دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره ۲

گروه ۲

درس یادگیری تعاملی

پاییز ۱۴۰۰

نام و نام خانوادگی	سپیده فاطمی خوراسگانی
شماره دانشجویی	۸۱۰۸۹۶۰۵۹

فهرست

چکیده

در این تمرین مسائل n armed bandit را مطرح می‌کنیم و برای یافتن پالیسی بهینه و پیدا کردن بهترین action الگوریتم هایی را ارائه می‌دهیم. مسائل را مدل سازی کرده و با استفاده از روش‌های گفته شده آن‌ها را حل می کنیم.

در این تمرین محیط ما single state است و به تعداد arm ها اکشن متفاوت می‌توانیم انجام دهیم که هر اکشن reward مخصوص به خود را دریافت می‌کند و هر agent در محیط دارای utility یا برداشت ذهنی متفاوتی می باشد.

سوال ۱ -

هدف سوال

هدف این سؤال حل یک مسأله n_armed_bandit می‌باشد که باید با استفاده از الگوریتم های مختلف بهترین انتخاب را به دست آوریم. در این سؤال این الگوریتم ها و هایپر پارامتر های آن‌ها را بررسی و مقایسه می کنیم. سپس با استفاده از درک و دانشی که به دست آوریم به سؤالات پاسخ می دهیم.

قسمت اول

الف) الگوریتم اپسیلون گریدی

ابتدا دو کلاس Reward را تعریف می‌کنیم که inherit شده از کلاس RewardBase می باشند.

دارای یک متد get_reward می‌باشند که abstract method است و در کلاس‌های بچه

Rewards

```
class GaussianReward(RewardBase):
    def __init__(self, mean, std):
        super(GaussianReward, self).__init__()
        self.mean = mean
        self.std = std

    def get_reward(self):
        return np.random.normal(loc=self.mean, scale=self.std)
```

```
class BernoulliReward(RewardBase):
    def __init__(self, bandit_prob, reward, punishment):
        super(RewardBase, self).__init__()
        self.bandit_prob = bandit_prob
        self.reward = reward
        self.punishment = punishment

    def get_reward(self):
        r = np.random.binomial(1, self.bandit_prob)
        return self.reward if r == 1 else self.punishment
```

حتما باید پیاده‌سازی شود. در enviroment_base در متد step اجرای action بر روی محیط صورت می‌گیرد و در این متد است که calculate_reward که متد abstract آن است صدا زده می‌شود در calculate_reward هر محیطی که از environment_base inherit شده است، تابع مخصوص به خود را اجرا می کند. در calculate_reward تابع get_reward صدا زده می‌شود و ریوارد هر action را دریافت می کند.

سپس Agent ها را پیاده‌سازی می کنیم.

Epsilon Greedy Agent:

در زیر متد select_action مربوط به ایجنت epsilon greedy را مشاهده می کنیم. این ایجنت به این صورت عمل می‌کند که با احتمال بالاتر ۱-ε اکشنی را انتخاب می‌کند که تا قبل از این مرحله q value بیشتری داشته است. q value ها برابر است با E[r] ها. یعنی میانگین ریوارد ها را به عنوان q value پس از انتخاب هر اکشن update می کنیم.

```
def select_action(self):
    available_actions = self.available_actions
    eps = self.epsilon

    best_action = np.argmax(self.q_values)
    random_action = np.random.choice(available_actions)

    prob_best_action = 1 - eps + (eps / available_actions)
    prob_random_action = eps / available_actions
    sum = prob_best_action + prob_random_action #normalize
    selected_action = np.random.choice([best_action, random_action], p=[prob_best_action/sum, prob_random_actior
    return selected_action
```

پس از پیدا کردن بهترین اکشن، ابتدا utility یعنی ذهنیت ما از پاداش دریافتی را محاسبه می‌کند و سپس پاداش دریافت شده (utility) را به ماتریسی که در آن مجموع ریوارد های دریافت شده تا آخرین لحظه را در اختیار داریم اضافه می کنیم. (rewards[])

تعداد پاداش هایی که تا آخرین لحظه دریافت کرده‌ایم را در counts ذخیره می کنیم.

```
def update(self, action, r):
    r = self.utility(r)
    self.rewards[action] += r
    self.counts[action] += 1
    self.q_values[action] = self.rewards[action] / self.counts[action]
    if self.status == "adaptive":
        self.epsilon = self.epsilon/2
```

ε ثابت یا متغیر بودن را در متد update رسیدگی می کنیم. اگر ε متغیر باشد پس از هر trial مقدار آن را نصف می کند. در غیر این صورت با همان ε ثابت ادامه می دهد.

ب) الگوریتم UCB

در این الگوریتم ابتدا upper confidence bound را برای هر اکشن به دست می‌آورد و سپس اکشنی را که بزرگ‌ترین UCB را دارد انتخاب می کند.

در این الگوریتم c ضریب exploration می باشد.

```
def calculate_ucb(self, action):
    if self.counts[action] == 0:
        return np.inf
    else:
        return self.q_value[action] + self.c * np.sqrt((np.log(sum(self.counts))) / self.counts[action])

def select_action(self):
    available_actions = np.arange(1, self.available_actions)
    ucb = [self.calculate_ucb(action) for action in available_actions]
    return np.argmax(ucb)
```

