Правительство Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Магистерская программа «Науки о данных»

Департамент анализа данных и искусственного интеллекта

Курсовая работа

на тему:

Ранжирование методом многорукого бандита

студенка группы НоД_ИССА15

Куликова Анна Геннадиевна

Руководитель

Кертес-Фаркаш Аттила

Москва 2016

Government of Russian Federation

Federal State Autonomous Educational Institution of Higher Professional Education

National Research University

«Higher School of Economics»

Faculty of computer science

Master program «Data science»

Department of data analysis and artificial intelligence

Course project

Learning to Rank with Multi-Armed Bandits

Student of group DS_ISSA15

Kulikova Anna

Scientific Advisor

Kertész-Farkas Attila

Москва 2016

Contents

0 Introduction

The multi-armed bandit problem models an agent that should "explore" environment and simultaneously "exploit" (choose the optimal action based on existing knowledge). After choosing action, agent receive from environment a reward . In this way the goal of agent is maximized the cumulative reward.

There are many multi-armed bandit algorithms. Which one to choose depends on the data, computer performance and goals. These methods applies in recommendation systems. Another example of the application of these method is use it instead of A/B testing. Google and Yandex used the multi-armed bandit for rank web pages. Also a lot of shops using it on their web cites. First this problem was invented in 1933 and has been used for choosing a medicine for the patient.

The most well-known multi-armed bandit algorithms are UCB1, ϵ-greedy, Gittins Index and Thompson sampling. But this algorithms do not use information about environment or arms. So that was invented contextual bandit problem. Contextual bandit problem is a multi-armed bandit problem, where selection strategy based on information about environment or arms.

This paper contains description of algorithms and their implementation. The data for this algorithms downloaded from Yahoo datasets and presented here. Also this work contains comparison of results.

1 Multi-armed bandit

1.1 Definition

Multi-armed bandit (or K-armed bandit) problem is a sequential allocation problem defined by a set of actions. The action is chosen at each time step and a payoff is observed. The goal is maximize the payoff. Bandit problem based on an action selection with limited information. Thus we have a trade-off between an exploration and an exploitation.

The problem was named so, because it is like a set of slot machines (one-armed bandit) and you want to get as much as possible money. Each time step you should choose between "exploit" the best machine (based on the data from the past) or "explore" another machine.

Thompson was the first who began to study this problem on a clinical trials.He had a lot of drugs for each disease and he must choose how to cure a person.

There are three formalizations for this problem: stochastic, adversarial, and Markovian. The difference is nature of the reward process.[1] In this paper describes only a stochastically formalized algorithm.

A multi-armed formulation.

K- number of arms, for each trial t = 1,2 ...:

1. choose an arm a from K arms
2. receive a payoff r_a from the environment (independently from the past)
3. make a prediction based on a received payoff

In the stochastic formalization the pseudo-regret function can be written as

$$\overline{R}_n = n\mu^* - \sum_{i=1}^{n} E(\mu_i),$$

where a_i is an arm, $i = 1,2,...,K$, μ_{a_i} - a mean of reward of i-th arm,

$$\mu^* = \max_{i \in 1,2,..,K} \mu_i.$$

1.2 Upper Confidence Bound (UCB) algorithms

The UCB algorithms has been invented by Cesa-Bianchi and Fisher in 2002. The main idea of an algorithm is as follows: for each action an estimated confidence threshold of the action is $U(a)$. We need to estimate this such that the following inequation $Q(a) \leq \overline{Q_t}(a) + U(a)$ holds with high probability, where $\overline{Q_t(a)}$ is an average payoff obtained after selection the action a.

The main variation of this algorithm is known as UCB1. Each trial UCB1 selects an action according to the equation:

$$a = \arg \max_{a \in 1,2,...,K} \overline{Q(a)} + \sqrt{\frac{2\log(t)}{N_t(a)}}$$

The article [2] states that the algorithm has a logarithmic loss function. One interesting thing about this algorithm that on each step we require only $O(1)$ time for choosing the best arm. The other good feature is that there is no any assumptions about rewards probability distribution.

