ESE 545 - Project 3
Recommendation Systems and Clustering

Tejas Srivastava

May 03, 2020

1 Module 1 - Recommendation System

1.1 Problem 1

Analysis of data and problem formulation as a Multi Armed Bandit Problem

1. The dataset consists of 1005 rows representing the movies and 14999 columns representing a time measure(or day). Each row of the dataset corresponds to a movie and each column corresponds to a day. On a given day, 1 signifies that some user has given high rating to that movie and similarly, 0 signifies low or no rating.
2. In terms of formalizing the problem as a Multi Armed Bandit Problem, each movie is considered as an arm, and the time step is an iteration. The movie to be recommended by the recommendation system on a given day is a parallel to the best arm selected by solving the multi arm bandit problem at a given iteration which maximizes the expected gain, with the partial (or full) knowledge of each choice's output at the end of the day after the best arm has been selected (or best movie has been recommended).
3. Thus the goal is to design a recommendation system which recommends movies in an online manner by achieving the lowest possible regret over a given time period. With each recommendation the system makes, the system receives a reward (either 0 or 1), which is used to calculate the regret for all the algorithms.

1.2 Problem 2

Algorithms based on Partial Feedback
We try to solve the problem using classic exploration exploitation strategies in order to achieve minimum regret, but with only Partial feedback in this section i.e. only observing the rewards of the movie we choose or recommend at a given time / day. We use both stochastic and non stochastic algorithms(non deterministic) to solve the problem.

Note on Regret Calculation

Here, for calculation of instantaneous regret of the algorithm at any given timestep/day, we find the difference of reward of the best arm so far and the reward of the arm selected on the current day. Also, we use the full feedback information till the given day to calculate reward of the best arm so far (with the maximum reward till that day) on any given day, and use only partial expected reward to calculate the reward for the selected arm on the current/given day, since the algorithms we try in this section are Partial feedback algorithms, and **do not** use full feedback information to pick the selected arm / recommendation at any given day.

$$regret_t = \mu^* - \mu_{it}$$

where, μ^* is the expected (full feedback) reward for the best arm so far and μ_{it} is the (partial feedback) reward of the selected arm 'i' in the 't' iteration by the algorithm.
Thus, the total regret for each algorithm is nothing but the sum of the instantaneous regrets at all times 't' i.e.

$$TotalRegret_T = \sum_{s=1}^{T} (\mu^* - \mu_{is})$$

We will be trying to design no regret algorithms, i.e. algorithms which have sublinear regret in terms of T, i.e. $\frac{regret_T}{T} \rightarrow 0$

1.2.1 Epsilon Greedy Algorithm (Stochastic)

- Epsilon greedy is the simplest algorithm which conditionally tries to exploit the best arm found so far with a probability ϵ and explores new options to pick with a probability $1 - \epsilon$.

- To perform the exploration step, the algorithm randomly selects an arm from a uniform distribution of values $1 \rightarrow k$, from 'k' arms. To perform the exploitation step it selects the arm with the maximum expected partial reward till time step 't'.

- An important hyperparameter, in this algorithm is the choice of ϵ, since it decides the extent of exploration and exploitation that the algorithm undertake. A smart (and beautiful) trick to decide ϵ is to take ϵ as inversely proportional to some power of 't' (the current iteration index), thus resulting in large ϵ, resulting in more exploration in the starting iterations and as the number of iterations increase, thus decreasing the value of ϵ thereby trying less explorations, and trying to exploit the best arms found so far.

- During implementation of the algorithm, I initially used a value of $\frac{1}{t}$ for ϵ, but found out the value to be very less, thus resulting in very less exploration, and majorly resulting only in exploitation throughout, thus I tried to use multipliers for trying to find a good value of ϵ. Thus we used the values $\frac{1}{t}$, $\frac{10}{t}$, $\frac{100}{t}$ and $\frac{1000}{t}$ for the value of ϵ, to find the best value of ϵ. The results for the trials are as follows:

- The instantaneous regrets for different values of epsilon:

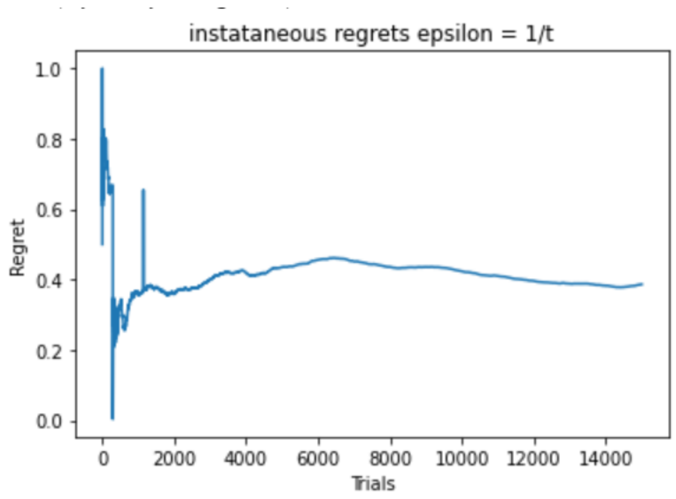

Figure 1: regret, epsilon = 1/t

–

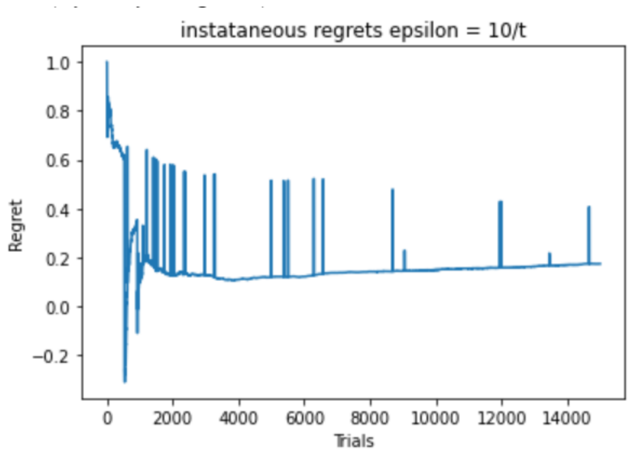

Figure 2: regret, epsilon = 10/t

–

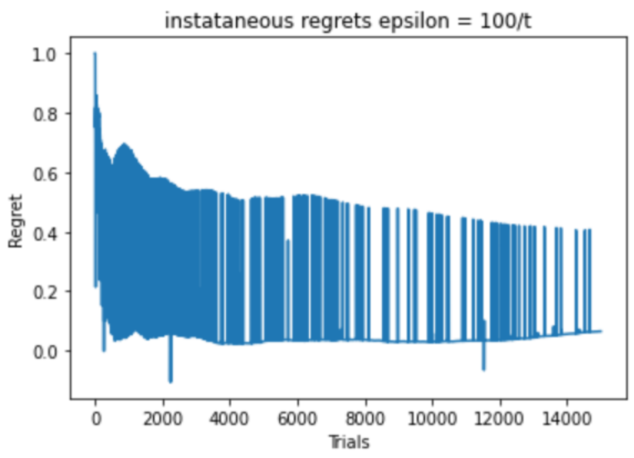

Figure 3: regret, epsilon = 100/t

–

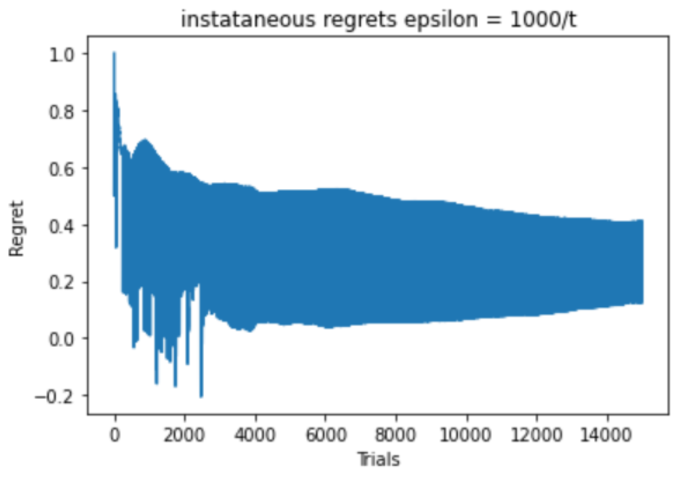

Figure 4: regret, epsilon = 1000/t

–

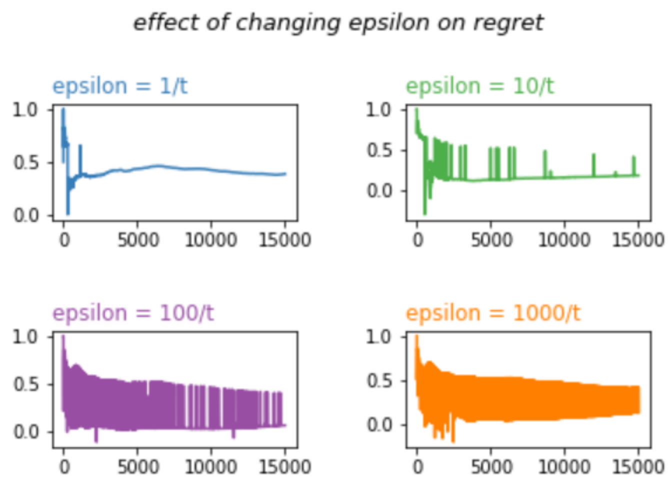

Figure 5: Comparison of all values

–

- Thus, we observe as we increase the value of epsilon the algorithm tries to explore more, and thus we can see a lot of exploration (resulting in noisy and oscillating plots) for high values of ϵ such as 100/t and 1000/t. But this does not give us a measure of best epsilon, thus we plot the cumulative regret to make a comparison.

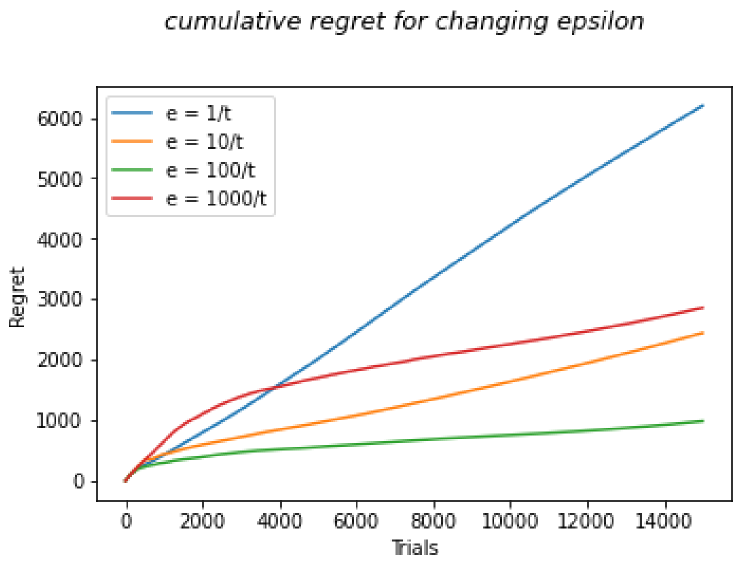

Figure 6: Comparison of cumulative regrets over time for all epsilons

- We also plotted the cumulative losses for these values of ϵ for this algorithm. The plot for loss is as follows:

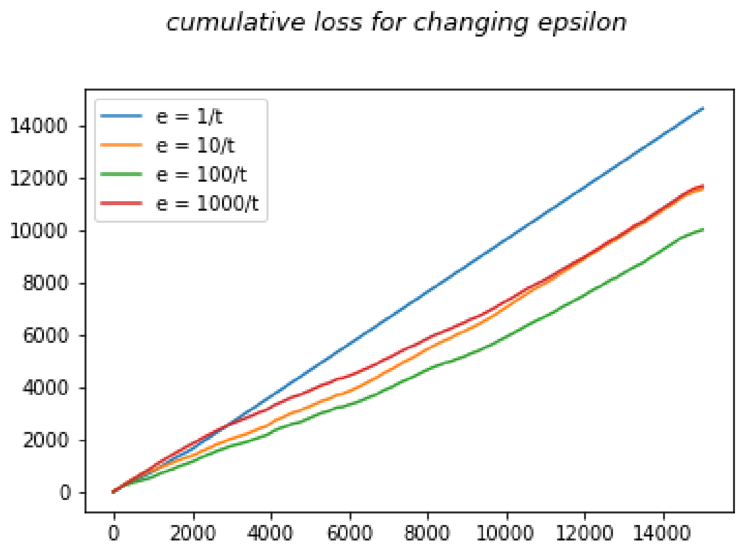

Figure 7: Comparison of cumulative losses over time for all epsilons

- Thus, we find that the value of $\epsilon = \frac{100}{t}$ gives us the minimum regret and minimum loss over time and thus we choose this value for our ϵ greedy algorithm.
- We also plotted the frequency of reward being 1 for the algorithm to observe the behaviour of each algorithm. The plot for epsilon = 100/t is as shown.