ج) مقایسه تغییرات $\alpha\ \beta\ \lambda$

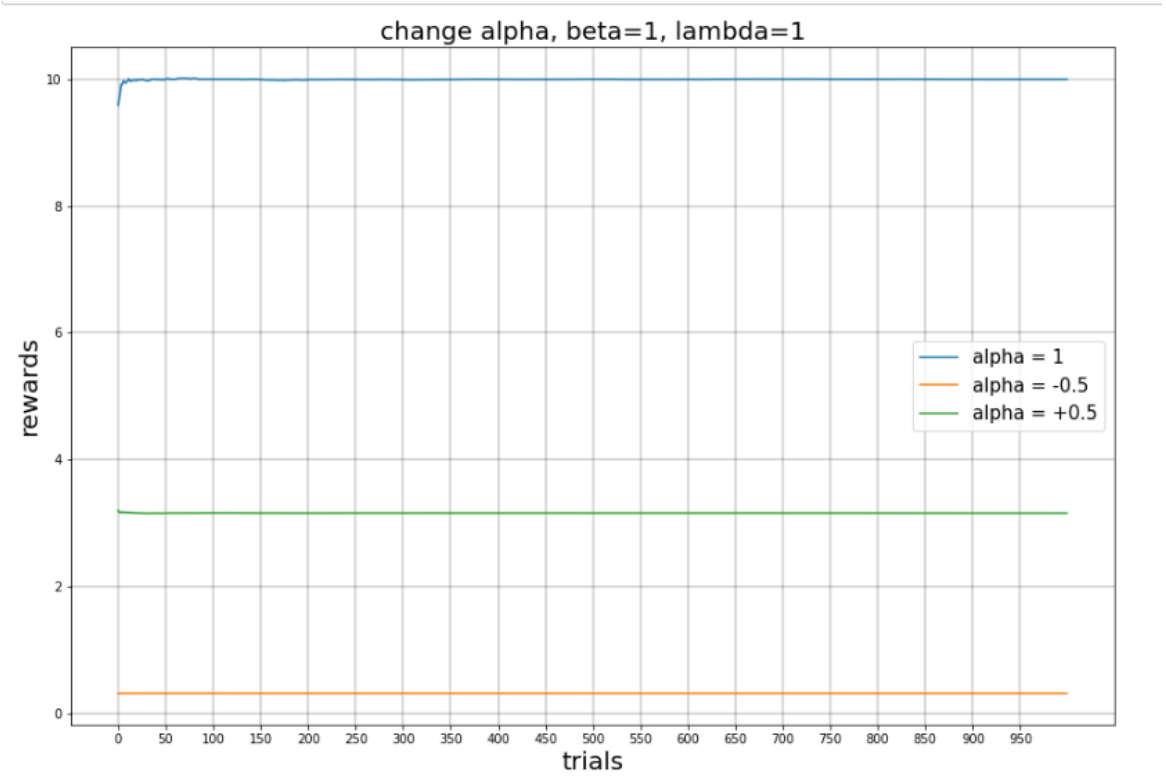
- تغییر alpha

در این نمودار به دلیل متفاوت بودن اسکیل داده‌ها نمودار ها به صورت خطی بـــه نظـــر می‌رسند ولی اگه آن‌ها را به صورت تکی در اسکیل خودشان رسم کنیم همانند نمودار ucb صفحه قبل می شود.

برای α =1 میانگین ریوارد های دریافت شده در مجموع بیشـــتر اســـت. در این مـــورد یعنی α =1 ایجنت پاداش مثبت را به همان میزان واقعی می بیند. یعنی ذهنتیش با واقعیت مطابق است.

برای α=0.5 هم به همین صورت است منتها به دلیل اینکه پاداش مثبت را بـــه صـــورت مجذور آن می‌بیند پس میانگین ریوارد ها تقریباً نصف شده است.

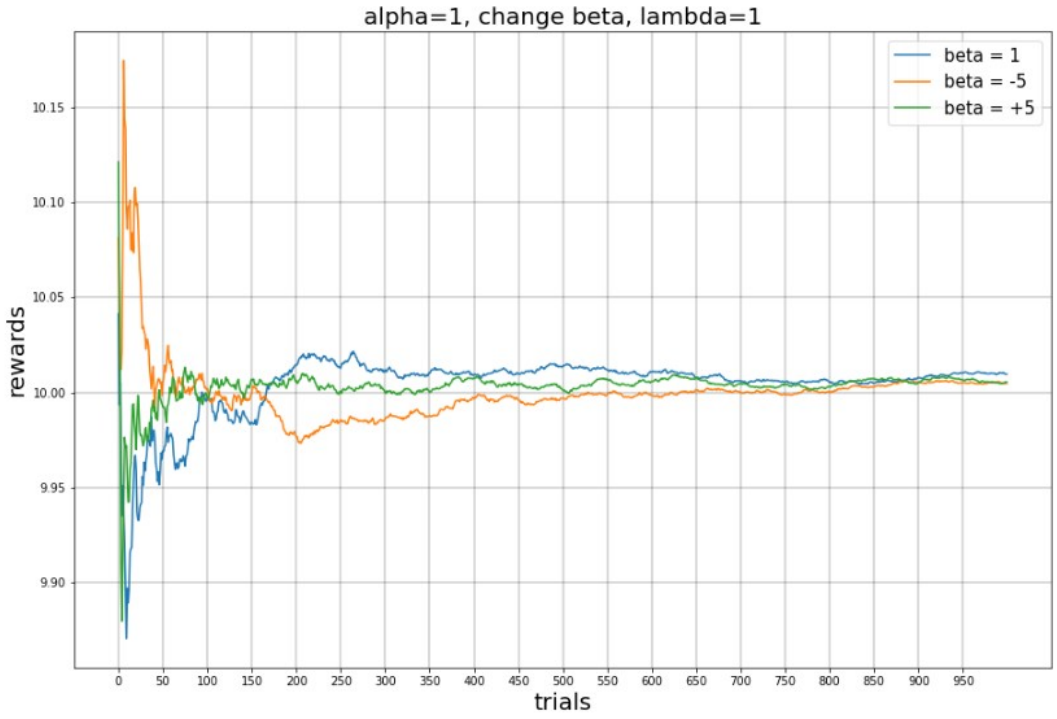
برای α= -0.5 در این مورد ایجنت پاداش مثبت را به صورت $r^{(-0.5)}=(\frac{1}{r})^{0.5}$ می‌بیند یعنی از نظر ذهنی و دریافتی تمایل کمتری دارد که پاداش مثبت دریافت کنـــد. بنـــابر این میانگین پاداش ها در این ایجنت تقریباً برابر با صفر است.



- **تغییر** beta

beta توان r در پاداش های منفی می‌باشد هرچه این مقدار بزرگ‌تر باشد ایجنت پاداش منفی را خیلی منفی تر می‌بیند. در صورتی که توان منفی باشد پـاداش منفی را کم‌تـر منفی می‌بیند.