But this algorithm choose without information about an arm. Suppose we should rank web pages for the query. This algorithm will put in the first place the most popular web site for this query. But the UCB1 isn't using any information about user. Using this information we can rank pages better. So appeared contextual bandit approach.

1.3 Contextual-Bandit approach

The contextual-bandit approach modifies the existing algorithms of the multi-armed bandit problem. At this approach often used a UCB algorithm, but you can use the ϵ-greedy algorithm. [3] Suppose we need to organize a personalized recommendation of news articles for each user. Often users are represented as a set of features. This features may include a different information such as: demographic (for instance: age, sex), historical activities and so on. These information should be represented as a vector of real numbers. It isn't necessary to use only the information about user, we can use also a description of article or both. In this paper I used only user information. These information (features) are named "contextual" information.

The contextual-bandit problem is a learning algorithm which selects news articles for user. The selection is based on contextual information. In this case better to use clicks as a reward. It's may be 0 if there is no click and 1 if there is. You can use CTR information, but it will need more RAM. So I used the first approach.

The contextual-bandit algorithms divided into two classes. The first class contains algorithms with linear prediction functions, which are LinUCB with disjoint linear models and LinUCB with hybrid linear models.

The other class contains algorithms with non-liner prediction functions such as UCBogram algorithm, NeuralBandit algorithm, KernelUCB algorithm and Bandit Forest algorithm.

I implemented LinUCB algorithm with disjoint linear models.

1.4 LinUCB algorithm with disjoint liner models

The article [3] showed that a confidence interval can be computed efficiently in closed form when a payoff model is linear. An expected payoff of the arm a is liner. The payoff based on the vector x_t,a of user features (d-dimensional vector) with some unknown vector of coefficients θ_a^* o, where t is trial:

$$E[r_t,a|x_t,a] = x_t^T,a\theta_a^*$$

This model was named so because each arm (article) has it's own parameters. Paper [3] states that the selection strategy is as follows:

$$a_t = \arg\max_{a\in A_t}(x_{t_a}^T\hat{\theta}_a + \alpha\sqrt{x_{t,a}^T A_a^{-1} x_{t,a}})$$

, where

$$\hat{\theta}_a = (D_a^T D_a + I_d)^{-1} D_a^T c_a$$

, D_a is $m \times d$-dimensional matrix of history, b_a is d-dimensional vector of click history, c_a vector contains confidence intervals corresponding articles in rows of D_a, α the constant parameter,

$$A_a = D_a^T D_a + I_d$$

, I_d is $d \times d$-dimensional identity matrix.