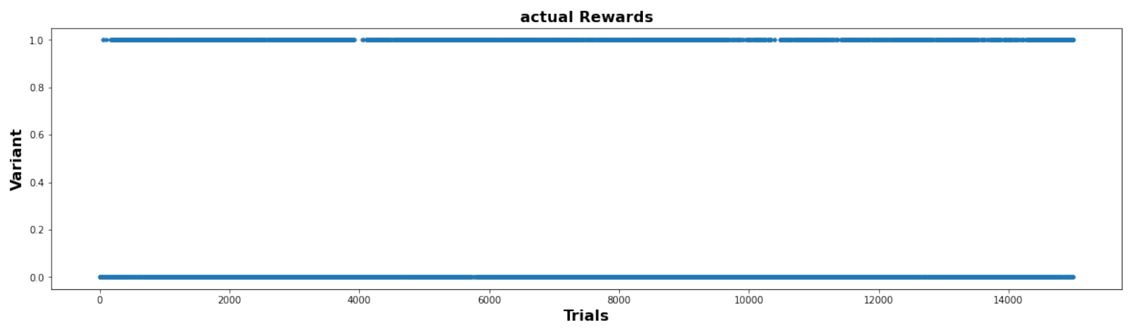

Figure 8: frequency of reward =1

- Thus, we observe that the algorithm explores a little in the start, and exploits for most of the duration of the algorithm.
- Total rewards over 14999 iterations with $\epsilon = 100/t$: 5895.0
- total regrets over 14999 iterations with $\epsilon = 100/t$: 580.1548900579286

1.2.2 UCB1 Algorithm (Stochastic)

- UCB1 algorithm is an optimistic algorithm which considers the Upper Confidence Bound of the expected reward of each arm to decide the arm to select in the current iteration (thus called optimistic). An estimate of the upper confidence bound is made by using the Hoeffding's inequality.

- The Hoeffding's inequality states that

$$Pr(|\mu - \hat{\mu_m}| \geq b) \leq 2e^{-2b^2m}$$

, where μ is the true mean of m i.i.d random variables, and $\hat{\mu_m}$ is the empirical mean.

- Considering δ to be a very small number, if we choose b = $\sqrt{\frac{1}{2m}\log\frac{2}{\delta}}$, then

$$Pr(|\mu - \hat{\mu_m}| \geq b) \leq \delta$$

- Thus, we choose different values of δ to find the best version of this algorithm. Here, we try the values $1/t^8$, $1/t^4$ and $1/t^2$ to get a tighter bounds of upper confidence on expected rewards.

- For each value of δ UCB is calculated as follows:
  - $\delta = 1/t^4$, $UCB_i = \hat{\mu_i} + \sqrt{\frac{2\log 2t}{n_i}}$
  - $\delta = 1/t^2$, $UCB_i = \hat{\mu_i} + \sqrt{\frac{\log 2t}{n_i}}$
  - $\delta = 1/\sqrt{t}$, $UCB_i = \hat{\mu_i} + \sqrt{\frac{\log 2t}{2n_i}}$
  - $\delta = 1/t^{0.25}$, $UCB_i = \hat{\mu_i} + \sqrt{\frac{\log 2t}{4n_i}}$

- To find the best value of δ, we try to plot the regrets, the instantaneous regrets for different values of delta:

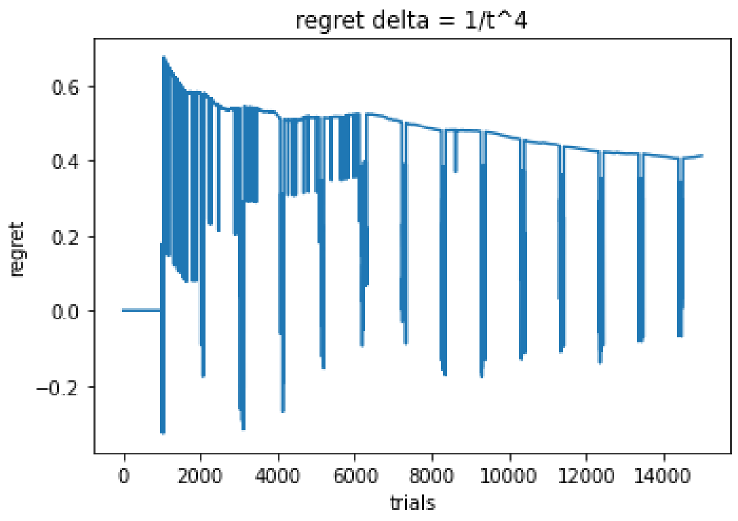

Figure 9: regret, delta = 1/t^4

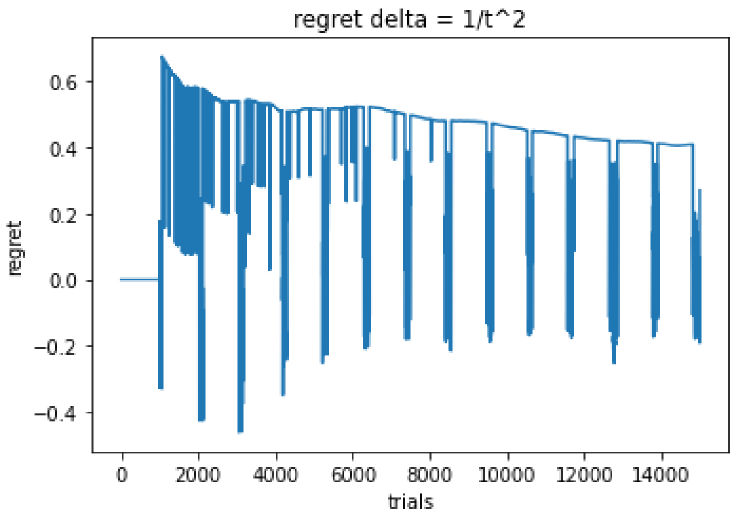

Figure 10: regret, delta = $1/t^2$

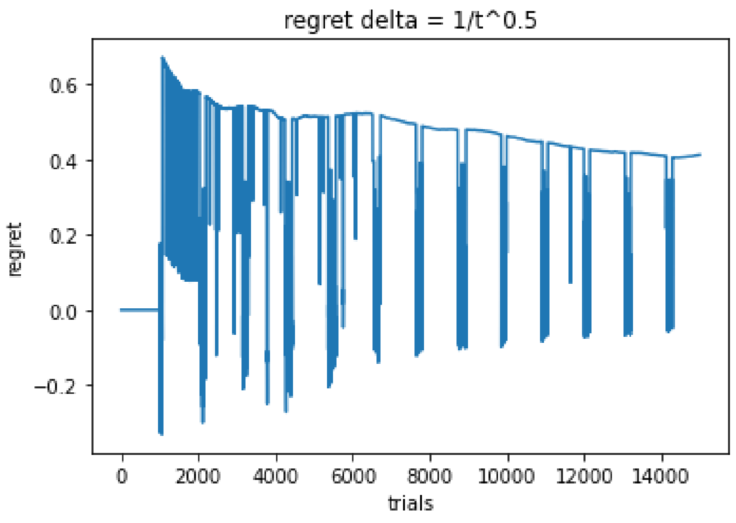

Figure 11: regret, delta = $1/t^{0.5}$

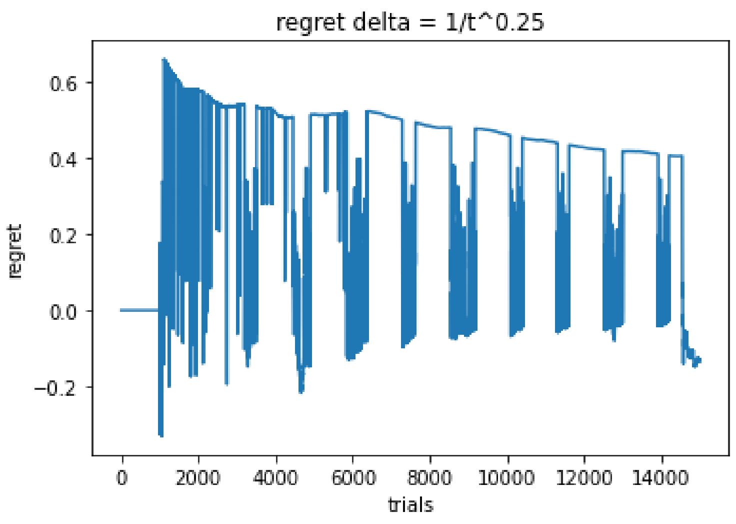

Figure 12: regret, delta = $1/t^{0.25}$

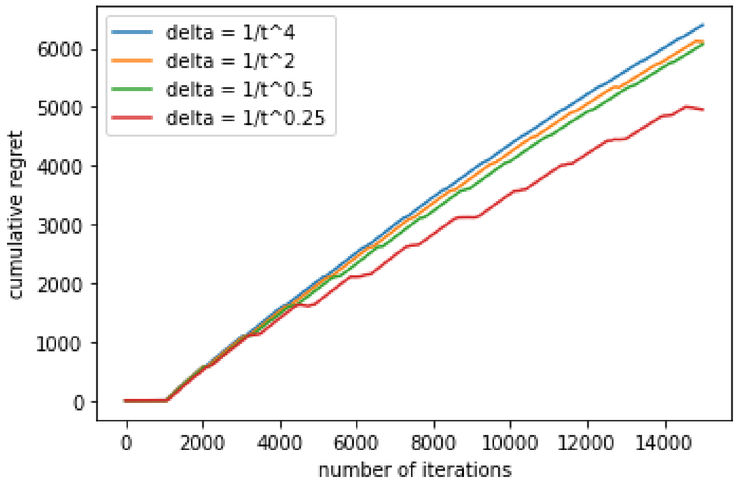

Figure 13: Comparison of cumulative regrets

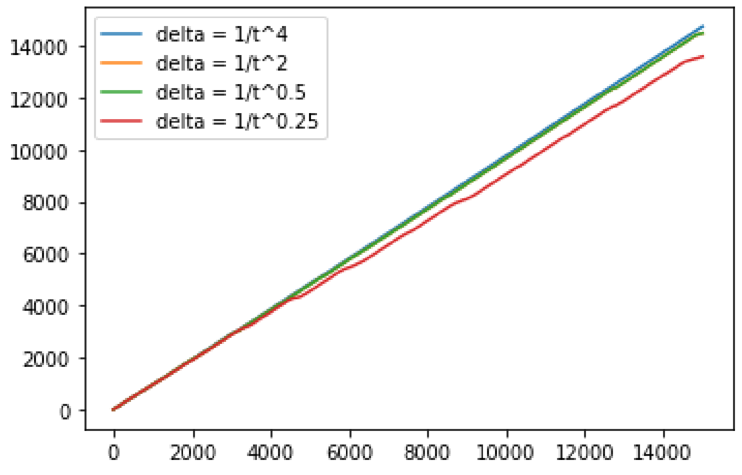

Figure 14: Comparison of cumulative loss

- Thus, we realize that changing the value of delta does not have much effect on the cumulative regret or loss, but we still choose $\delta = \frac{1}{t^{0.25}}$ as it gave us the minimum regret and loss.
- We also plot the cumulative loss and regret of UCB algorithm to compare it with the epsilon greedy algorithm.