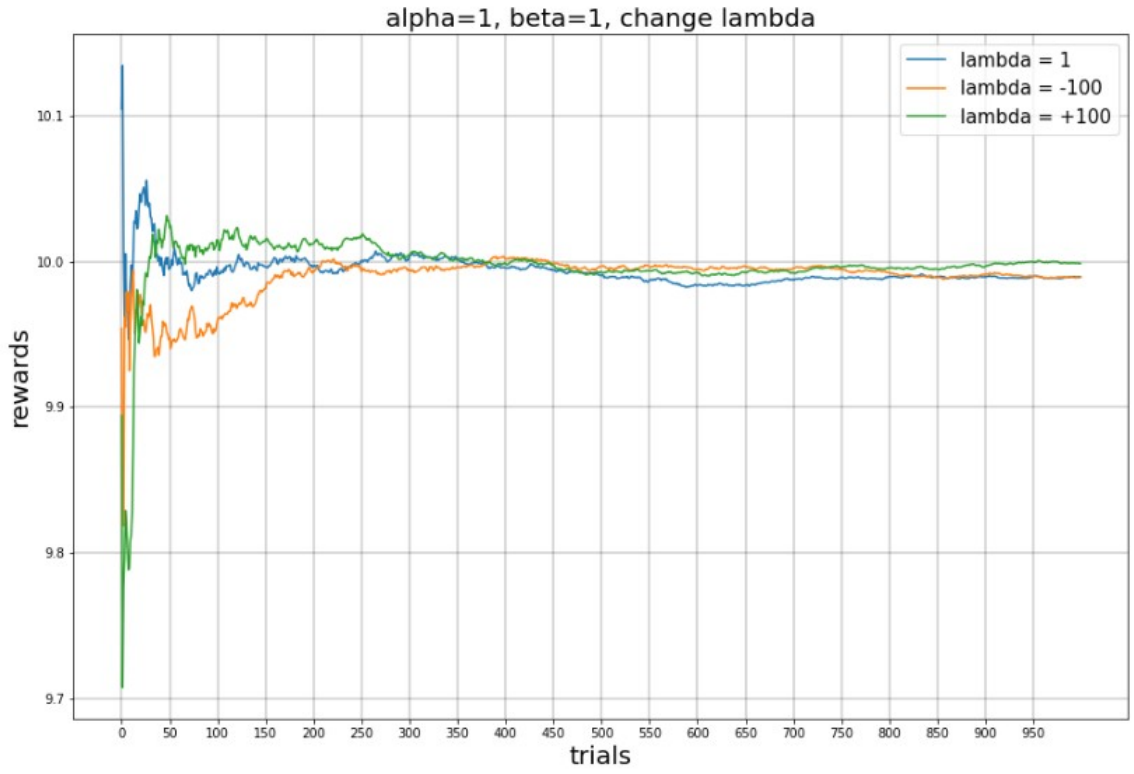
همان‌طور که مشاهده می‌شود برای بتا منفی شیب نمودار به صورت نـزولی می‌باشـد و دلیل آن هم این است که پاداش منفی را تقریباً صفر می‌بیند بنابر این ایجنت از انتخاب آن اکشن اجتناب نمی کند.



- **تغییر** lambda

لاندا ضریب پاداش منفی می‌باشد. اگر لاندا را منفی قرار دهیم به این معنی است که ایجنت از پاداش منفی گرفتن لذت می برد.

در نمودار لاندا منفی ابتدا شیب کاهش می‌یابد به دلیل اینکه همه پاداش ها را مثبت می‌بیند و در نهایت اکشنی را که ucb **بالاتری دارد انتخاب می کند**



قسمت دوم

الف) تأثیر ε بر سرعت یادگیری و تفاوت ε ثابت و متغیر

بر اساس کدی که زده‌ام در این الگوریتم اکشنی که q value بالا تری داشته است با احتمال $(1-epsilon)$ بالاتر از اکشن رندوم انتخاب می شود.

بنابر این هرچه ε بزرگ‌تر باشد به دلیل اینکه $(1-epsilon)$ به صفر نزدیک‌تر می شود، اکشن ها عملا به صورت رندوم انتخاب می شوند. که در نتیجه آن ایجنت به سمت exploration می‌رود و اکشن ها را رندوم تر انتخاب می‌کند و به صورت greedy اکشنی را که تا آن مرحله بهتر بوده انتخاب نمی کند.

هرچه ε کوچک تر باشد به دلیل اینکه $(1-epsilon)$ به یک نزدیک‌تر می شود، اکشنی که q value بالا تری دارد حریصانه تر انتخاب می شود. که در نتیجه آن الگوریتم به صورت greedy عمل کرده و ایجنت به سمت exploitation می‌رود یعنی اکشنی را که تا الان خوب بوده است را تکرار می‌کند و اهمیت کمتری به اکشن هایی که تا به حال انجام نداده می دهد(حتا اگر اکشن بهتری هم وجود داشته باشد ایجنت از آن اطلاعاتی ندارد چون تا به حال آن را انتخاب نکرده است.)

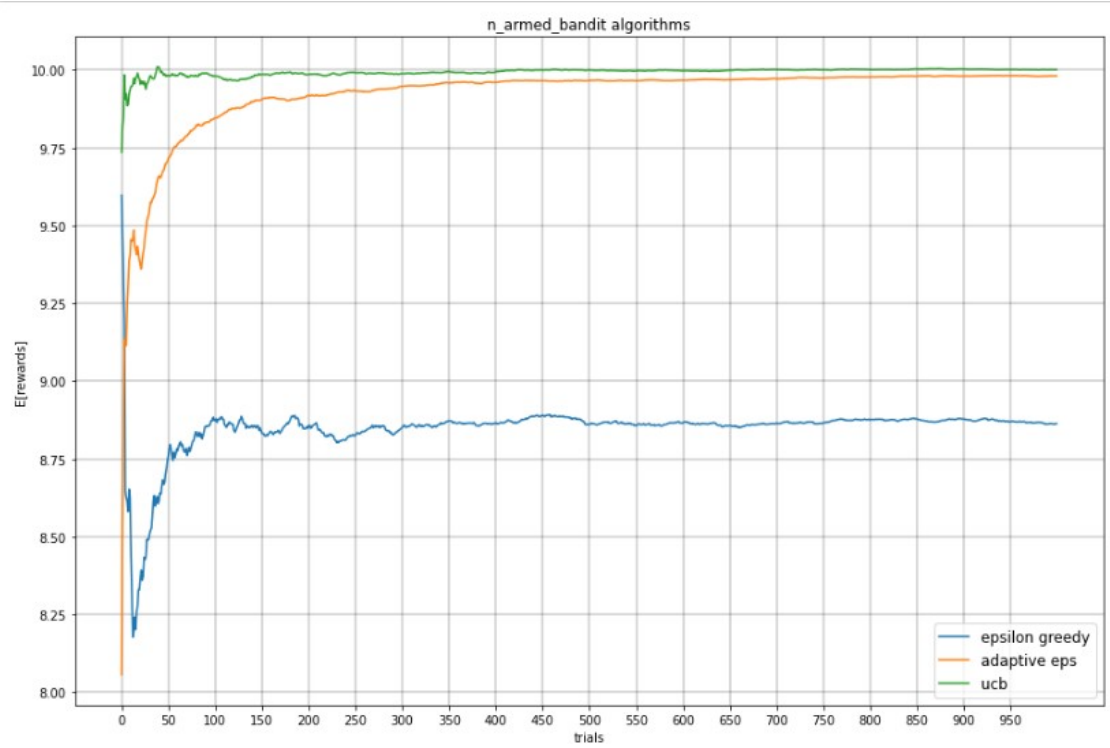
ε متغیر: در این الگوریتم در حالتی که اپسیلون متغیر باشد باید کم کم از یک به صفر تغییر کند. یعنی در مراحل اولیه که هنوز اطلاعاتی از پاداش ها و محیط ندارد explore کند و در ادامه پس از کسب اطلاعات کافی به exploit روی بیاورد.