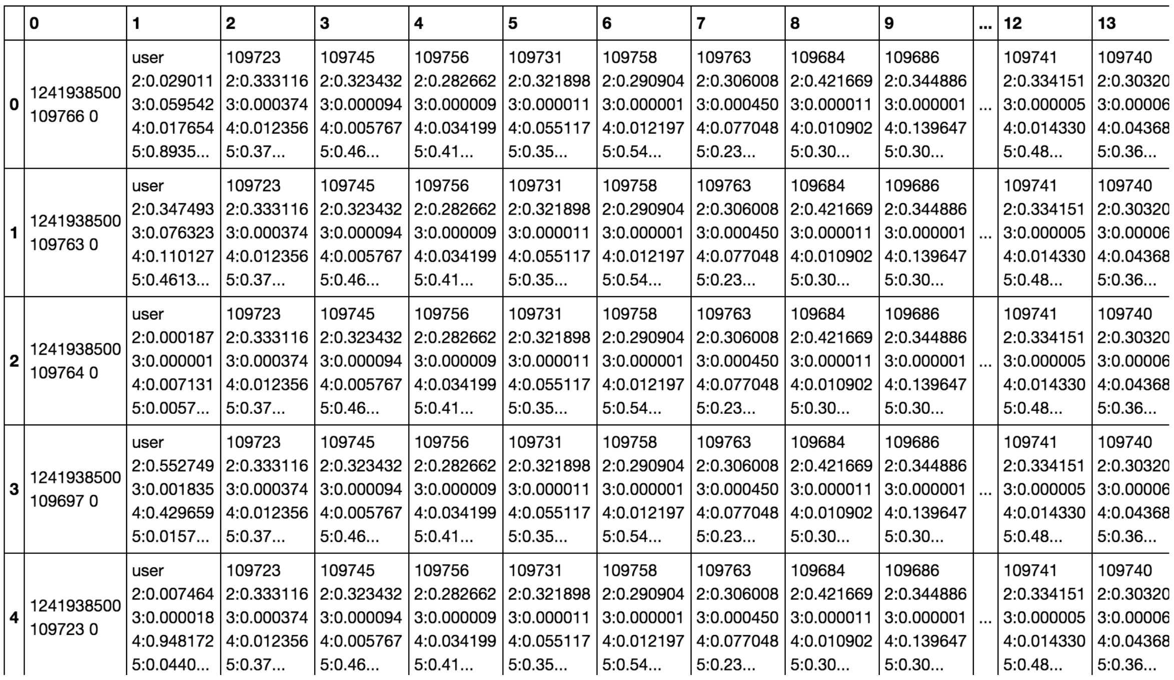
Algorithm 1 LinUCB with disjoint liner models

```
    Inputs: $\alpha \in \mathbb{R}_+$
 1: for $t \leftarrow 1$ to $T$ do Observe $x_{t,a} \in \mathbb{R}^d$ (users features vector)
 2:     for all arms $a \in A_t$ (list of arm can change) do
 3:         if $a$ is new then
 4:             $A_a \leftarrow I_d$ (d-dimensional identity matrix)
 5:             $b_a \leftarrow 0_{d\times 1}$ (d-dimensional zero vector)
 6:         end if
 7:         $\hat{\theta}_a \leftarrow A_a^{-1} b_a$
 8:         $p_{t,a} \leftarrow \hat{\theta}_a^T x_{t,a} + \alpha\sqrt{x_{t,a}^T A_a^{-1} x_{t,a}}$
 9:     end for
10:         choose arm $a = \arg\max_{a\in A_t} p_{t,a}$ with ties broken arbitrarily,
11:         and observe a real-value payoff $r_t, a$
12:         $A_a \leftarrow A_a + x_{t,a} x_{t,a}^T$
13:         $b_a \leftarrow b_a + r_t x_{t,a}$
14: end for
```

2 Data description

I took the data at the Yahoo datasets. The first column contains three numbers: first is time step, second is displayed article, which was chosen uniformly at random from the article pool and third is click information (it is 1 if user click on this article and 0 if otherwise). The second column contains user features. The first feature always same and equals 1, but the other five features are different. From the second column to the twenty-second column are presented the pool of available articles id and features. The features was 5-dimensional like user features and contains one constant feature. I used only user features. At each visit algorithm should choose one article from the pool and this article will displayed.

The downloaded ZIP file contains 10 ZIP files with data. After I unzip the minimal file the size was 6,6Gb. This file contains 3,770,122 lines, 39 different articles and 2,668,056 different users.