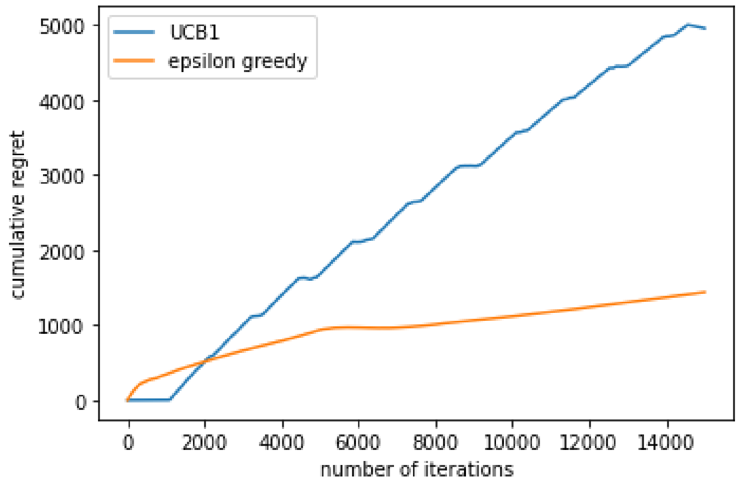

Figure 15: Comparison of regret of epsilon greedy and UCB1

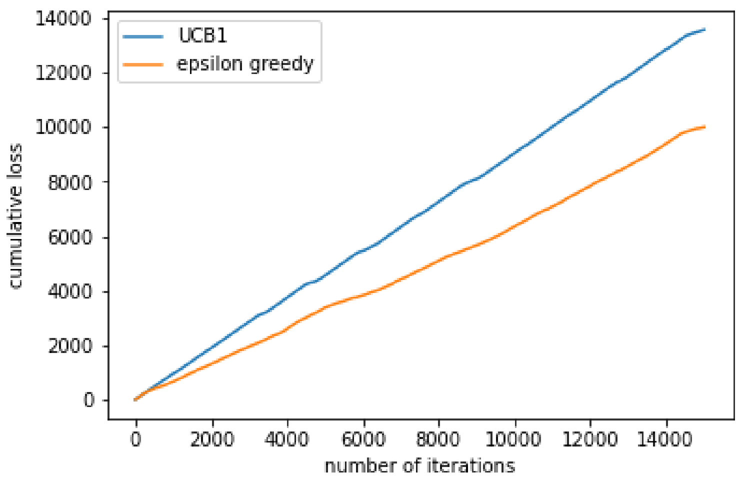

Figure 16: Comparison of loss of epsilon greedy and UCB1

- We also plotted the frequency of reward being 1 for the algorithm to observe the behaviour of each algorithm. The plot for UCB1 with $\delta = 1/t^{1/4}$ is as shown.

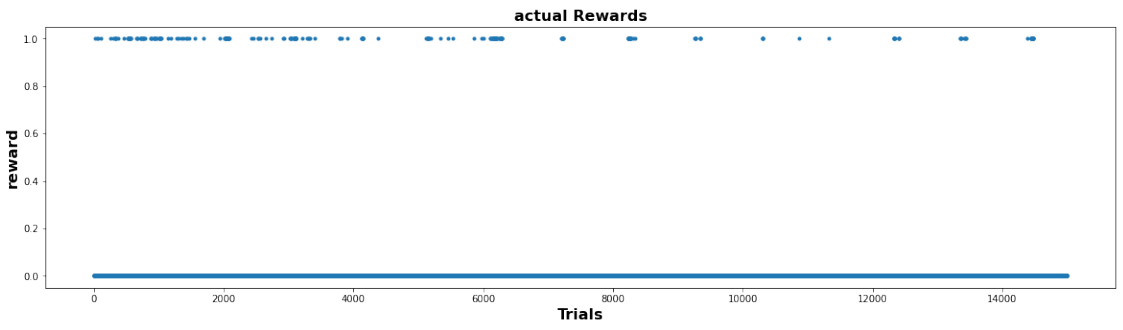

Figure 17: frequency of reward =1

- Total rewards over 14999 iterations: 1419.0
- total regrets over 14999 iterations: 4954.521027452012

1.2.3 Thompson Sampling (Stochastic)

- This is a bayesian method which is not a greedy algorithm and its exploration is more sophisticated.
- We start with a uniform prior distribution between 0 and 1 (Beta(1,1)) for each arm's expected reward, draw a parameter theta from the each k's posterior distribution. Select the arm k that is associated with the highest parameter theta. Observe the reward and update the beta distribution parameters according to the observed reward.

- We consider two parameters for the beta parameter as S and F, both of which are initially 1 for all the arms, when a given arm (say m) is selected, the reward is observed, if the reward is 1, S is incremented, if the reward is 0, then F is incremented. And further next best theta is drawn from this updated distribution for the current arm, whereas the same distribution for the rest of the arms, since we are implementing a partial feedback algorithm here.
- The instantaneous regret of Thompson Sampling is as follows:

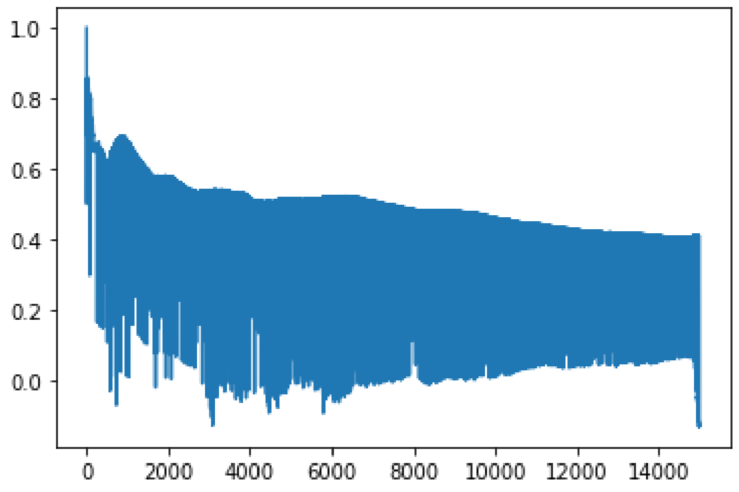

Figure 18: Instantaneous regret Thompson Sampling

- Comparison of cumulative regrets of Thompson Sampling with UCB1 and epsilon greedy is as follows::

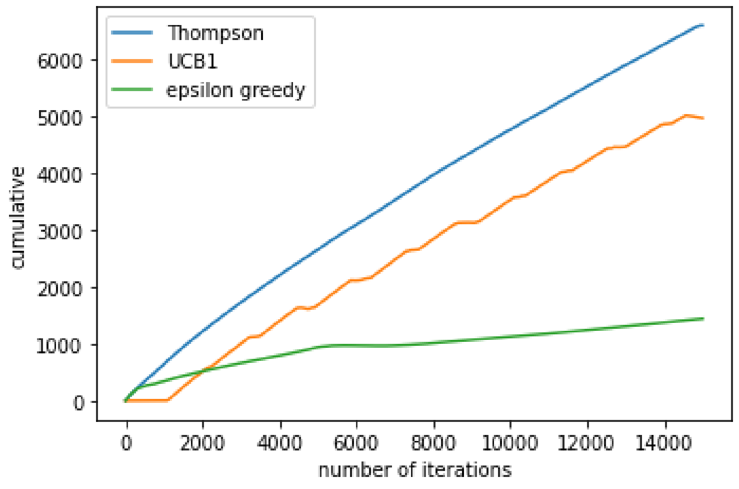

Figure 19: Cumulative regret

- Comparison of cumulative losses of Thompson Sampling with UCB1 and epsilon greedy is as follows::

