Université Jean Monnet

Advanced Machine Learning

Assignment Report

Online Learning, Bandits and Reinforcement Learning

Author:
Rohil Gupta
Mehdi Zarria

Supervisor:
Amaury Habrard

December 18, 2018

Contents

1 Introduction **3**

2 Online Learning: Passive Agreesive Algorithm **4**
2.1 Overview of method 4
2.2 Experimental Setup 5
2.3 Results 6
2.3.1 Evaluation of accuracy involving updated weights per 50 examples 6
2.3.2 Evaluation of accuracy of the model on all examples . 9
2.3.3 Evaluation of accuracy of the model on all examples after randomly flipping the labels 11
2.3.4 Comparison of accuracy with Support Vector Machine 12
2.4 Conclusion 12

3 Bandits **13**
3.1 Incremental uniform 13
3.1.1 Theory study 13
3.1.2 Experiments and results 14
3.2 ϵ-Greedy 19
3.2.1 Theory study 20
3.2.2 Experiments and results 21
3.3 UCB 25
3.3.1 Experiments and results 26

4 Reinforcement Learning **29**
4.1 problem presentation 29
4.2 Experiments 29

List of Figures

1 Passive Aggressive Online algorithm 5
2 Accuracy for Classic Update 6
3 Accuracy per 50 examples for different values of C 7
4 Accuracy per 50 examples for different values of C 8
5 Plot for Accuracy of Algorithm for First Relaxation Update for different values of C 9

6 Plot for time taken for Algorithm for First Relaxation Update for different values of C 9
7 Plot for Accuracy of Algorithm for Second Relaxation Update for different values of C 10
8 Plot for time taken for Algorithm for Second Relaxation Update for different values of C 10
9 Plot for Accuracy of Algorithm for First Relaxation Update for different values of C after random flipping of the labels . 11
10 Plot for Accuracy of Algorithm for Second Relaxation Update for different values of C after random flipping of the labels 12
11 Impact of the number of experiments on the same probabilities 17
12 Impact of the number of experiments on the same probabilities 18
13 Impact of the number of episodes on the same probabilities . 19
14 Impact of the number of episodes on the same probabilities . 22
15 Impact of the number of episodes on the same probabilities . 23
16 Impact of the number of episodes on the same probabilities . 24
17 UCB bound and mean value presentation 25
18 The development of the average reward UCB 27
19 Occurrences of chosen bars 28
20 Changes of the reward with respect to ϵ 30
21 Changes of the reward with respect to γ 31

1 Introduction

This assignment addresses three methods as follows:

1. **Online Learning: Passive Aggressive Algorithm**
   In computer science, online machine learning is a method of machine learning in which data becomes available in a sequential order and is used to update our best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once.

   In the specific task we had to implement the Passive Aggressive Algorithm from scratch which can be used for Online Learning for the stream of data. Passive Agressive Online algorithm is a special case of online algorithms for binary classification that can be seen as a extension of SVM to the context of online learning. The principle followed is that for each iteration to update the classifier in order to remain as close as possible to the current one while achieving at least a unit margin on the most recent example. However, forcing a unit margin might turn out to be too aggressive in the presence of noise, so there were 2 variations that can be considered casting a tradeoff between the desired margin and the proximity to the current classifiers.

2. **Bandits** One of the important problems that can be solved using the machine learning algorithms is Bandits problem. In this practical session, we will try to implement three of bandits algorithms, Incremental uniform, UCB and ϵ-Greedy.

3. **Reinforcement Learning**: Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In the operations research and control literature, reinforcement learning is called approximate dynamic programming, or neuro-dynamic programming. The problems of interest in reinforcement learning have also been studied in the theory of optimal control, which

is concerned mostly with the existence and characterization of optimal solutions, and algorithms for their exact computation, and less with learning or approximation, particularly in the absence of a mathematical model of the environment. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality. In machine learning, the environment is typically formulated as a Markov Decision Process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible. [1]

2 Online Learning: Passive Agreesive Algorithm

2.1 Overview of method

In this method a passive aggressive algorithm is used which can be seen in the pesudo code given in figure1. The methods has three different updates for τ given by:

- The classic update : $\tau_t = \frac{l_t}{||x_t||^2}$
- A first relaxation : $\tau_t = \min\{\mathrm{C}, \frac{l_t}{||x_t||^2}\}$
- A second relaxation : $\tau_t = \frac{l_t}{||x_t||^2 + \frac{1}{2C}}$

[1] https://en.wikipedia.org/wiki/Reinforcement$_l earning$

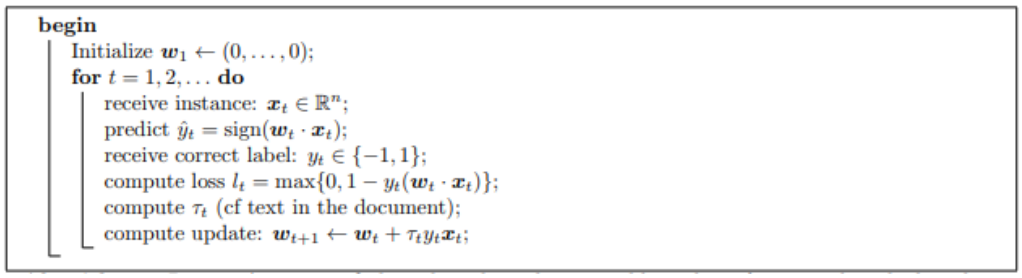


Figure 1: Passive Aggressive Online algorithm

2.2 Experimental Setup

The implementation of this algorithm is divided into different phases, which can be breifly described in following points:

- The algorithm given in figure 1 is implemented with different update strategies for τ which can be selected in the function made.
- There are two evaluation techniques being used:
    1. The first technique involves getting the set of updated weights every 50 samples of data and using these set of weights on the test data to check the performance of the online learning.
    2. The second technique involves checking the accuracy of the model after all the examples, which involves predicition and update of weights for each new sample of data.
- The other step invloves comparing the Online Passive Aggressive Algorithm with Support Vector Machines.
- Last phase involves random flipping of the class labels to see the impact on the accuracy levels for all update strategies.

These different phases are descriptively explained in Jupyter Notebook. The results obtained are explained in following section phase wise.

2.3 Results

2.3.1 Evaluation of accuracy involving updated weights per 50 examples

1. **For Classic Update of τ:** The figure 2 shows the accuracy of the model computed from the weights obtained every 50 examples for whole dataset and using these weights to predict the labels of the test set.