همان‌طور که مشاهده کردیم الگوریتم ε متغیر بر روی نموداری که از E[R] ها رسم کردیم بالاتر از ε ثابت می باشد. ε متغیر جواب بهتری به ما می‌دهد به دلیل که با احتمال بالا تری اکشن بهینه را پیدا می کند.

برای اینکه الگوریتم ε متغیر به جواب بهینه همگرا شود باید ε را کم کم تغییر دهیم و این کار را به صورت ناگهانی انجام ندهیم. به دلیل اینکه ممکن است بدون اینکه یک بار اکشن بهینه را انتخاب کرده باشد شروع به exploit کردن بکند.

راه حل دیگر این است که تعداد trial ها را زیاد بدهیم در این صورت احتمال اینکه اکشنی هیچ وقت انتخاب نشود بسیار کم می شود.

نمودار زرد رنگ ε متغیر و نمودار آبی ε ثابت را نشان می دهد.



در ε ثابت نمودار دیرتر همگرا می‌شود و به صورت میانگین پاداش کمتری هم دارد.

ε متغیر سریع‌تر همگرا می‌شود و به تعداد trial کمتری نیاز دارد در نتیجه الگوریتم بهینه‌تری می باشد.

ب) مقایسه گرادیان و اپسیلون گریدی

برای مقایسه این بخش کد مربوط به الگوریتم گرادیان را زدم که به شهود بهتری از مطلب برسم.

در ادامه شبه کد مربوط به آن را قرار می‌دهم و توضیحاتی می دهم.

در این الگوریتم پالیسی را بر اساس رابطه غیر خطی به دست می آوریم. بنابر این احتمال انتخاب شدن هر اکشن را داریم و در select_action یک اکشن را بر اساس احتمال هایشان انتخاب می کنیم.

در هر trial یک action را انتخاب می‌کنیم و مقادیر H و $\overline{R}$ را اپدیت می‌کنیم و که در نتیجه آن پالیسی یعنی احتمال انتخاب هر اکشن هم تغییر می کند.

نکته مهم این الگوریتم غیر خطی بودن پالیسی می باشد.

این الگوریتم به این صورت عمل می‌کند که برای اکشن انتخاب شده اگر پاداش دریافت شده اش از میانگین پاداش های دریافتی قبلی بیشتر باشد یعنی $R_t>\overline{R}$ ، $H(A_t)$ آن را به اندازه فاصله احتمال انتخاب اکشن A_t با سیاست گریدی افزایش می دهد. یعنی هرچه احتمال انتخاب اکشن بالا تر باشد، تغییرات کم‌تر باشد و هر چه احتمال انتخاب اکشن کم‌تر باشد و هرچه احتمال انتخاب اکشن کم‌تر باشد یعنی فاصله آن با سیاست گریدی بیشتر باشد، تغییرات H متناسب با $R_t-\overline{R_t}$ بیشتر است.

بقیه اکشن ها به جز اکشن انتخاب شده، با ضریبی متناسب با $R_t-\overline{R_t}$ و احتمـــال آن‌هـــا کاهش می یابد. هر چه احتمال اکشنی بیشتر بوده به مقدار بیشتری کم می شود.

برای اکشن انتخاب شده اگر پاداش دریافت شده اش از میانگین پاداش های دریـــافتی قبلی کم تر باشد یعنی $R_t<\overline{R}$ ، $H(A_t)$ آن را به اندازه فاصله احتمال انتخاب اکشـــن A_t بـــا سیاست گریدی کاهش می دهد. اکشنی که احتمال بالاتری دارد کم‌تر کاهش میابد. در مقابـــل اکشن های رقیب را متناسب با احتمالشان بالا می برد.

$Initialization\, H(a)=0\,;\forall(a), learning\, rate\,, t=0\,, \bar{r}=0$

$Do\, forever$:

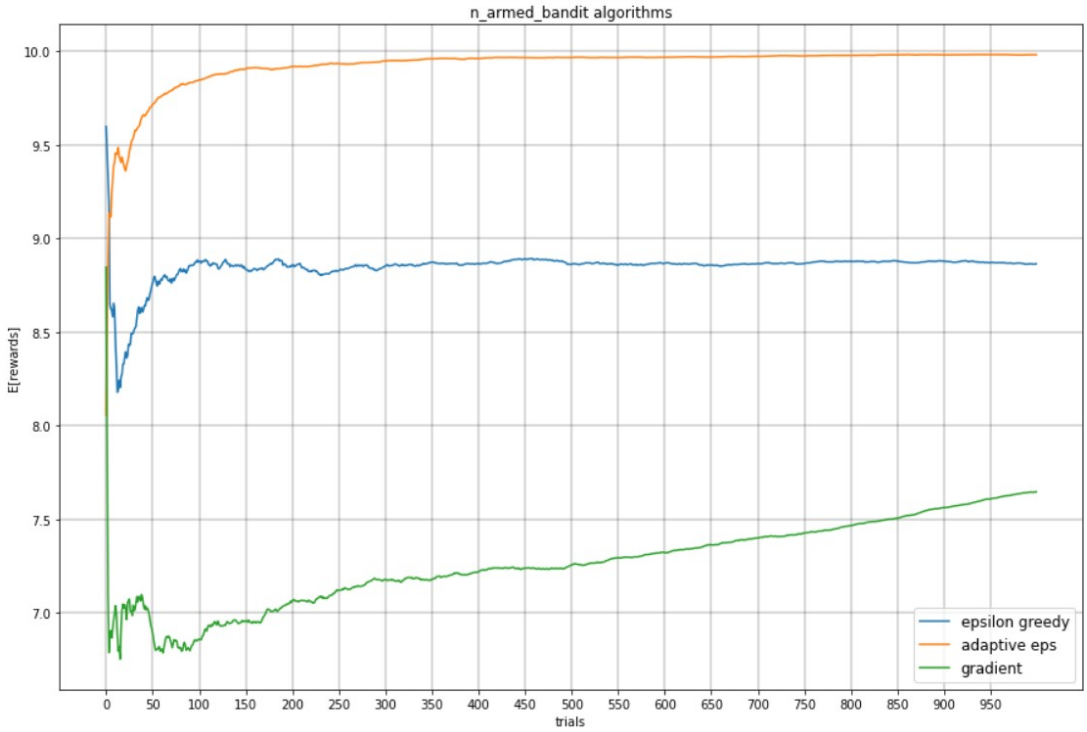
$Select\ A_t\, based\, on\, policy\ p(a)=\frac{e^{H(a)}}{\sum_b e^{H(b)}}$

$Do\ A_t\, get\ R_t$

$Update\ H(A_t)\leftarrow H(A_t)+\alpha(R_t-\overline{R})(1-\pi(A_t))$

$Update\ H(a)\leftarrow H(a)-\alpha(R_t-\overline{R})(\pi(A_t))$

$Update\ \overline{R}\leftarrow \frac{\overline{R_{(t-1)}}+R_t}{t}=\frac{R_1+R_2+...+R_t}{t}$



در مقایسه با اپسیلون گریدی متوجه می‌شویم که مجموع ریوارد های آن سیر صعودی دارد و دیر تر همگرا می‌شود و کندتر است.

با توجه به توضیحات داده شده به دلیل اینکه پس از اعمال هر اکشن احتمالات اکشن های دیگر هم تغییر می‌کند، به همین دلیل این الگوریتم دیرتر همگرا می‌شود ولی در trialهای بالا حتما به بهترین جواب همگرا می شود.

در این روش بر عکس اپسیلون گریدی که ممکن است در صورت انتخاب اشتباه اپسیلون جواب بهینه را پیدا نکنیم و به همه اکشن ها فرصت ندهیم، در این روش اگر اکشنی انتخاب نشده باشد احتمال آن را بالا می‌برد که انتخاب شود. بنابر این همه اکشن ها حتماً بررسی می شوند.

ج)

۱-

در این سؤال ایجنت می‌تواند utilityدیگر بازیکنان را ببیند ولی از خود پاداش دریافتی آن‌ها اطلاعی ندارد. با توجه به اینکه utility همه بازیکنان با هم برابر است، بازیکن ما با مشاهده رفتار دیگر بازیکنان می‌تواند کاملاً بر اساس تصمیم آن‌ها پیش برود. اگر utility ها متفاوت بود انجام این کار ممکن نبود.

بازیکن در ابتدا نمی‌تواند بر اساس utility های دریافتی متوجه شود که در مجموع چه کسی بهتر بازی می‌کند و انتخاب های بهتری دارد. ما می‌دانیم در الگوریتم اپسیلون گریدی با اپسیلون متغیر الگوریتم به سرعد همگرا می‌شود و به دلیل این که ۴ جعبه داریم بازیکن دارای این سیاست در حدود ۵ تا۶ حرکت انتخاب بهینه را پیدا می کند. الگوریتم ابتدا explore می‌کند و سپس انتخاب بهینه را دنبال می‌کند و تکرار می کند.

اپسیلون ثابت انتخابی ۱.۰ است و به این معنی است که با فرض اینکه تغییرات اپسیلون از ۱ به ۰ است، بازیکن بیشتر تمایل دارد انتخابی را که تا آن مرحله فکر می‌کند بهترین است را ادامه دهد و در این مورد ممکن است این بازیکن حتا تمام جعبه ها را باز نکند.

بنابر این باید تلاش کنیم بازیکن با الگوریتم اپسیلون گریدی با اپسیلون متغیر را پیدا کنیم.

در انتخاب های اول بهتر است جعبه‌ای را انتخاب کنیم که هنوز انتخاب نشده است و میزان رضایت بازیکنان دیگر از انتخاب ایشان را هم ذخیره کنیم. .سپس به دلیل اینکه ممکن است فردی که به صورت رندوم و یا اپسیلون ثابت به طور اتفاقی جعبه با پاداش بیشتر را انتخاب کنند، بنابر این در ادامه پس از باز شدن همه جعبه ها (حداقل یک بار توسط همه بازیکن ها) جعبه‌ای را انتخاب کنیم که برای دیگران رضایت بیشتری داشته است.

پس از چند دور بازی به احتمال زیاد بازیکن با سیاست اپسیلون متغیر جعبه بهینه را پیدا کرده و هر بار آن جعبه را انتخاب می‌کند بنابر این به هر حال بازیکن ما می‌تواند جعبه بهینه را پیدا کند.

۲- اولین نفری که شروع کرد به باز کردن یک جعبه ثابت همان کسی است از سیاست اپسیلون متغیر استفاده می‌کند و این زمان بهتری برای مشاهده و انتخاب جعبه او می باشد.

۳- اگر منظور از ندیدن پاداش ندیدن رضایت افراد باشد، تأثیری که بر الگوریتم می‌گذارد این است که با تأخیر و دیر تر به جواب می‌رسیم و دیر تر همگرا می شود.