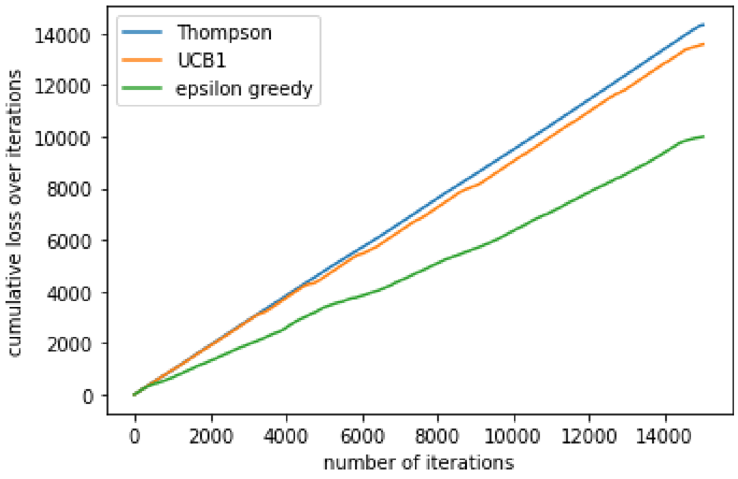

Figure 20: Cumulative loss

- We also plotted the frequency of reward being 1 for the algorithm to observe the behaviour of each algorithm. The plot for Thompson Sampling is as shown

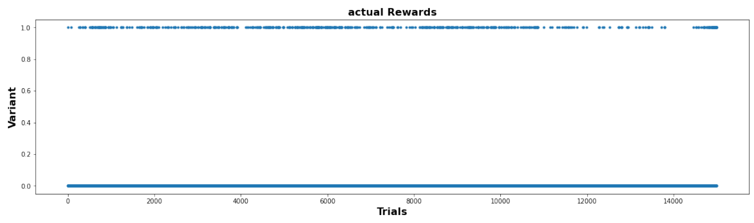

Figure 21: Frequency of reward being 1 for Thompson Sampling

- Total rewards over 14999 iterations: 714.0
- Total regrets over 14999 iterations: 6580.0284133257655

1.2.4 EXP 3 - Exponential Weights for Exploration and Exploitation (Non Stochastic)

- This is a probabilistic random algorithm which even performs good when there is an adversary which can manipulate the losses, as the randomness helps in restricting the adversary's power in manipulating the losses.
- The algorithm uses a probability distribution which is updated in every iteration according to the estimated cumulative loss for the selected arm and all the arms.
- The algorithm also maintains an cumulative estimated loss for each arm which is computed and updated by taking the ratio of actual loss for the selected arm and the value from the probability distribution for that arm.

- In each iteration an arm is selected from the probability distribution randomly, the estimated loss for the arm is found using the loss value, the probability distribution and the cumulative estimated loss for that arm is updated. Further, the probability distribution over all the arms is calculated for the next iteration by using the cumulative estimated loss for each of the arms and a hyperparameter η. In our implementation we took $\eta = \sqrt{\frac{\ln k}{tk}}$.

- The instantaneous regret of EXP 3 algorithm is as follows:

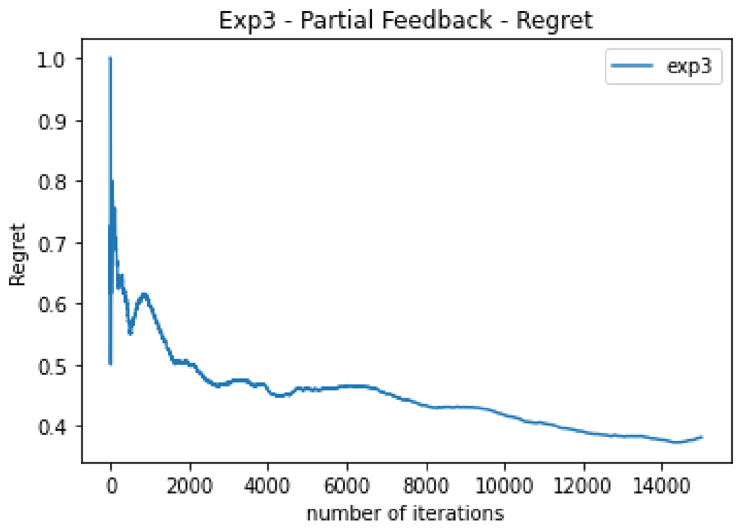

Figure 22: Instantaneous regret EXP3

- Comparison of cumulative regrets of EXP3 with all other stochastic algorithms is as follows::

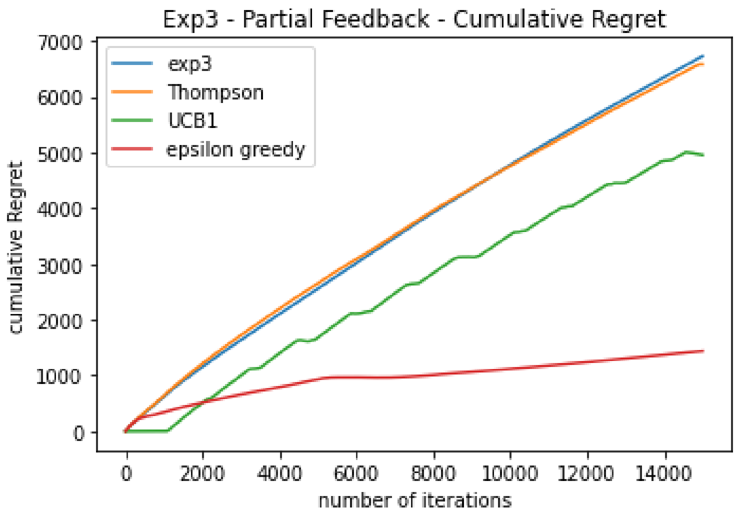

Figure 23: Cumulative regret

- Comparison of cumulative losses of EXP3 with other stochastic algorithms is as follows::

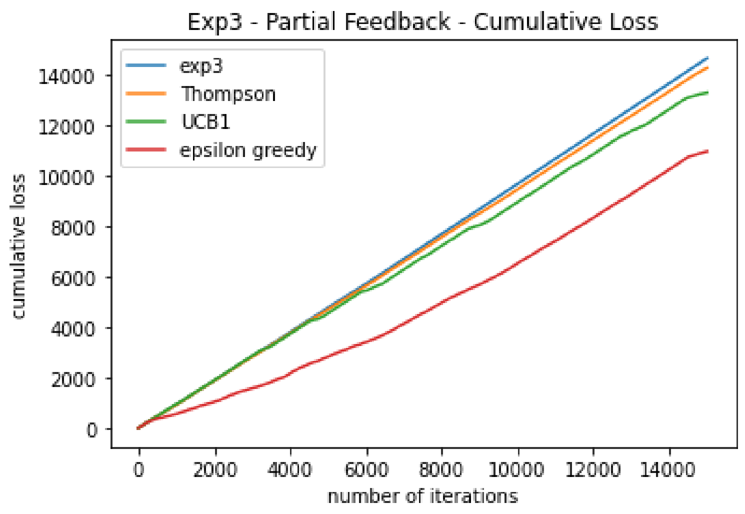

Figure 24: Cumulative loss for all the Partial Feedback Algorithms

- Total rewards over 14999 iterations: 457.0
- Total regrets over 14999 iterations: 6725.107853605043