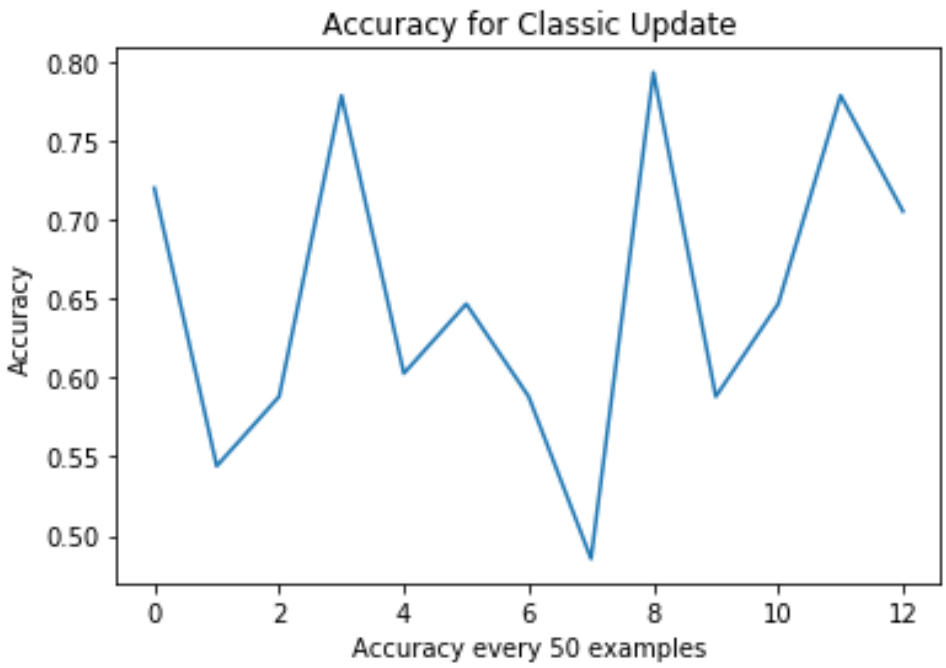
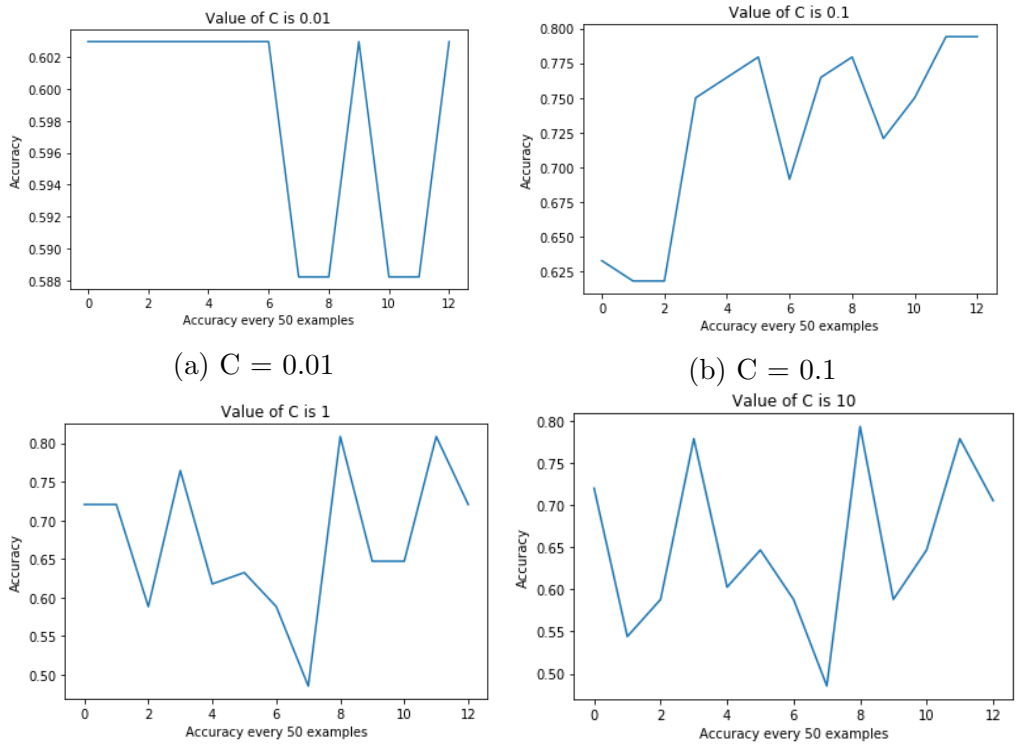


Figure 2: Accuracy for Classic Update

2. **For First Relaxation Update of τ:** The figure 3 shows the accuracy graph obtained per 50 examples for different values of C after testing it on the test data. It is seen in figures 3a, 3b, 3c and 3d that it follows a general expected trend for the increase in accuracy levels after every 50 examples observed.
   There are though some exceptions observed with accuracy drops, this can be seen because of the outliers in the data that diverges the weights from the optimal values. It can be avoided by removing these outliers to get the expected trend in increase in accuracy.



(a) C = 0.01

(b) C = 0.1

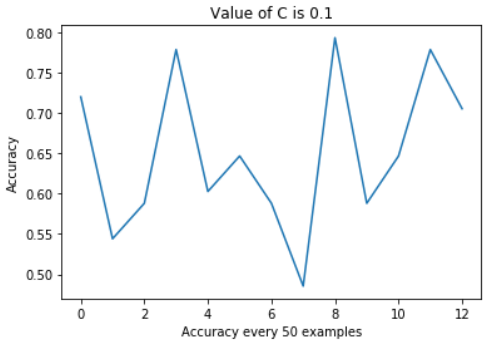
(c) C = 1

(d) C = 10

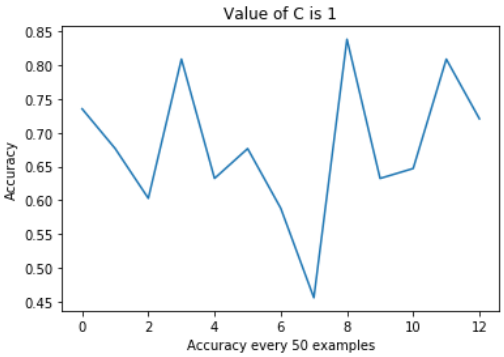
Figure 3: Accuracy per 50 examples for different values of C

It is also observed that the accuracy level increases with the value of C till value 1 and after that there is a drop in the accuracy observed.

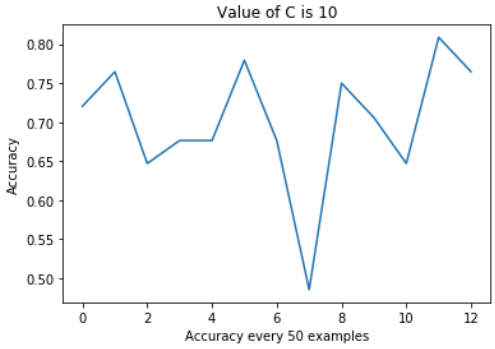
3. **For Second Relaxation Update of τ:** In Second Relaxation Update also same type of plots are observed which can be seen in figure 4. It is seen in figures 4a, 4b, 4c, 4d, **??** and **??** that it follows a general expected trend for the increase in accuracy levels after every 50 examples observed.
   There are though some exceptions observed with accuracy drops, this can be seen because of the outliers in the data that diverges the weights from the optimal values. It can be avoided by removing these outliers to get the expected trend in increase in accuracy.



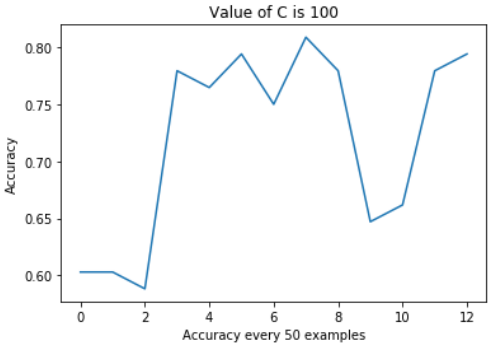
(a) C = 0.1



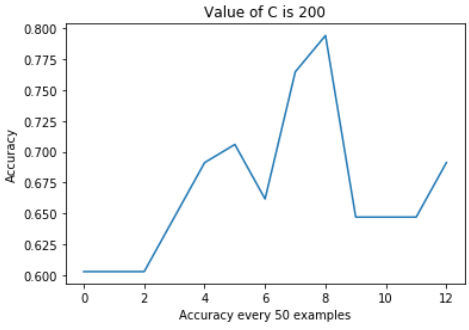
(b) C = 1



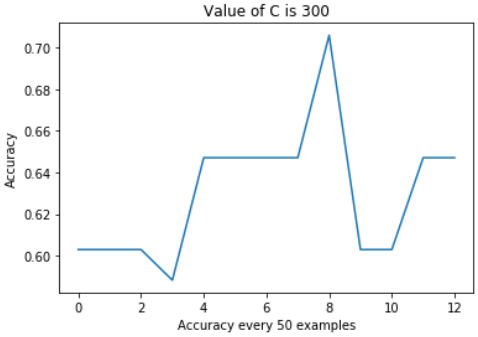
(c) C = 10



(d) C = 100



(e) C = 200



(f) C = 300

Figure 4: Accuracy per 50 examples for different values of C

It is also observed that the accuracy level increases with the value of C till value 100 and after that there is a drop in the accuracy observed.

2.3.2 Evaluation of accuracy of the model on all examples

1. **Classic Update of τ:** The accuracy of 0.6842857142857143 with time taken 0.632 seconds is observed for Classic Update on all examples.

2. **For First Relaxation Update of τ:** The figure 5 shows the accuracy obtained for different values of C. It is clearly observed that the accuracy increases till 0.1 value and than starts decreasing.