سوال ۲ -

هدف سؤال

هدف از این سؤال مدل سازی مسأله برای پیدا کردن بهترین پیشنهاد برای بسته های اینترنتی به مشتریان است

پیاده‌سازی

در این سؤال به دلیل اینکه لازم بود سیگما را از ریوارد به ایجنت منتقل کنم environment مخصوص به خودش را زدم و sigma را به عنوان information به agent منتقل کردم.

```
def calculate_reward(self, action):
    reward , sigma = self.arms_rewards[action].get_reward()
    self.sigma = sigma
    return reward
```

```
def get_info(self, action):
    return self.sigma
```

در بخش agent تابع utility را به این صورت قرار دادم که اگر کاربر بسته ای را انتخاب نکرده است همان -۱۰ باشد. و اگر مثبت بود پاداش گفته شده را در نظر گرفتم.

```
def utility(self, r):
    if self.sigma == np.inf:
        utility = r
    else:
        utility = r * self.sigma * 0.0005
    return utility
```

مهم‌ترین بخش سؤال تابع reward است که به این صورت است که کاربر با احتمال ۷۰ درصد بسته را می‌خرد و با احتمال ۳۰ درصد نمی خرد. اگر بخرد پاداش آن به این صورت محاسبه می‌شود که ابتدا بر اسس توزیع بتا و پارامتر x و صورت سؤل، اکشن انتخابی را پیدا می کنیم.

```
def calculate_arg_selected_choice(self):
    x = np.random.beta(2, 5, 1)[0]
    ratio = [x*p/v for p, v in zip(self.price, self.volume)]
    self.arg_selected_choice = np.argmin(ratio)
```

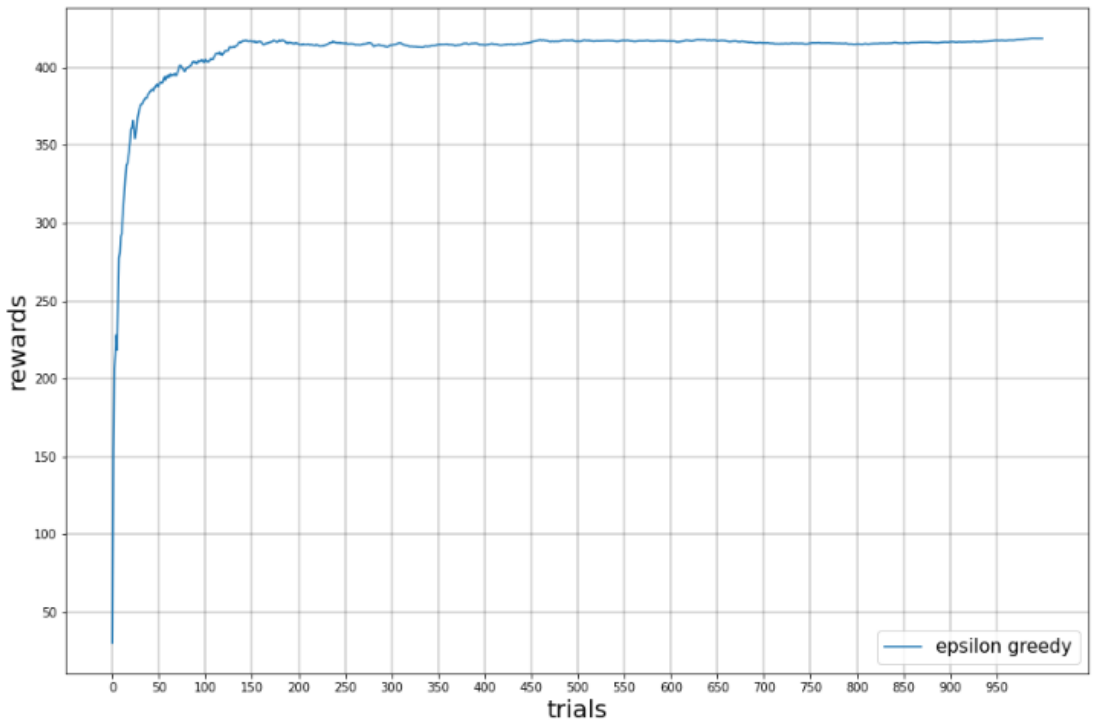
سپس بر اساس رابطه گفته شده میزان پاداش به صورت زیر می باشد:

```
remained_price = self.calculate_remained_price()
reward1 = remained_price + self.price[i]
```

در نهایت سیگما را برای استفاده در utility محاسبه می کند.

```
def calculate_sigma(self, reward1, reward2, reward):
    i = self.arg_selected_choice
    if reward == reward1:
        if i == 0:
            sigma = self.price[i] - 3000
        else:
            sigma = self.price[i] - 1600
    if reward == reward2:
        sigma = np.inf
    return sigma
```

در نهایت نمودار ما به صورت زیر می‌باشد که می‌بینیم به عدد مثبتی همگرا شده یعنی با انتخاب یک اکشن بهینه شرکت سود می‌کند.



```
{} 3531.5341831204373 False 400
945:    action=5
{} 3430.123053562792 False 400
946:    action=5
{} -10.0 False inf
947:    action=5
{} 3499.4004585701955 False 400
948:    action=5
{} 3561.2564861333917 False 400
949:    action=5
{} 3582.819277331625 False 400
950:    action=5
{} 3510.7652322964022 False 400
951:    action=3
{} 3526.613096670645 False 400
952:    action=5
{} 3511.1068184976452 False 400
953:    action=3
```

ب توجه به داده‌ها مشخص است که به بسته پنج همگرا شده است یعنی قیمت های زیر:

[۳۹۰۰, ۲۰۰۰] برای بسته های ۹۰ و ۲۵۰ مگابایتی. که منطقی هم می‌باشد.

به نظر من با توجه به اینکه کاربر بر اساس احتمال مستقلی برای خریدن یا نخریدن تصمیم گیری می‌کند و وابسته به حجم و قیمت بسته ها نیست، طبیعی است که بسته های با حجم بالا برای پیشنهاد به کاربر و به عنوان اکشن بهینه انتخاب شود.

سؤال ۳

هدف سؤال

هدف از این سؤال مدل سازی مسأله برای پیدا کردن بهترین مسیر رفتن به دانشگاه است.

در این مسأله از روش اپسیلون گریدی استفاده کردم و هر یک از ۴ مسیر را یک اکشن در نظر گرفتم و ریوارد های آن‌ها را متناسب با مسأله مطرح شده پیاده‌سازی کردم.

پیاده‌سازی

Reward1

در این پاداش تا انقلاب با مترو می‌رویم و سپس تا امیر آباد تاکسی سوار می شویم.

هزینه مترو هزینه ابتلا به کرونا می‌باشد که مقدار آن را در متد metro_corona_cost به دست می آوریم.

امکان تأخیر مترو را یک توزیع نرمال با میانگین ۵ دقیقه و واریانس ۲ دقیقه درنظرم گرفتم. که این مقدار با توجه به اینکه برنامه قطار ها برنامه‌ریزی شده است منطقی می باشد.