	0	1	2	3	4	5	6	7	8	9	...	12	13
0	1241938500 109766 0	user 2:0.029011 3:0.059542 4:0.017654 5:0.8935...	109723 2:0.333116 3:0.000374 4:0.012356 5:0.37...	109745 2:0.323432 3:0.000094 4:0.005767 5:0.46...	109756 2:0.282662 3:0.000009 4:0.034199 5:0.41...	109731 2:0.321898 3:0.000011 4:0.055117 5:0.35...	109758 2:0.290904 3:0.000001 4:0.012197 5:0.54...	109763 2:0.306008 3:0.000450 4:0.077048 5:0.23...	109684 2:0.421669 3:0.000011 4:0.010902 5:0.30...	109686 2:0.344886 3:0.000001 4:0.139647 5:0.30...	...	109741 2:0.334151 3:0.000005 4:0.014330 5:0.48...	109740 2:0.30320 3:0.00006 4:0.04368 5:0.36...
1	1241938500 109763 0	user 2:0.347493 3:0.076323 4:0.110127 5:0.4613...	109723 2:0.333116 3:0.000374 4:0.012356 5:0.37...	109745 2:0.323432 3:0.000094 4:0.005767 5:0.46...	109756 2:0.282662 3:0.000009 4:0.034199 5:0.41...	109731 2:0.321898 3:0.000011 4:0.055117 5:0.35...	109758 2:0.290904 3:0.000001 4:0.012197 5:0.54...	109763 2:0.306008 3:0.000450 4:0.077048 5:0.23...	109684 2:0.421669 3:0.000011 4:0.010902 5:0.30...	109686 2:0.344886 3:0.000001 4:0.139647 5:0.30...	...	109741 2:0.334151 3:0.000005 4:0.014330 5:0.48...	109740 2:0.30320 3:0.00006 4:0.04368 5:0.36...
2	1241938500 109764 0	user 2:0.000187 3:0.000001 4:0.007131 5:0.0057...	109723 2:0.333116 3:0.000374 4:0.012356 5:0.37...	109745 2:0.323432 3:0.000094 4:0.005767 5:0.46...	109756 2:0.282662 3:0.000009 4:0.034199 5:0.41...	109731 2:0.321898 3:0.000011 4:0.055117 5:0.35...	109758 2:0.290904 3:0.000001 4:0.012197 5:0.54...	109763 2:0.306008 3:0.000450 4:0.077048 5:0.23...	109684 2:0.421669 3:0.000011 4:0.010902 5:0.30...	109686 2:0.344886 3:0.000001 4:0.139647 5:0.30...	...	109741 2:0.334151 3:0.000005 4:0.014330 5:0.48...	109740 2:0.30320 3:0.00006 4:0.04368 5:0.36...
3	1241938500 109697 0	user 2:0.552749 3:0.001835 4:0.429659 5:0.0157...	109723 2:0.333116 3:0.000374 4:0.012356 5:0.37...	109745 2:0.323432 3:0.000094 4:0.005767 5:0.46...	109756 2:0.282662 3:0.000009 4:0.034199 5:0.41...	109731 2:0.321898 3:0.000011 4:0.055117 5:0.35...	109758 2:0.290904 3:0.000001 4:0.012197 5:0.54...	109763 2:0.306008 3:0.000450 4:0.077048 5:0.23...	109684 2:0.421669 3:0.000011 4:0.010902 5:0.30...	109686 2:0.344886 3:0.000001 4:0.139647 5:0.30...	...	109741 2:0.334151 3:0.000005 4:0.014330 5:0.48...	109740 2:0.30320 3:0.00006 4:0.04368 5:0.36...
4	1241938500 109723 0	user 2:0.007464 3:0.000018 4:0.948172 5:0.0440...	109723 2:0.333116 3:0.000374 4:0.012356 5:0.37...	109745 2:0.323432 3:0.000094 4:0.005767 5:0.46...	109756 2:0.282662 3:0.000009 4:0.034199 5:0.41...	109731 2:0.321898 3:0.000011 4:0.055117 5:0.35...	109758 2:0.290904 3:0.000001 4:0.012197 5:0.54...	109763 2:0.306008 3:0.000450 4:0.077048 5:0.23...	109684 2:0.421669 3:0.000011 4:0.010902 5:0.30...	109686 2:0.344886 3:0.000001 4:0.139647 5:0.30...	...	109741 2:0.334151 3:0.000005 4:0.014330 5:0.48...	109740 2:0.30320 3:0.00006 4:0.04368 5:0.36...

However often we need train this model or check it on "offline" data. I had only "offline" data, so that I was forced to model the feedback. When the algorithm choose an article form article pool, I check what the article was chosen at the data and if this articles are same, I collect the feedback information (payoff), else I ignore it and move on to a new raw.

3 Work description

I wrote my code on Python 2.7. For the Python were written a great number of libraries. It's convenient to perform mathematical calculations and visualize the results on the Python.

I processed the data line by line, each line is trial. In each trial I choose article from article pool and collect the feedback or ignore it (if I had no information about feedback of this article in that line).