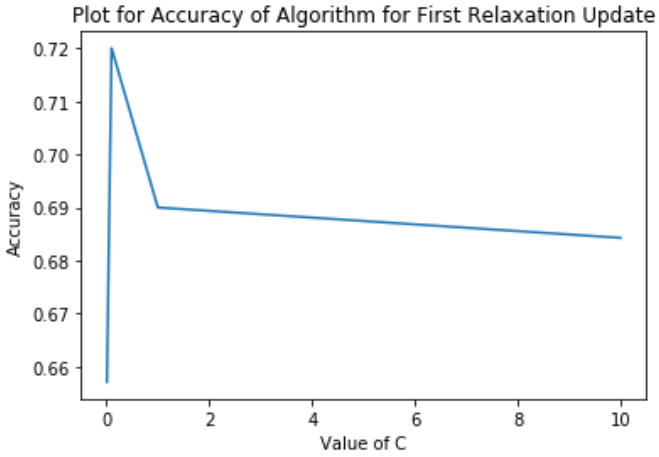


Figure 5: Plot for Accuracy of Algorithm for First Relaxation Update for different values of C

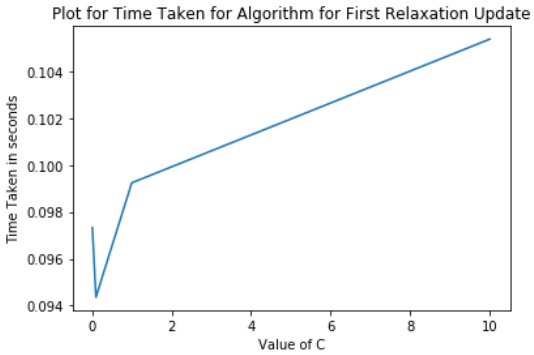


Figure 6: Plot for time taken for Algorithm for First Relaxation Update for different values of C

It can also be seen in the figure 6 that the time taken for the algorithm increases with the increasing value of C.

3. **For Second Relaxation Update of τ:** The figure 7 shows the accuracy obtained for different values of C. It is clearly observed that the accuracy increases till 10 value and than starts decreasing.

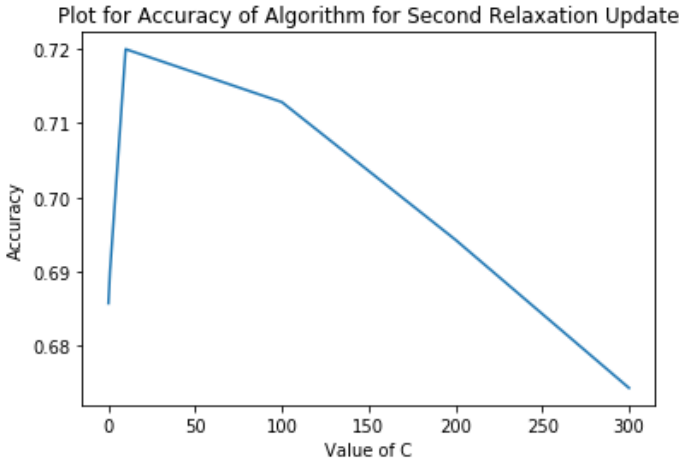


Figure 7: Plot for Accuracy of Algorithm for Second Relaxation Update for different values of C

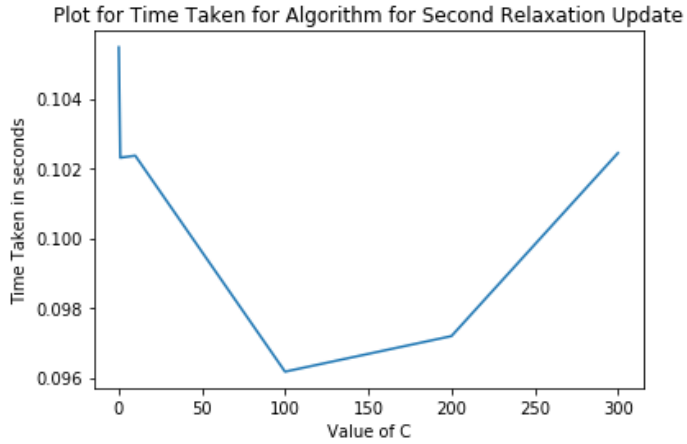


Figure 8: Plot for time taken for Algorithm for Second Relaxation Update for different values of C

It can also be seen in the figure 8 that the time taken for the algorithm decreases and than increases with the increasing value of C.

2.3.3 Evaluation of accuracy of the model on all examples after randomly flipping the labels

1. **Classic Update of τ:** The accuracy drops to 0.53 with time taken 0.118 seconds after the random flipping of the labels.

2. **For First Relaxation Update of τ:** The figure 9 shows the accuracy obtained for different values of C after random flipping of the labels. It is clearly observed that the accuracy decreases overall compared to before random flipping of the labels. The reason being the very fact that random flipping changed the distribution of the data, which deviates the algorithm to achieve optimal weights.

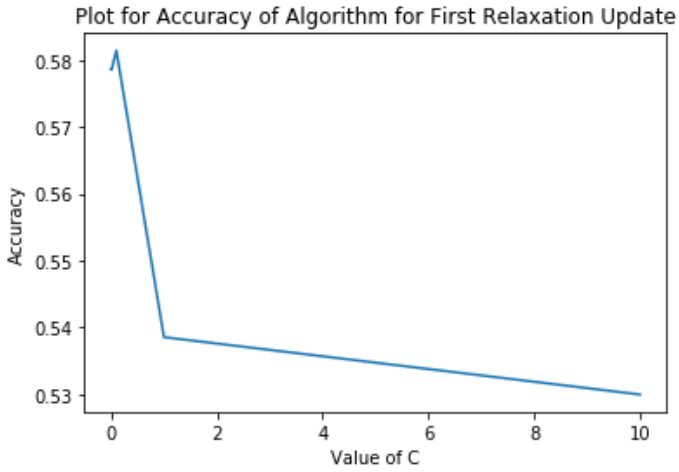


Figure 9: Plot for Accuracy of Algorithm for First Relaxation Update for different values of C after random flipping of the labels

3. **For Second Relaxation Update of τ:** The figure 10 shows the accuracy obtained for different values of C after random flipping of the labels. It is clearly observed that the accuracy decreases overall compared to before random flipping of the labels.

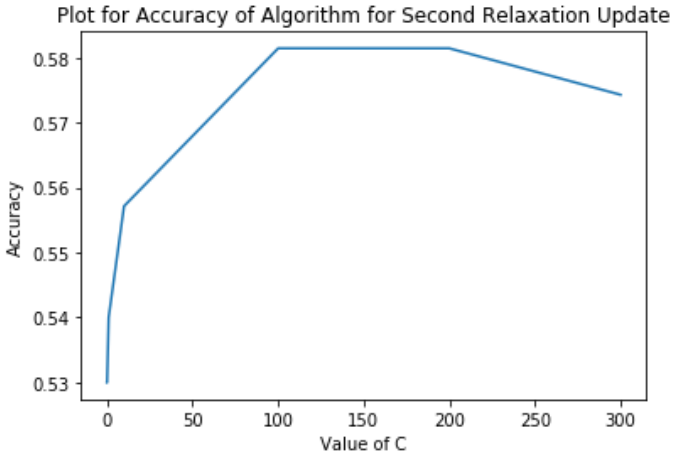


Figure 10: Plot for Accuracy of Algorithm for Second Relaxation Update for different values of C after random flipping of the labels

The reason being the very fact that random flipping changed the distribution of the data, which deviates the algorithm to achieve optimal weights.

2.3.4 Comparison of accuracy with Support Vector Machine

The Support Vector Machine gave an accuracy of 0.81, which is higher than Online Passive Aggressive but it can also be seen that the Passive Aggressive Algorithm gave consistent results with satisfactory accuracy for the test data.

2.4 Conclusion

The online passive aggressive is a robust algorithm for online learning with consistent good results. It was also seen that the Second and First Relaxation improved a results a lot compared to Classic Update. The random flipping of labels have a drastic impact on the accuracy of the model as it completely changes the distribution.

3 Bandits

One of the important problems that can be solved using the machine learning algorithms is Bandits problem. In this practical session, we will try to implement three of bandits algorithms, Incremental uniform, UCB and ϵ-Greedy.

The aim of this section is to give a experimental study to the different implemented algorithm.

A multi-bandit problems is described using the following architecture:

1. Actions : N the number of arms, in our case it will be 8 arms
2. Reward : is the reward we can get by choosing a given arm, mathematically, the reward is a function from $[\![1;n]\!]$ to 0,1

For every approach, we will provide o summary of the theoretical study, the architecture, the pseudo-code and the experimental results.

For the all implementations, we will use a Gaussian distribution with a mean = 0 and a standard deviation = 1.

At the beginning we fix a probability reward for each arm based on the normal distribution. To evaluate the taken action, we compare the fixed probability with a new generated value.

