسئله multi armed bandit مواجه هستیم که آرم ها، طرح های پیشنهادی هستند که هر کدام 2 بسته ی مختلف ارائه می دهند و میتوانیم بگوییم 4 آرم داریم که انتخاب بین طرح ها براساس سیاست اپسیلون گریدی با اپسیلون متغیر از کاملا رندوم به سمت گریدی انجام میشود و انتخاب هر کدام از بسته های یک طرح طبق رابطه ی گفته شده در صورت سوال است که یک عدد رندوم ایجاد می کنیم اگر از 0.33 کمتر بود، بسته طبق رابطه ی اول و اگر از 0.33 بیشتر بود، بسته از رابطه ی دوم انتخاب میشود و همچنین یک کلاس BetaReaward ایجاد می کنیم که از کلاس RewardBase ارث بری می کند و از این کلاس در انتخاب نوع بسته ها استفاده میکنیم .

```
def SelectPack(volumes, prices):
  random_number = np.random.random_sample()


  if random_number >= 0.33:
    x = BetaReward(2,5).get_reward()
    volumes_beta = []
    for i in range(len(volumes)):
      volumes_beta.append(volumes[i] * x)
    selected_pack = np.argmax(volumes_beta)
  else:
    x = BetaReward(3,2.5).get_reward()
    volumes_prices_beta = np.divide(volumes, prices) * x
    selected_pack = np.argmin(volumes_prices_beta)
  return selected_pack
```

یک کلاس Pack تعریف میکنیم که یک نمونه از یک بسته را با ویژگی های حجم و قیمت ایجاد می کند:

```
class Pack:
    def __init__(self, p, v):

        self.price = p
        self.volume = v

    def get_price(self):
      return self.price
    def get_volume(self):
      return self.volume
```

بعد از انتخاب یک آرم یعنی یک طرح و یکی از بسته های پیشنهادی، این اکشن به محیط اعمال می شود و پاداش درنظر گرفته شده برای آن را دریافت میکند.

محاسبه پاداش به شکل زیر است که یک لیست دوتایی است و هر عنصر آن خودش یک لیست دوتای است.

```
rewards = []
#plan1
pack1 = Pack(21, 6)
pack2 = Pack(35, 12)
#plan2
pack3 = Pack(12, 3)
pack4 = Pack(22, 8)

prices = np.array([[pack1.get_price(), pack2.get_price()], [pack3.get_price(), pack4.get_price()]])
volumes = np.array([[pack1.get_volume(), pack2.get_volume()], [pack3.get_volume(), pack4.get_volume()]])
remain_volume = 1-GaussianReward(0.61,0.05).get_reward()
rewards = prices + (prices * remain_volume)
```

برای اجرای الگوریتم و محاسبه ی regret نیاز داریم که میانگین پاداش بهترین اکشن و همچنین میانگین پاداش ها را در تمام ترایال ها داشته باشیم. بنابراین ابتدا در 10000 ترایال هر دو اکشن را اجرا کرده و میانگین پاداش ها را محاسبه می کنیم و اکشن بهینه را پیدا میکنیم. سپس در 10 اجرا و هر بار برای 1000 ترایال الگوریتم را اجرا کرده و میانگین پاداش در آن لحظه را حساب کرده و اختلاف آن را از میانگین پاداش عمل بهینه حساب میکنیم و با مقدار regret قبل جمع میکنیم.