As you can see, LinUCB algorithm requires to store the information about article. Such as the matrix A and the vector b. Thus I constructed two hash tables: article id $\to$ matrix A and article id vector b. Also LinUCB algorithm contains the matrix A inversion and the matrix multiplication. So that I constructed two additional hash tables: article id $\to$ inverse matrix A, article id $\to$ $\hat{\theta}$. LinUCB algorithm use user features, so I create the function, that construct the features vector from the data. This function gets the string from each cells in the second collumn and returns 5-dimentional (constant feature I ignored) vector of floats:

```
def extract_features(features_string):
    return np.matrix(map(lambda x:
        float(x.split(':')[1]),
            features_string.split(' ')[1:6]))
```

For each line I calculate the payoff of all articles in the pool and chose article with biggest payoff. After I collect the real payoff from the first column in data (if it exist) and update information about article in all hash tables.

My code runs near 3384 seconds (40 minutes) and processes the data with rate 1580 lines in second.

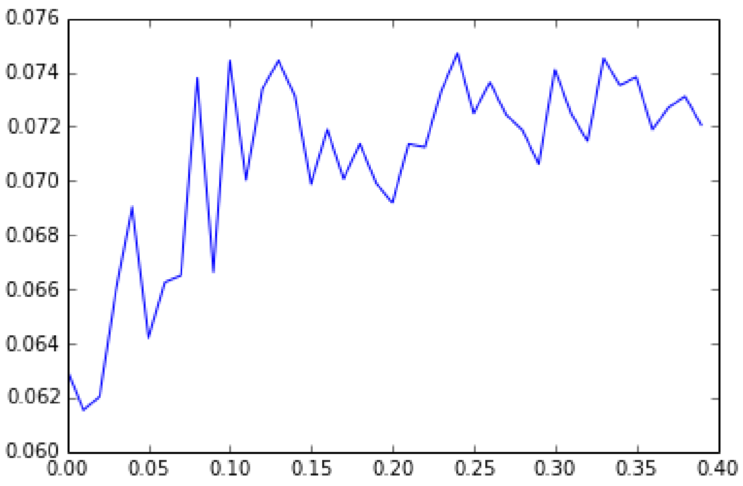
Also I wrote the UCB1 algorithm for the comparing results. This algorithm used only two hash tables: article id $\rightarrow$ number of shows this article and article id $\rightarrow$ expected payoff. The runing time of this algorithms was 421 seconds and rate - 8942 lines per second.

4 Results

In this work I calculated performance as the ratio of clicks, that was received by the algorithm and of the total number of clicks. This calculation of performance don't give information about how this algorithm is better than random selection of articles. But it's give information how LinUCB is better than UCB1. And makes it possible to find the best value of performance with different alpha and compare the results of algorithms!

As you can see, LinUCB algorithm depends on $0 < \alpha$, so I considered the performance for each α on the interval $[0.0, 0.4]$ with step 0.01 in Figure 1.

Figure 1: Performance of LinUCB algorithm depends of alpha

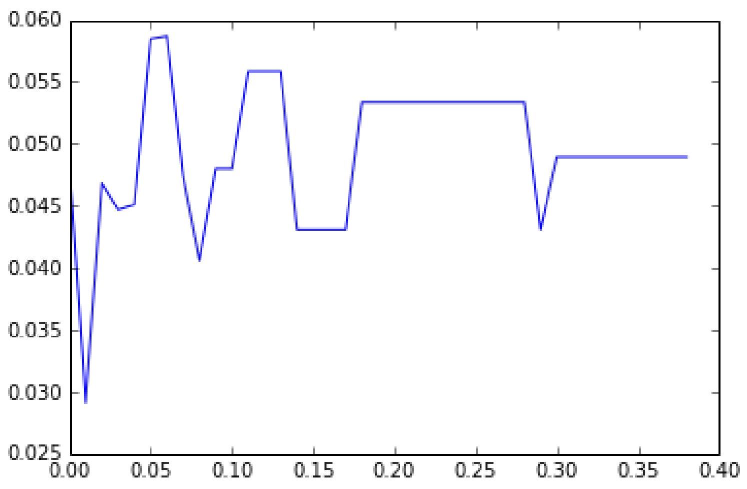


The performance reached the highest value at $\alpha = 0.24$ and it value is approximately equal 0.075. In paper [3] showed same result: performance reached highest value at $\alpha = 0.2$ (there performance compute with larger step, than in my work)

Also I drew the dependence on alpha for the performance of algorithm UCB1 in Figure 2.