3.1 Incremental uniform

3.1.1 Theory study

Incremental uniform approach, can be seen as the most basic algorithm. where we test each arm with no difference between choosing two arms, and each arm is pulled once every iteration. whenever an arm is pulled, we take the reward and we calculate the average reward of each iteration. The accumulation of the average reward leads us to decide which arm is more accurate to give the best gains after M iterations.

To implement the described approach we followed the pseudo-code bellow.

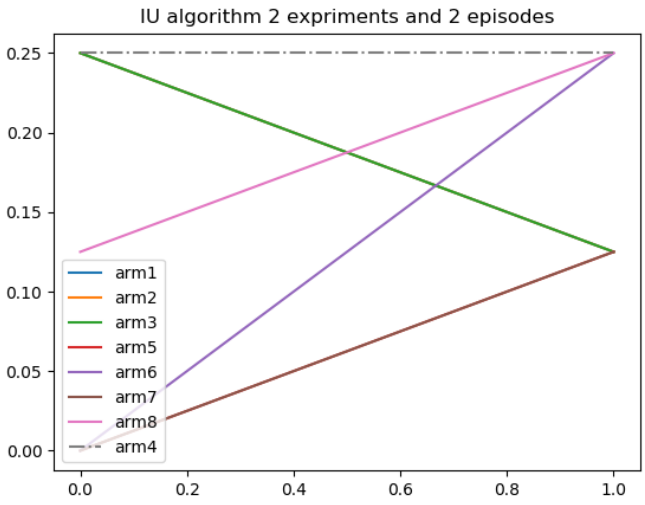
Algorithm 1 Incremental uniform

Input: N : number of the arms, M : number of iterations
Output: The best arm, the average reward
procedure
    $bandits\_probs \leftarrow [prob_i \text{ for } i \text{ in } range(N)]$
    $reward\_history \leftarrow [0 \text{ for } i \text{ in } range(N)]$
    $exp\_reward\_history \leftarrow [reward\_history \text{ for } i \text{ in } range(M)]$
    $exp\_reward\_arm \leftarrow [0 \text{ for } i \text{ in } range(N)]$
loop: for iter in range(M):
    For arm in range(N):
        $reward\_history[arm] \leftarrow reward \in \{0, 1\}$
    $exp\_reward\_history[iter] \leftarrow average(reward\_history)$
loop: for arm in range(N):
    $exp\_reward\_arm[arm] \leftarrow average(exp\_reward\_history[:, arm])$
return $argmax(exp\_reward\_arm)$

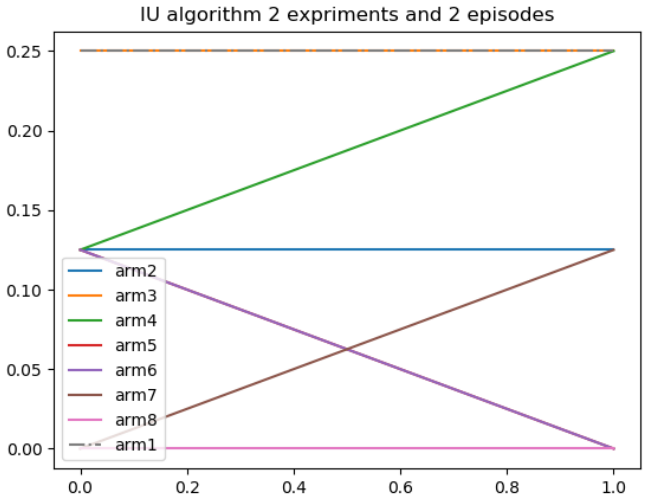
As we notice in the previous pseudo-code, there is no probabilistic difference between two arms, and the choice of an arm doesn't affect the next steps.

3.1.2 Experiments and results

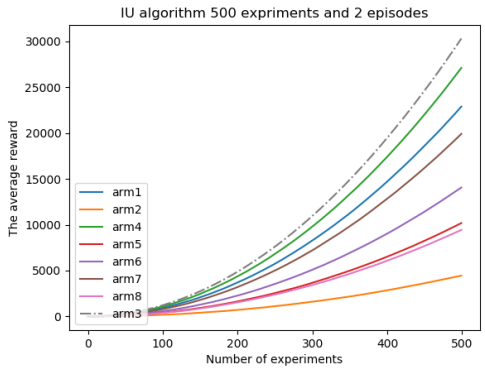
In this section we will study the experiments we test using the IU algorithm. In the following plots we illustrate the average reward got in every episode(sequence of pulling the 8 arms) of the algorithms. So that we can understand the impact of the number of iterations and the episodes.



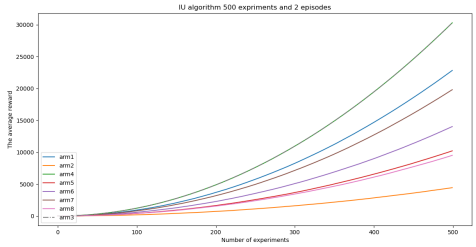
For this first experiment we start our problem with the following probability values for each arm, [1.11, 0.18, 2.20, 1.51, 0.42, 0.60, 0.91, 0.39].
In this graph we can notice that both arm 3 and 4 start with the same average reward, but at the final step the fourth one ends to be the best one with maximum average reward. In that case when the number of episode and experiments is low, the uniform incremental approach is not suitable, because it needs more iterations so that the arm with the greater probability make an impact on the average reward.
Another impact of the low number of experiments can be seen in the following experiments. For this one we used the same values as the previous experiment.
We can notice that instead of having the greatest starting probability value, the average reward of the third arm decrease, but for the fourth, it remains the same.
At the end of the experiment, we found that the average reward is the following [0.5, 0.25, 0.5, 0.375, 0.125, 0.125, 0]. The arm that is finished to be the best is the first one with an average value similar to the third arm, in that case the algorithm can not decide and ends by choosing the first one. We can conclude from these two experiments that for low numbers of experiments and episodes, the uniform incremental is not decidable and can lead to an arbitrary choice.



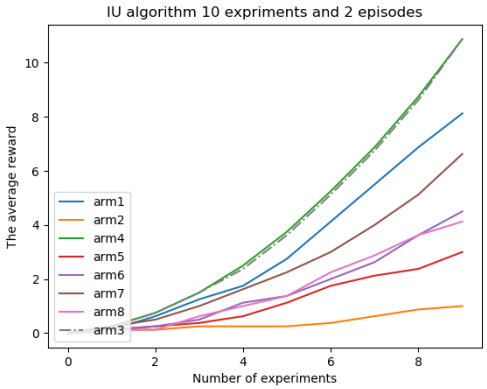
In the following experiments, we will increase gradually the number of experiments, and fix the number of episodes at 2.
From the following graph, we can notice the impact of the number of experiments, between 0 and 100 experiments, the difference of the average rewards of the arms is not very important, but after adding other experiments, we notice that the gap between these values becomes more and more important.



The algorithm can decide effectively which arm to choose even in the first steps , that is because from the beginning there is slit difference between arm 3 and arm 4. For this reason we assign to the arm 3 and 4 the same probability values.



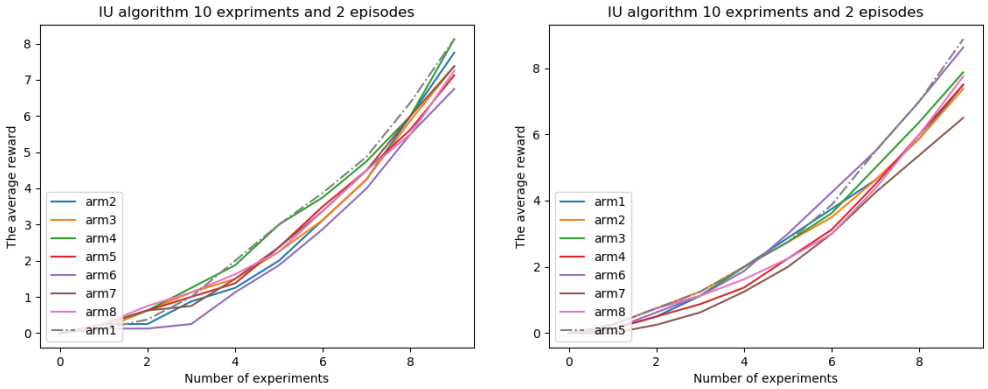
(a) 500 experiments



(b) 10 experiments

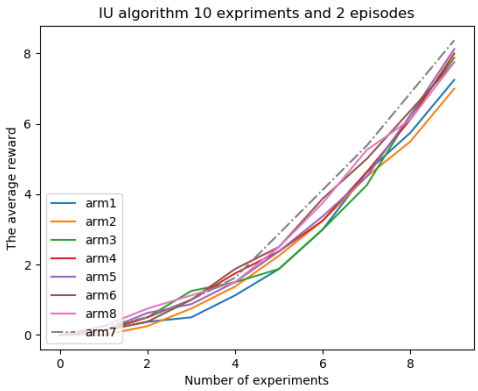
Figure 11: Impact of the number of experiments on the same probabilities

For large number of experiments, we notice that the both arms have a small difference of average reward, a difference that can not be taken in consideration to decide which arm is the best. To illustrate this result more better, we did experiments on small number of experiments, in this case it is absolutely not decidable which arm is the best, the following graphs illustrate this result.