هزینه تاکسی علاوه بر هزینه پرداختی اصلی هزینه زمانی زیادی دارد. به دلیل اینکه مسیر انقلاب تا امیر آباد همیشه ترافیک سنگین دارد. بنابر این هزینه زمانی این بخش را توزیع نرمال ۱۰ دقیقه با واریانس ۵ دقیقه درنظر گرفتم.

برای تاکسی هزینه گرفتن تاکسی را هم در نظر گرفتم که معمولاً بین ۳ تا ۷ دقیقه ممکن است طول بکشد.

```
class Reward1(RewardBase):
    def __init__(self, metro_mean, metro_std, taxi_mean, taxi_std, corona_cost, time_delay_cost_per_minute, corona_p
        super(Reward1, self).__init__()
        self.metro_mean = metro_mean
        self.metro_std = metro_std
        self.taxi_mean = taxi_mean
        self.taxi_std = taxi_std
        self.corona_cost = corona_cost
        self.time_delay_cost_per_minute = time_delay_cost_per_minute
        self.corona_prob_per_10_person = corona_prob_per_10_person

    def metro_corona_cost(self):
        metro_population = np.random.normal(loc=self.metro_mean, scale=self.metro_std)
        corona_prob = metro_population/10 * self.corona_prob_per_10_person
        return corona_prob * self.corona_cost

    def metro_time_delay(self): # 6 istgah ta enghelab  #1000t per minute jarime dir residan
        time_delay = np.random.normal(loc=5, scale=2)  #minutes
        return time_delay*self.time_delay_cost_per_minute

    def taxi_cost(self):
        return np.random.normal(loc=self.taxi_mean, scale=self.taxi_std)

    def taxi_time_delay(self):
        time_delay1 = np.random.normal(loc=5, scale=2)  # 5 minutes taxi gereftan
        time_delay2 = np.random.normal(loc=10, scale=5)  # 15 minutes trafik
        return (time_delay1 + time_delay2) * self.time_delay_cost_per_minute

    def get_reward(self):
        total_metro_cost = self.metro_corona_cost()
        taxi_cost = self.taxi_cost()
        metro_time_delay = self.metro_time_delay()
        taxi_time_delay = self.taxi_time_delay()
        return -(total_metro_cost + taxi_cost + metro_time_delay)
```

Reward2

اکشن شماره ۲ به این صورت است که تا ایستگاه تربیت مدرس با مترو برویم. زمان بیشتری می‌برد و در نتیجه احتمال مبتلا شدن به کرونا را ۲ برابر حالت قبل در نظر گرفتم.

هزینه تأخیر را هم به دلیل این که تعداد ایستگاه ها بیشتر می‌باشد به طور میانگین ۱۵ دقیقه در نظر گرفتم.

```
class Reward2(RewardBase):
    def __init__(self, metro_mean, metro_std, corona_cost, time_delay_cost_per_minute, corona_prob_per_10_person):
        super(Reward2, self).__init__()
        self.metro_mean = metro_mean
        self.metro_std = metro_std
        self.taxi_mean = taxi_mean
        self.taxi_std = taxi_std
        self.corona_cost = corona_cost
        self.time_delay_cost_per_minute = time_delay_cost_per_minute
        self.corona_prob_per_10_person = corona_prob_per_10_person

    def metro_corona_cost(self):
        metro_population = np.random.normal(loc=self.metro_mean, scale=self.metro_std)
        corona_prob = metro_population/10 * self.corona_prob_per_10_person
        return corona_prob * self.corona_cost * 2

    def metro_time_delay(self): # 15 istgah ta tarbiat modares
        time_delay = np.random.normal(loc=15, scale=5)  #minutes
        return time_delay*self.time_delay_cost_per_minute

    def get_reward(self):
        metro_corona_cost = self.metro_corona_cost()
        metro_time_delay = self.metro_time_delay()
        return -(metro_corona_cost + metro_time_delay)
```

Reward3

در این حالت که کلاً ۳ تاکسی می‌گیریم احتمال ترافیک را به طور میانگین ۱۵ دقیقه در نظر گرفتم.

```
class Reward3(RewardBase):
    def __init__(self, taxi_mean, taxi_std, corona_cost, time_delay_cost_per_minute):
        super(Reward3, self).__init__()
        self.taxi_mean = taxi_mean
        self.taxi_std = taxi_std
        self.corona_cost = corona_cost
        self.corona_prob_per_10_person = corona_prob_per_10_person
        self.time_delay_cost_per_minute = time_delay_cost_per_minute

    def taxi_cost(self):
        return np.random.normal(loc=self.taxi_mean, scale=self.taxi_std) * 3

    def taxi_time_delay(self): #3 ta taxi
        time_delay1 = np.random.normal(loc=5, scale=2)  # 5 minutes taxi gereftan
        time_delay2 = np.random.normal(loc=10, scale=5)  # 15 minutes trafik
        return (time_delay1 + time_delay2)*self.time_delay_cost_per_minute*3

    def get_reward(self):
        taxi_cost = self.taxi_cost()
        taxi_time_delay = self.taxi_time_delay()
        return -(taxi_cost+taxi_time_delay)
```