As you can see, the best performance reached, when α is approxi-

Figure 2: Performance of UCB1 algorithm depends of alpha



mately equal 0.05 and it's value is 0.0584. So that, if you have web site with news, where you should choose for each user main article from article pool. And billion of people will visited you site, the LinUCB algorithm will collected on 16,600 clicks more than UCB1.

Let's look at the performance of the dependency of the data size. For this I consistently used six files from the dataset for UCB1 algorithm and five for LinUCB algorithm. In this six files contained 170 unique articles and 18,883,028 unique users.

Figure 3: Performance of LinUCB algorithm depends of quantity of data

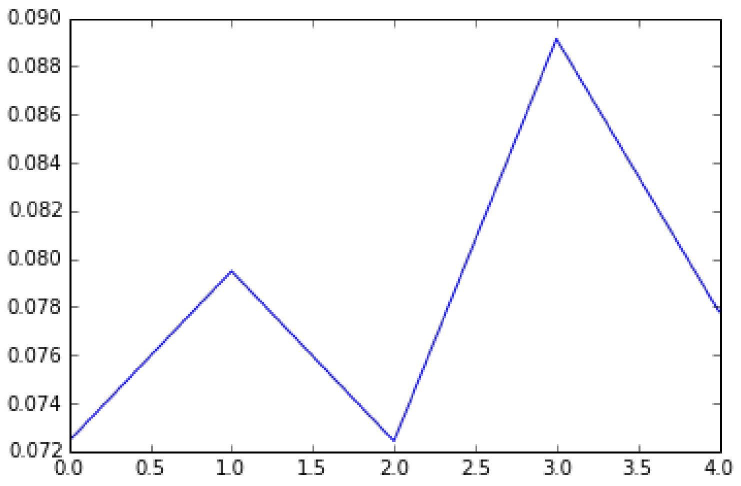
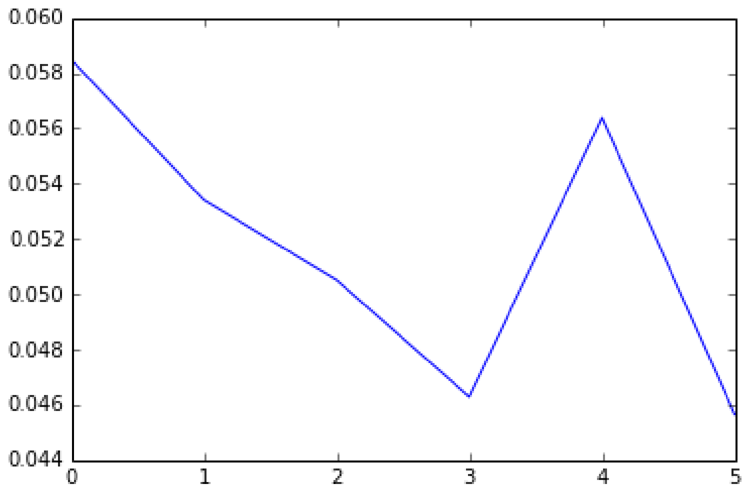


Figure 4: Performance of UCB1 algorithm depends of quantity of data



Each file has a size that is different from other, so that in Figure 3 and Figure 4 are different steps. The numbers on the abscissa indicate how many data files were used. As you can see, performance of the UCB1 algorithm decreases, when in LinUCB algoritm increases. It happened, because when UCB1 meets new data, the algorithm "explores" this data in contrast to the LinUCB algorithm that makes decisions based on the context information. The average values of the performance fluctuations for the both algorithms are equal the best values of performance from Figure 1 and Figure 2 with 0.005 accuracy.