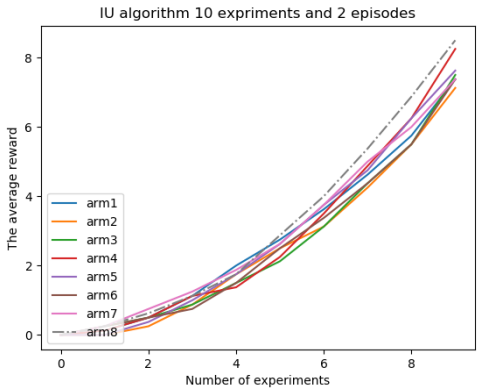


(a) best arm is 1

(b) best arm is 5



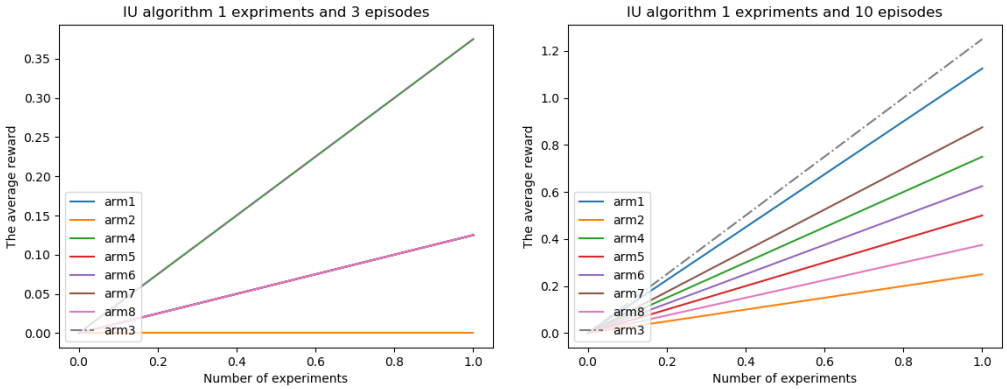
(c) best arm is 7



(d) best arm is 8

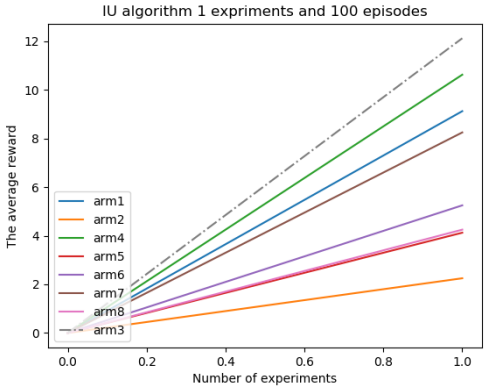
Figure 12: Impact of the number of experiments on the same probabilities

In this last section we will see the impact of the number of episodes on choosing the best arm.

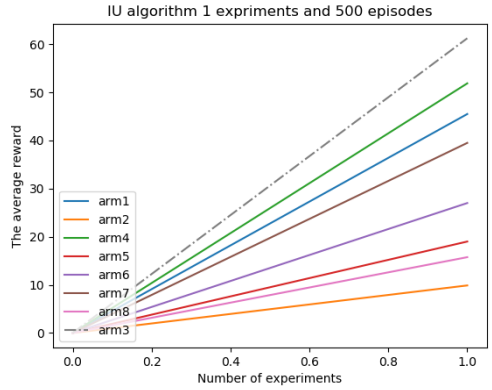


(a) Number of episodes : 3

(b) Number of episodes : 10



(c) Number of episodes : 100



(d) Number of episodes : 500

Figure 13: Impact of the number of episodes on the same probabilities

From the previous graphs, we notice that the number of episodes has the same impact as the number of experiments, but we notice that there a slit difference in term of time complexity, that is to say, even with high number of episodes, the algorithm is more powerful.

3.2 ϵ-Greedy

The third approach we used to solve the multi-bandit arms is the ϵ-Greedy algorithm. In this section we will present a small review on the theory behind and at the end we will present the experiments we played and the results.

3.2.1 Theory study

In the Incremental uniform approach, we saw that there is no difference in choosing an arm, but, for the greedy approach, we set a threshold ϵ, to decide how we can explore the game and choose the next arm. The pseudo-code used in this algorithm is similar to previous one, the only difference is on the way we choose the arm. To implement this approach we used the following pseudo-code.

Algorithm 2 ϵ-Greedy

Input: N : number of the arms, M : number of iterations, ep : ϵ value
Output: The best arm, the average reward
procedure
    $bandits\_probs \leftarrow [prob_i \text{ for } i \text{ in } range(N)]$
    $reward\_history \leftarrow [0 \text{ for } i \text{ in } range(N)]$
    $exp\_reward\_history \leftarrow [reward\_history \text{ for } i \text{ in } range(M)]$
    $exp\_reward\_arm \leftarrow [0 \text{ for } i \text{ in } range(N)]$
loop: for iter in range(M):
    $action \leftarrow get\_action(N, ep)$
    $reward \leftarrow get\_reward(action)$
    $update\_Q(action, reward)$

Algorithm 3 get_action function

Input: N : number of the arms, ep : ϵ value
Output: The arm to pull
procedure
    $rand \leftarrow$*random probability value*
    if $rand < ep$: *return random_arm*
    else : return $argmax(Q\_table)$

Algorithm 4 get_reward function

Input: probs : initial probability vector, arm : the chosen arm
Output: The reward
procedure
    $rand \leftarrow$*random probability value*
    return 1 if rand < probs[arm] else 0

Algorithm 5 update_Q function

1: *Input*: Q_table, K_table, arm, reward
2: *Output*: updated Q_table
3: **procedure**
4: $K\_table[arm] \leftarrow K\_table[arm] + 1$
5: $Q\_table[arm] \leftarrow Q\_table[arm] + \frac{1}{K\_table[arm]} * (reward - Q\_table[arm])$

The K_table contains the number of times each arm is pulled

3.2.2 Experiments and results

The experiments we lead to test this approach were the following:

1. The impact of the epsilon value with fixed probabilities
2. The impact of the number of experiments and episodes

At the beginning, we fixed the number of experiment to 50 and episodes to 10. And we change the ϵ values from 0.1 to 1.0.