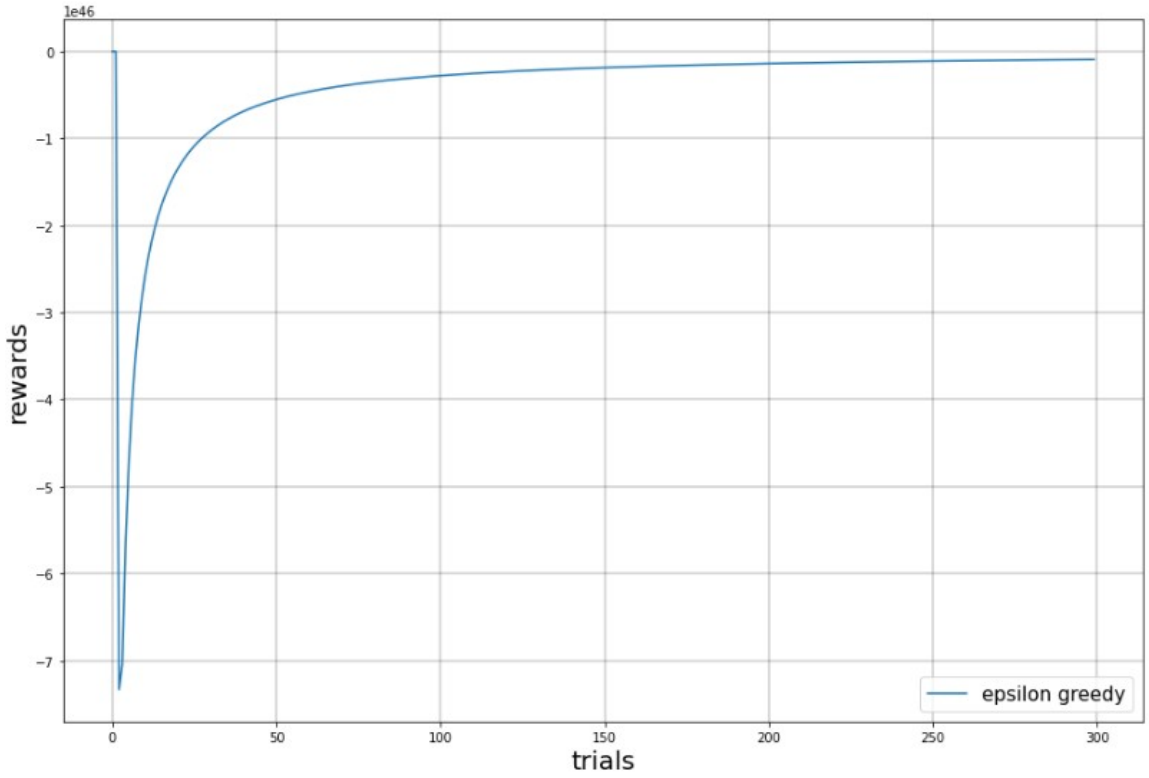
Reward4

در این حالت کل مسیر را یک اسنپ می گیریم.

```
class Reward4(RewardBase):
    def __init__(self, snap_mean, snap_std, corona_cost, time_delay_cost_per_minute):
        super(Reward4, self).__init__()
        self.snap_mean = snap_mean
        self.snap_std = snap_std
        self.corona_cost = corona_cost
        self.corona_prob_per_10_person = corona_prob_per_10_person
        self.time_delay_cost_per_minute = time_delay_cost_per_minute

    def snap_cost(self):
        return np.random.normal(loc=self.snap_mean, scale=self.snap_std)

    def snap_time_delay(self): #trafik
        time_delay1 = np.random.normal(loc=5, scale=2)  # 5 minutes snap gereftan
        time_delay2 = np.random.normal(loc=10, scale=5)  # 30 minutes trafik
        return (time_delay1 + time_delay2)*self.time_delay_cost_per_minute

    def get_reward(self):
        snap_cost = self.snap_cost()
        snap_time_delay = self.snap_time_delay()
        return -(snap_cost+snap_time_delay)
```

نتیجه

نمودار همگرا می شود.



با توجه به اکشن ها متوجه می‌شویم که در نهایت اکشن صفر یعنی استفاده از مترو تا انقلاب و سپس تاکسی تا امیر آباد به صرفه تر است.

```
  1:      action=0
action=1      :2
action=2      :3
action=3      :4
action=1      :5
action=0      :6
action=1      :7
action=1      :8
action=1      :9
action=1     :10
action=1     :11
action=1     :12
action=1     :13
action=1     :14
action=1     :15
action=1     :16
action=1     :17
action=1     :18
action=1     :19
action=1     :20
```

```
action=1     :21
action=1     :22
action=1     :23
action=1     :24
action=1     :25
action=1     :26
action=1     :27
action=1     :28
action=1     :29
action=1     :30
action=1     :31
action=0     :32
action=0     :33
action=1     :34
action=0     :35
action=0     :36
action=0     :37
action=0     :38
action=0     :39
action=0     :40
action=0     :41
action=0     :42
action=0     :43
action=0     :44
action=0     :45
action=0     :46
```

در این مسأله به دلیل اینکه ایجنت risk averse **است یعنی از ریسک کردن دوری می‌کند باید برای پاداش منفی که دریافت می‌کند آن را خیلی منفی تر ببیند. درواقع پارامتر لاندا که ضریب پاداش منفی در تابع** utility **است را زیاد قرار دادم.**

نکات مهم و موارد تحویلی

لازم است که به نکات زیر در نوشتن گزارش توجه داشته باشید.

۱. **ساختار کلی گزارش که در این فایل به آن اشاره شده باید رعایت شود. در صورت تمایل می‌توانید از latex یا هر نرم افزار دلخواه دیگر برای نوشتن گزارش استفاده کنید، به شرط اینکه ساختار کلی گفته شده رعایت شود. لذا در صورت رعایت نکردن ساختار کلی گزارش بخشی از نمره تمرین کم خواهد شد.**
۲. **برای تصاویر موجود در گزارش حتما زیر نویس و برای جداول استفاده شده در گزارش بالانویس (اجباری) قرار داده شود.**
۳. **نتایج و تحلیل‌های شما در روند نمره دهی اهمیت بسیار بالایی دارد، لذا خواهشمندیم کلیه نتایج و تحلیل‌های خواسته شده به صورت کامل و دقیق در گزارش آورده شوند.**
۴. **در صورت مشاهده شباهت بین گزارش شما و افراد مختلف نمره این سری تمرین برای شما در نظر گرفته نمی‌شود.**

موارد تحویلی

۱. **برای هر سری از تمرینات، فقط یک فایل با فرمت PDF آماده کنید.**
۲. **به همراه فایل گزارش، یک پوشه به نام Codes ایجاد کنید و کدها و فایل‌های پیاده‌سازی هر سوال را به صورت تفکیک شده در پوشه‌های جداگانه قرار دهید.**
۳. **هیچ گونه جدول یا تصویر به صورت جداگانه خارج از گزارش ارسال نشود. مگر اینکه به صورت صریح در تمرین از شما خواسته شده باشد.**
۴. **در انتها، لطفا برای هر تمرین گزارش و پوشه کدها را به صورت گفته شده، در یک فایل زیپ با فرمت زیر در سامانه یادگیری الکترونیک بارگذاری نمایید.**

HW#_LastName_StudentNumber.zip

به طور مثال:

HW۱_Mesbah_۸۱۰۱۱۱۱۱۱.zip