Also I plotted the dependence of the runing time of algorithms on quantity of data on Figure 5 and Figure 6. The runing time dependence of the algorithm UCB1 on the quantity of data is similar to the linear. But for LinUCB algorithm runing time dependence of the quantity of lines in data file. For instance, the second file contains 4,714,618 lines and algorithm processed this file 4,212 seconds, when the first file contains 3,770,122 lines and it's runing time was 3,384 seconds. As you can see, on Figure 6 runing time dependence of lines in data file (the latest data files contain about 5 billion rows). As the files were processed one by one, hash tables collect the context information for each article. Note, that time of using information from hash tables very small.

Figure 5: Runing time of UCB algorithm depends of quantity of data

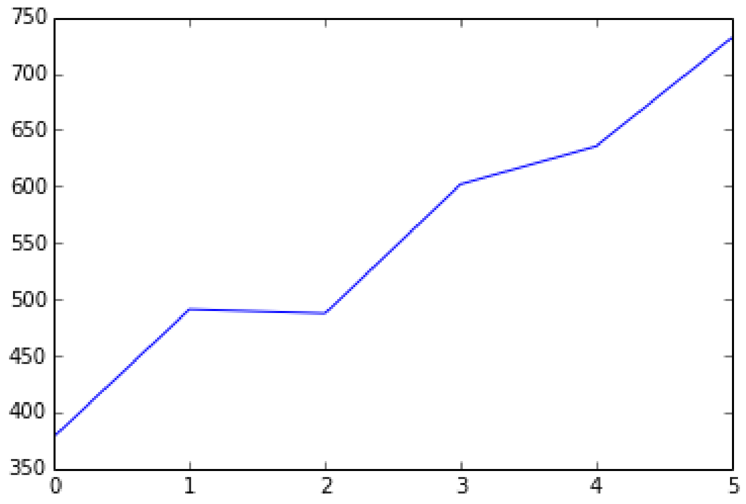
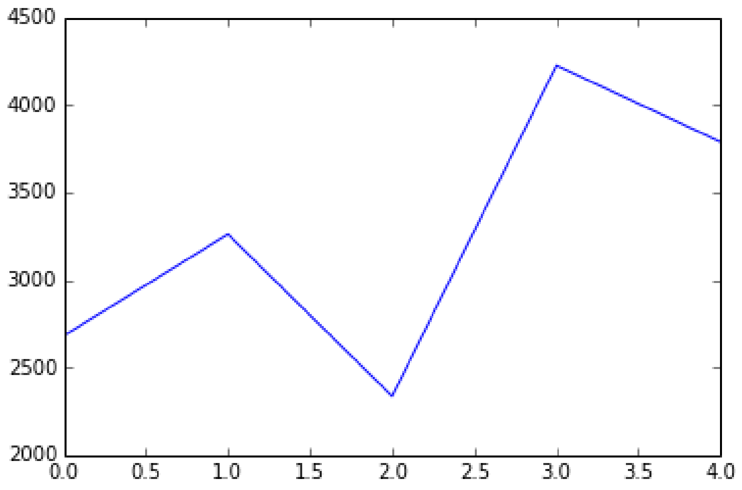


Figure 6: Runing time of LinUCB algorithm depends of quantity of data



5 Conclusion

In this paper has been formulated the exploration-exploitation problem. The main field of application solutions to this problem is recommendation systems. Here has been describe the most known basic algorithm (UCB1). It was also considered a contextual approach to the multi-armed bandit problem. And the most known algorithm of this approach is LinUCB. This algorithm has several formulations. In this paper used only one: LinUCB algorithm with disjoint liner models.

In this paper has been formulated the the multi-armed bandit problem and has been described two algorithms: UCB1 and LinUCB. Each algorithm was written on Python 2.7. The results of this algorithms was compared on Yahoo news dataset, where for each user algorithms should choose article from article pool. Research results have shown, that the algorithm of contextual bandit work better, than the UCB1 algorithm. But LinUCB algorithm process the data four times slower, than UCB1 algorithm and and needs more RAM.

References

[1] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *CoRR*, abs/1204.5721, 2012.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[3] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. *CoRR*, abs/1003.0146, 2010.