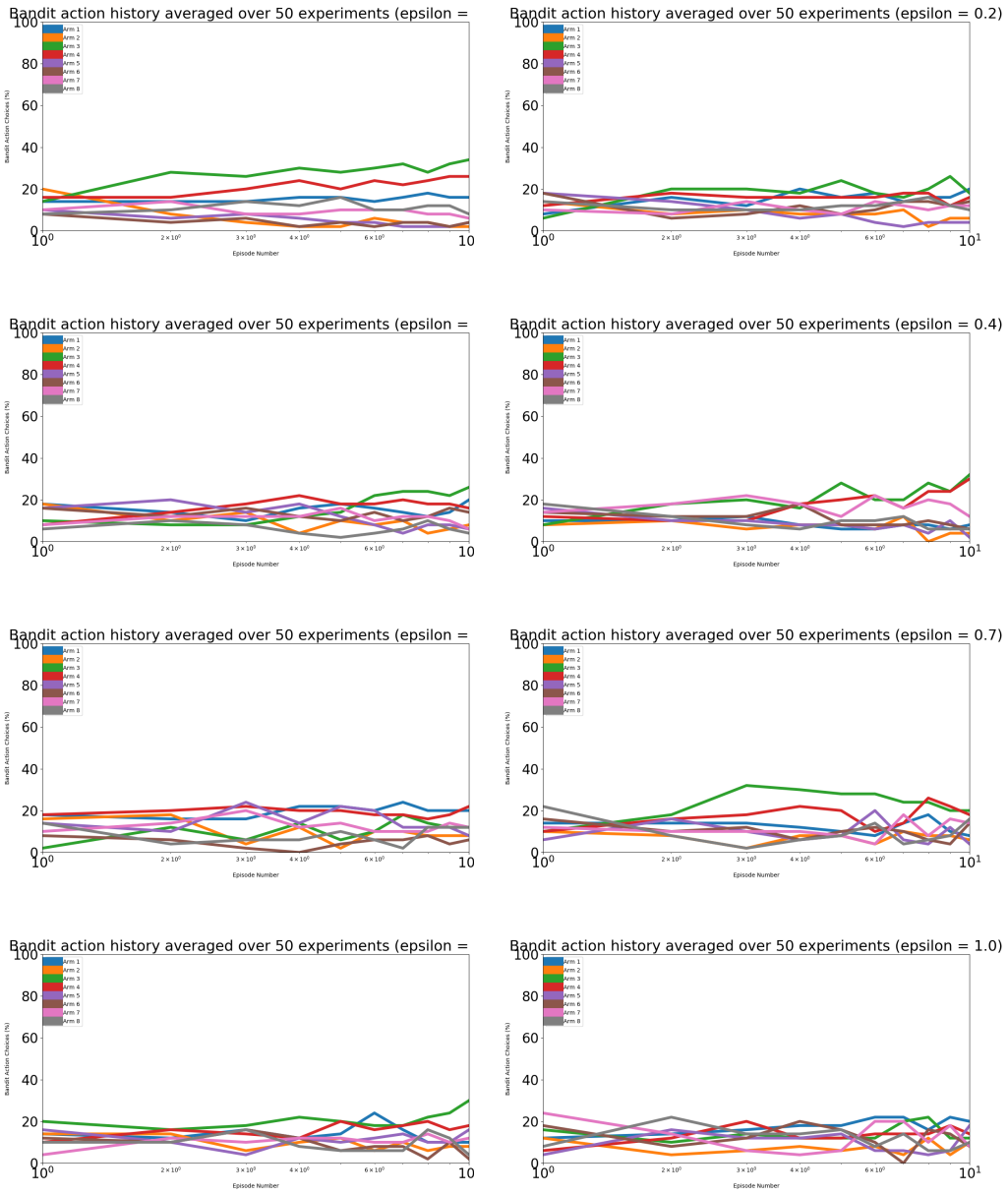


Figure 14: Impact of the number of episodes on the same probabilities

We can notice that for as far as the ϵ increase, we start loosing the decidable aspect of the algorithm, and the majority of arm end to have the same average reward and same probability to be the winning arm. we got

also the same results with small values of ϵ ($\epsilon < 0.01$), bellow this value, we had to increase both the value of experiments and tries in each experiment. To study the impact of the number of experiments, and the episodes in each experiment, we fixed $\epsilon = 0.05$.
The following graphs illustrate the results we got through these experiments.

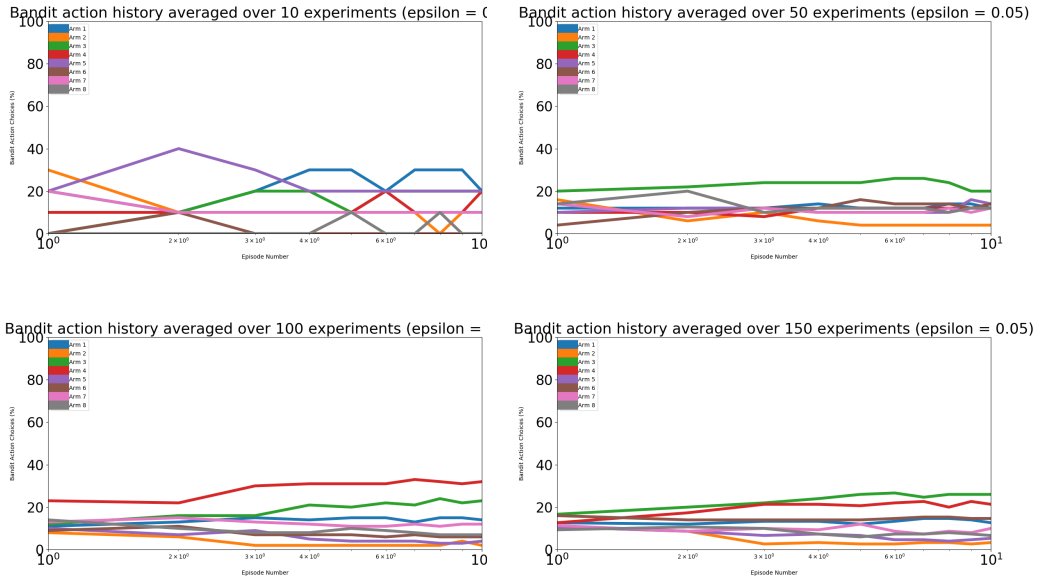


Figure 15: Impact of the number of episodes on the same probabilities

From these graphs we can deduce the important role that the number tries in each experiment. That is because, even with a good value of ϵ and high number of experiments, we still have some confusion in deciding which arm is to choose. So we increased the number of the episodes and obtain a final graphs

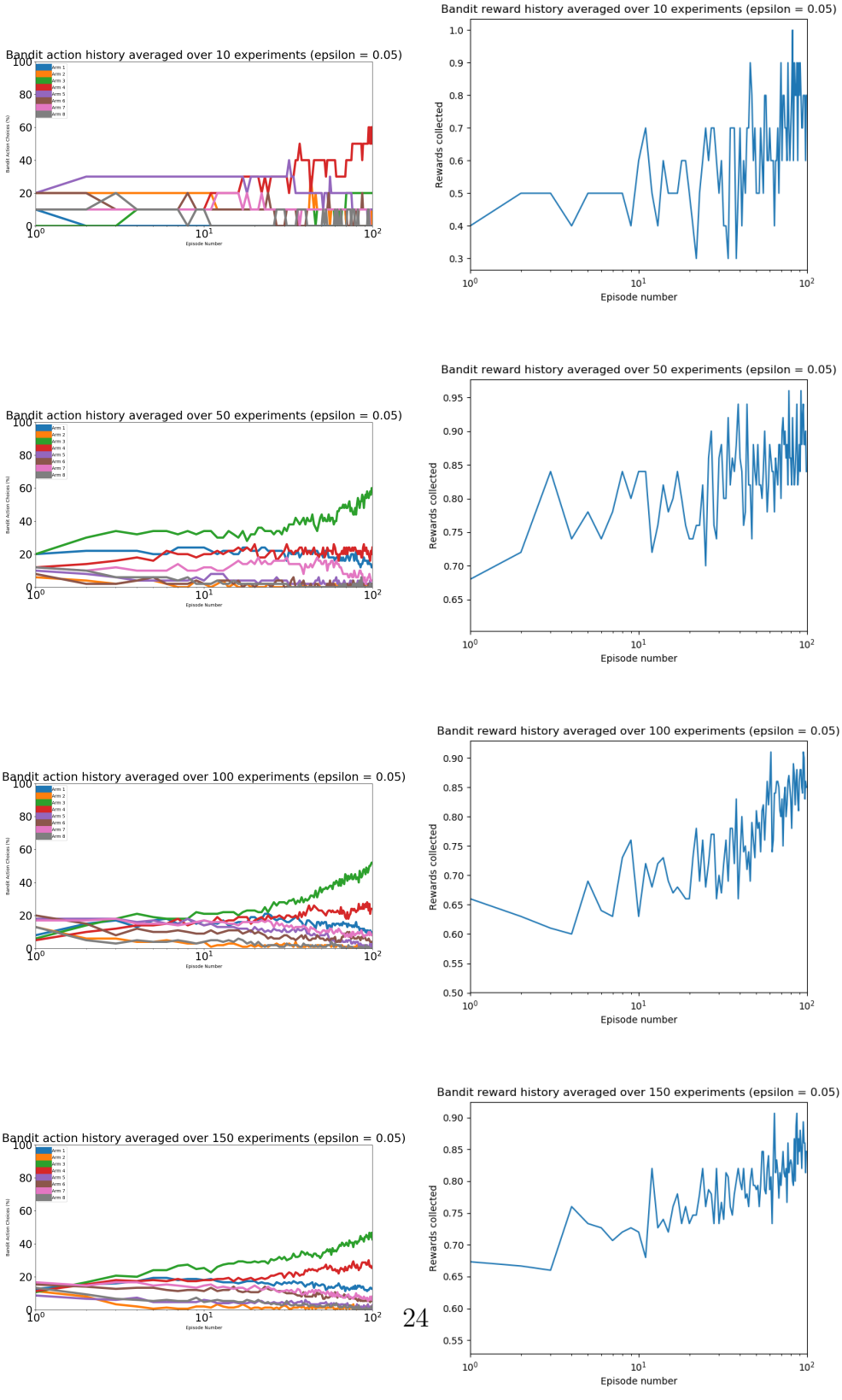


Figure 16: Impact of the number of episodes on the same probabilities

As we notice, the average reward increases in all cases, but in first experiment, for the algorithm the best arm is 4, while the arm with the best chance the be the winner is 3. As far as the number of experiments and the number of tries increase, the algorithm become more decidable and converge to the arm with high possibility to be the winner.