کد مربوط به پیدا کردن عمل بهینه :

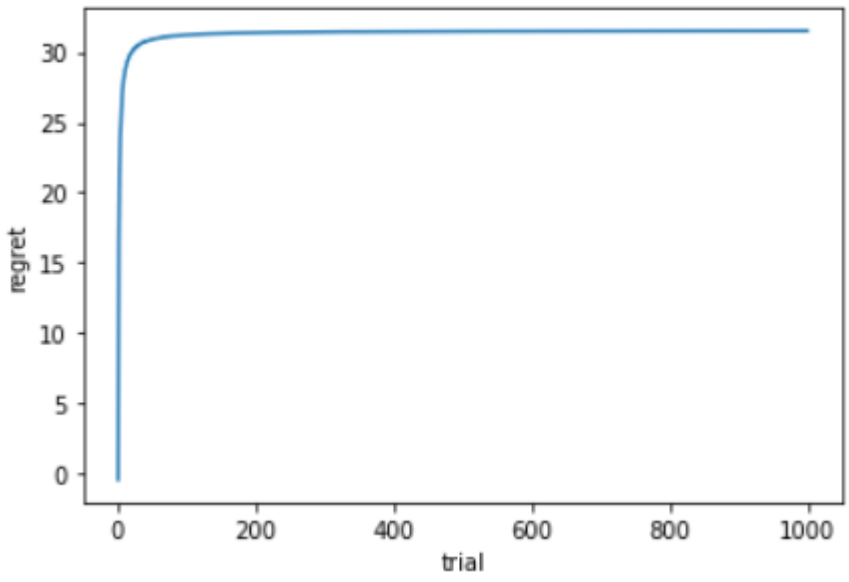
```
r1 = r2 = 0
count1 = count2 = 0
for k in range(1,10000):
    observation, reward, done, info, action, selected_pack = agent1.take_action()
    if (action == 0) and (selected_pack == 0):
        r1 += reward[0]
        count1 +=1
    elif (action == 0) and (selected_pack == 1):
        r1 += reward[1]
        count1 +=1
    elif (action == 1) and (selected_pack == 0):
        r2 += reward[0]
        count2 +=1
    else:
        r2 += reward[1]
        count2 +=1

average_action1 = r1/count1
if count2 == 0:
    average_action2 = 0
else:
    average_action2 = r2/count2

if average_action1 > average_action2:
    optimal_action = 0
    optimal_average = average_action1
else:
    optimal_action = 1
    optimal_average = average_action2
```

محاسبه regret :

```
for i in range(10):
  agent1.epsilon= 0.1
  count = [0 for col in range(2)]
  values = [0.0 for col in range(2)]
  rewards = []
  selected_action = []
  REGRET = []
  for j in range(1,1001):
    if j % 100 == 0:
      # print(j)
      agent1.epsilon += 0.00001
      # print(agent1.epsilon)
    observation, reward, done, info, action, selected_pack = agent1.take_action()

    count[action] +=1
    if (action == 0) and (selected_pack == 0):
      values[action] = reward[0] * (1/count[action]) + ((count[action]-1)/count[action])*values[action]
      rewards.append(reward[0])
    elif (action == 0) and (selected_pack == 1):
      values[action] = reward[1] * (1/count[action]) + ((count[action]-1)/count[action])*values[action]
      rewards.append(reward[1])

    elif (action == 1) and (selected_pack == 0):
      values[action] = reward[0] * (1/count[action]) + ((count[action]-1)/count[action])*values[action]
      rewards.append(reward[0])
    else:
      values[action] = reward[1] * (1/count[action]) + ((count[action]-1)/count[action])*values[action]
      rewards.append(reward[1])

    selected_action.append(action)

  for k in range(len(selected_action)):
    regret = (optimal_average - rewards[k]/k)
    REGRET.append(regret)

  plt.plot(REGRET)
  plt.xlabel("trial")
  plt.ylabel("regret")
  plt.show()
```

نتایج

نتایج به ازای تغییر اپسیلون از مقدار 0.1 و تغییر آن در هر 100 ترایال به اندازه ی 0.05، برای 1000 ترایال و بعد از 10 اجرا بصورت زیراست:



روند اجرای کد پیاده‌سازی

تمامی قسمت ها در یک نوت بوک آمده است و کافی است که در دایرکتوری پکیج آمالرن اجرا شود.