3.3 UCB

In this approach [1] we introduce a new concept of choosing the best arm for multi-bandit problem. Instead of using a probabilistic approach as we saw in the two last algorithms, we use the UCB algorithm which is considerate as a deterministic approach. In this section we will explain how the UCB works theoretically and we will present experiments we got by using this approach.
For the upper confident bound algorithm, each arm has a probabilistic distribution, the more the variance of the distribution is important the more the chance to explore the arm is important. Following this idea, we got certain if the pulled arm is leading to maximize the reward or not, while with the other approaches we prefer to pull the arm with high probability and forget the other choices.
After making a choice, we calculate the new mean reward of the distribution and the new bound, the bellow picture illustrate the changes after this process in the bounds.

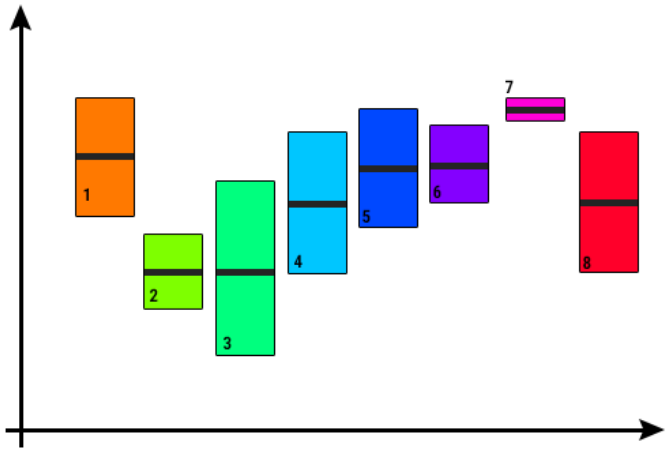


Figure 17: UCB bound and mean value presentation

In this case the algorithm will explore the third arm, because is has the largest confident bound. After M tries of the experiments, we get at the final step the best arm to explore,, the one having the greatest average reward. To implement this algorithm we used the following pseudo-code:

Algorithm 6 UCB

Input: N : number of the arms, M : number of experiments
Output: The best arm, the average reward
procedure
    $bandits\_probs \leftarrow [prob_i \text{ for } i \text{ in range(N)}]$
    $reward\_history \leftarrow [0 \text{ for } i \text{ in range(N)}]$
    $number\_selec \leftarrow [0 \text{ for } i \text{ in range(N)}]$
loop: for iter in range(M):
    $max\_bound = 0$
  For arm in range(N):
    $if number\_selec[arm] > 0 :$
      $avg\_reward \leftarrow reward\_history[arm]/number\_selec[arm]$
      $bound \leftarrow sqrt(1.5 * log(iter + 1)/number\_selec[arm])$
      $upper\_bound \leftarrow avg\_reward + bound$
    $else :$
      $upper\_bound \leftarrow \infty$
    $if max\_bound < upper\_bound :$
      $max\_bound \leftarrow upper\_bound$
      $selec\_arm = arm$
  $number\_selec[selec\_arm] \leftarrow number\_selec[selec\_arm] + 1$
  $reward \leftarrow get\_reward(bandits\_probs, selected\_arm)$
  $reward\_history[selec\_arm] \leftarrow reward\_history[selec\_arm] + reward$
(return max(reward_history),argmax(reward_history))

For the get_reward function it is similar to the we used in the previous approaches.

3.3.1 Experiments and results

The experiment we conducted for this method is targeting the average reward in each experiment, and try various of number of experiments.

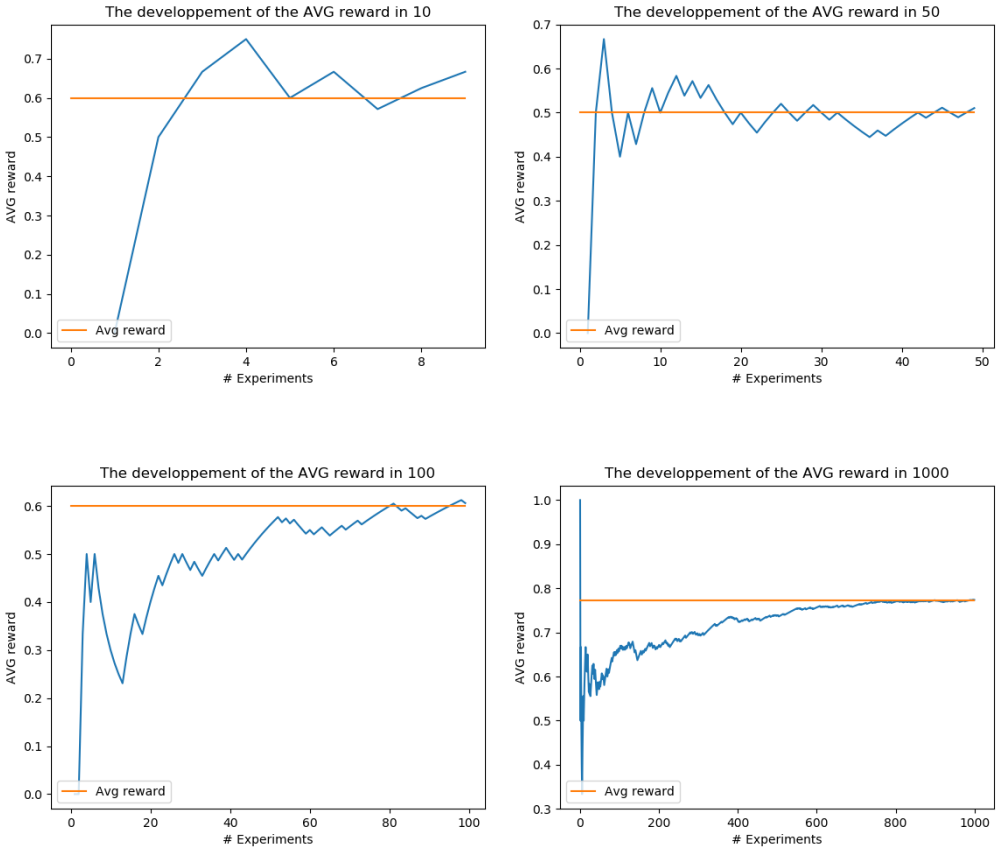


Figure 18: The development of the average reward UCB

As we can notice, through these experiments that the average reward is not monotone at he beginning of the experiments, this because the the algorithm tend to test arms with high bounds without taking in consideration the average reward. But, after many tries, we observe that the development of the avg reward converge as far as we have many experiment tries.

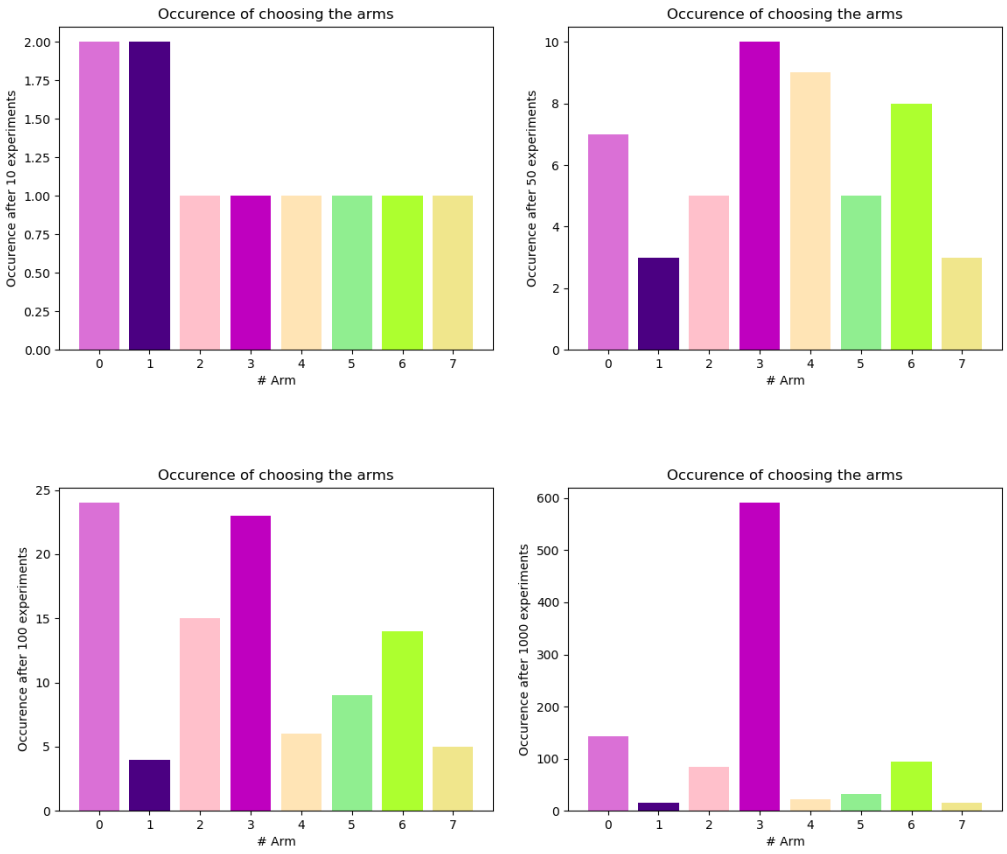


Figure 19: Occurrences of chosen bars

From the above graphs, we notice that for small experiments tries even the arms with small starting probabilities are explored, that is to say, the algorithm check every possibility to gain information about the arms. For this reason, when we run the algorithm for high number of tries we notice that the UCB algorithm get more confidence on choosing the one arm and increase the average reward as we saw before.

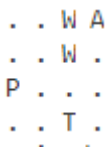
4 Reinforcement Learning

4.1 problem presentation

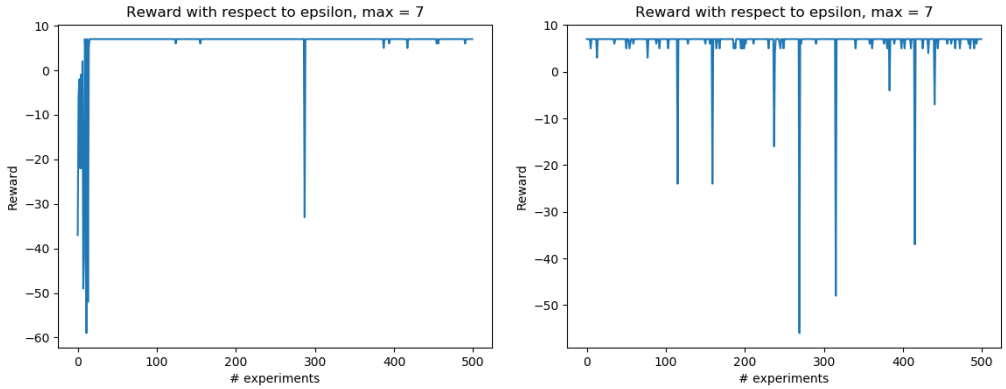
For the reinforcement section, we aim to implement algorithm to solve an atari game of labyrinth where we should find the path leading to maximize the reward of the game.
To do this, we got inspired by a work done by [2], using the ϵ-Greedy approach to do the Q-learning.

4.2 Experiments

For the experiment, we first of all build our game grid, wish remains static for all the experiments.

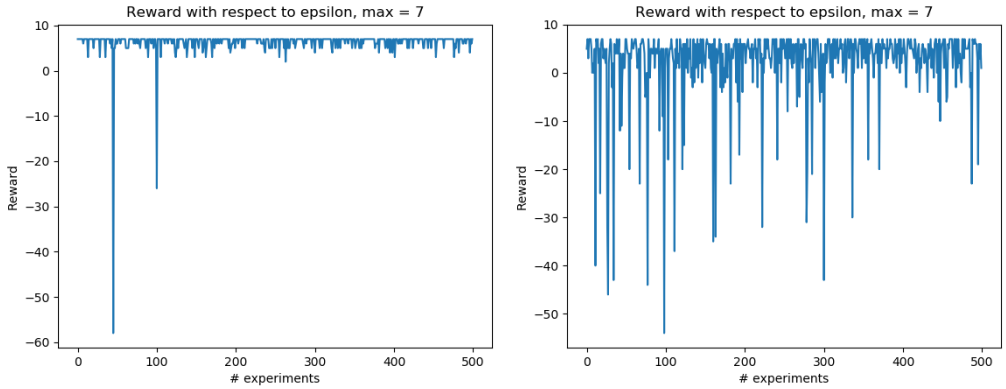


To evaluate this algorithm we know that the maximum reward would be 7, for this reason, we targeted the development of the reward with respect to different ϵ-values.



(a) $\epsilon = 0.01$

(b) $\epsilon = 0.05$

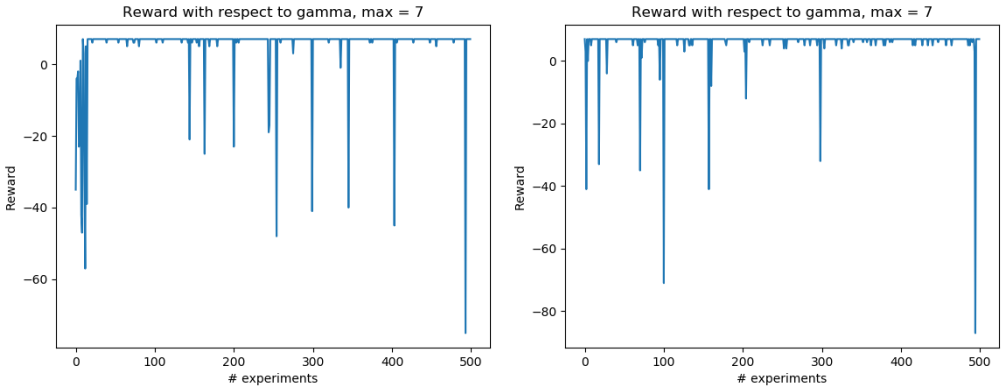


(c) $\epsilon = 0.1$

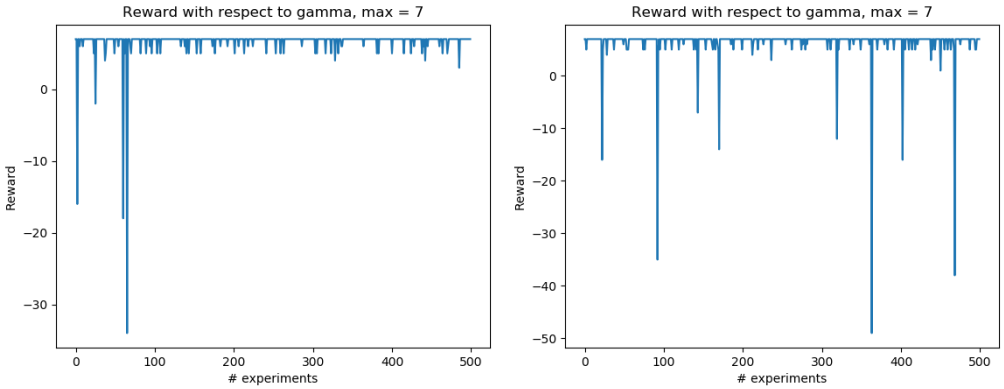
(d) $\epsilon = 0.5$

Figure 20: Changes of the reward with respect to ϵ

As we can notice with small values of ϵ we got the maximum reward much faster that big ones. After, we test the values of γ and fix ϵ to 0.05



(a) $\gamma = 0.01$ (b) $\gamma = 0.05$



(c) $\gamma = 0.1$ (d) $\gamma = 0.5$

Figure 21: Changes of the reward with respect to γ

Also for the γ parameter we notice the remark, for big γ values we don't get any results and the algorithm keep training, while for small values we got the exact result in the first tries of the training.

References

[1] Ankit Choudhary. "Reinforcement Learning Guide: Solving the Multi-Armed Bandit Problem from Scratch in Python". In: URL: https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/.

[2] Venelin Valkov. "Solving an MDP with Q-Learning from scratch". In: URL: https://medium.com/@curiousily/solving-an-mdp-with-q-learning-from-scratch-deep-reinforcement-learning-for-hackers-part-1-45d1d360c120.