Summary of Partial Feedback Algorithms

- Thus, to summarize the behaviour of partial feedback algorithms, both stochastic and non stochastic, the above two plots(Fig 23 and 24) can be used to compare the performance in terms of the cumulative regrets and cumulative losses of all four algorithms.
- Thus, from the plots it is evident that our tuned epsilon greedy with the value ofϵ as $\frac{100}{t}$ gives the least cumulative regret and least cumulative loss of all the partial feedback algorithms. This might be because of the nature of the given dataset where the same movies are recommended a lot of times, thus the strategy of epsilon greedy algorithm to exploit largely is beneficial in this case, which is very different from a realistic scenario, where non stochastic algorithms might be more useful.
- To even further summarize the performance of the studied partial feedback algorithms, the following table would be useful.

Algorithm	Type	Total Regret	Total Reward(actual)
Epsilon Greedy	Stochastic	580.15	5895.0
UCB1	Stochastic	4954.52	1419.0
Thompson Sampling	Stochastic	6580.03	714.0
EXP3	Non Stochastic	6725.10	457.0

1.3 Problem 3

Algorithms based on Full Feedback
In this section, we try the same algorithms but their full feedback counterparts, thus using the rewards of all the arms (irrespective of which arm is selected) to select the arm in the next iteration.

Note on Regret Calculation and Difference between Partial and Full Feedback Algorithms

Here we use the Expected Full Rewards to calculate the best arm so far till the current day, similar to how we did in Partial Feedback algorithms, but here we also use this Expected Full Rewards to choose the arm in the current iteration for all algorithms. Thus in each iteration of these algorithms, after choosing/ selecting an arm, the rewards of all the arms are observed and the expected (Full) rewards are updated according to the rewards of all the arms, which are further used to select the arm in the next iteration of the algorithm.

1.3.1 Epsilon Greedy (Stochastic)

- As discussed already in the previous sections about Epsilon greedy algorithm, the decision of choosing ϵ is a crucial one. Thus, here in full feedback implementation also, we experiment with some different values of ϵ and then run our algorithm with the best chosen epsilon.
- Also note that the regret for full feedback implementation of epsilon greedy algorithm is zero, in case when exploitation happens, because the best arm and the current arm is selected in the same way by finding the arm with the maximum expected rewards.
- The instantaneous regrets for different values of epsilon:

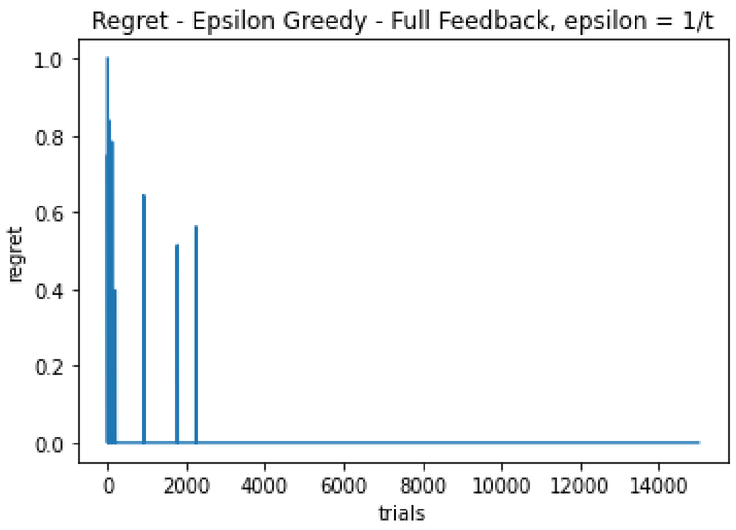

Figure 25: regret, epsilon = 1/t

–

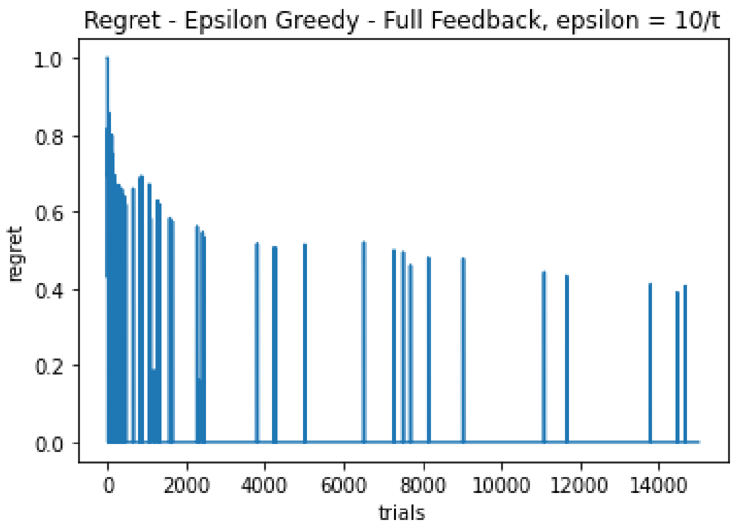

Figure 26: regret, epsilon = 10/t

–

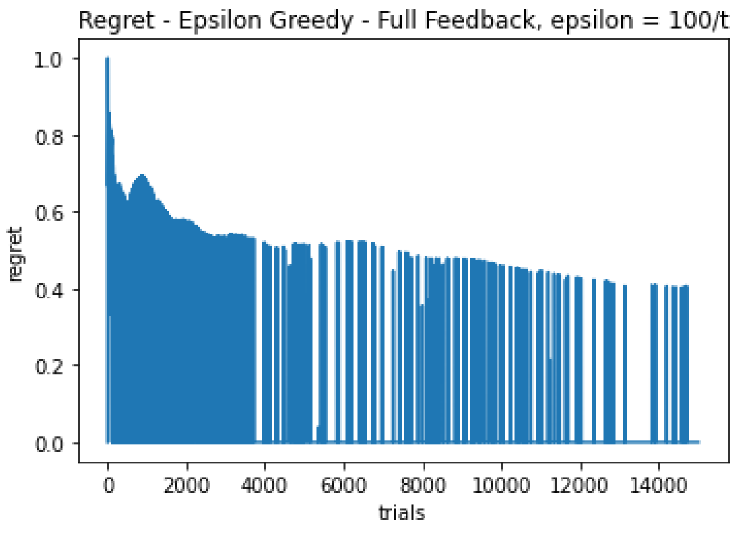

Figure 27: regret, epsilon = 100/t

–

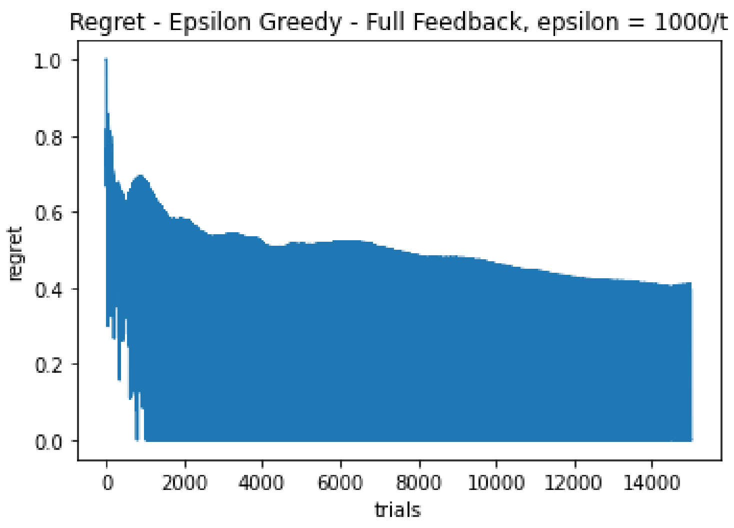

Figure 28: regret, epsilon = 1000/t

–

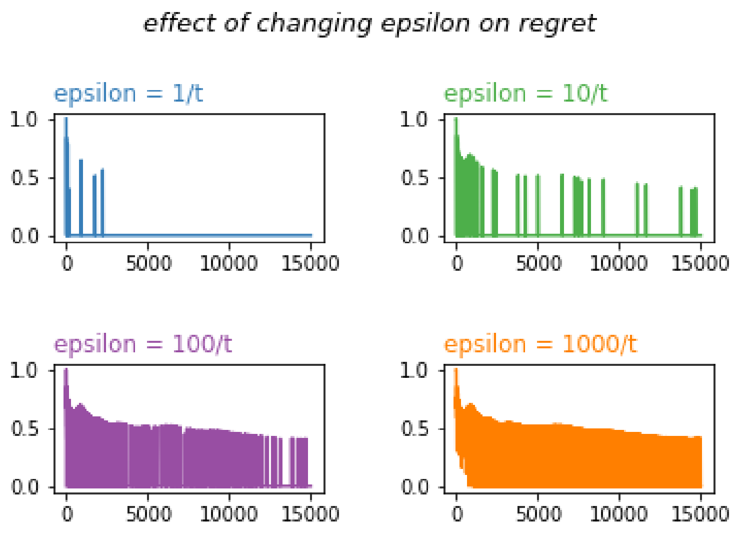

Figure 29: Comparison of all values

–

- Thus, as expected, we observe as we increase the value of epsilon the algorithm tries to explore more, and thus we can see a lot of exploration (resulting in noisy and oscillating plots) for high values of ϵ such as 100/t and 1000/t. But this does not give us a measure of best epsilon, thus we plot the cumulative regret to make a comparison.

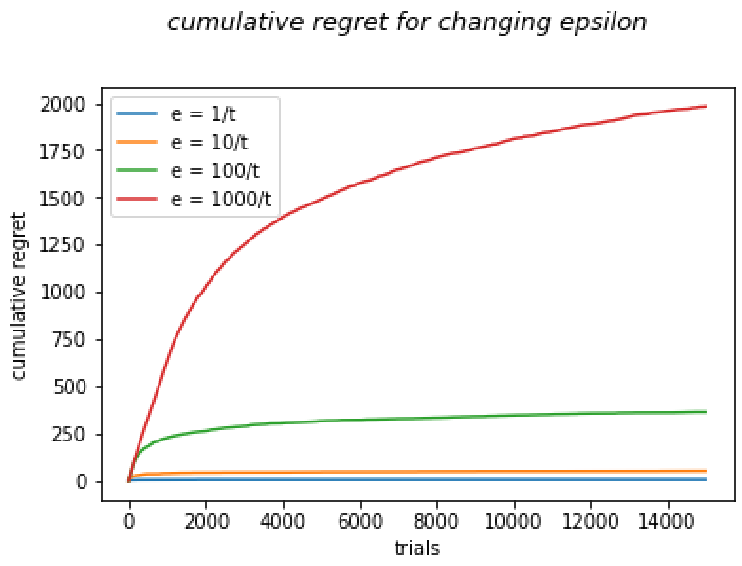

Figure 30: Comparison of cumulative regrets for different ϵ

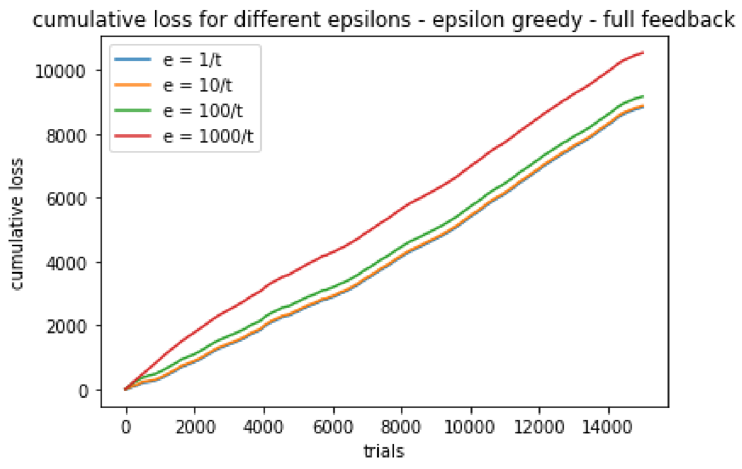

Figure 31: Comparison of cumulative losses for different ϵ

- Thus, in full feedback there is not much difference in the cumulative loss on changing the values of ϵ, but the values of cumulative regret keep increasing as we increase ϵ and as we explore more. Thus in this case we choose the best value of ϵ as $\frac{1}{t}$ as it gives the minimum cumulative regret and loss.
- The frequency of the actual reward being one for is as shown:

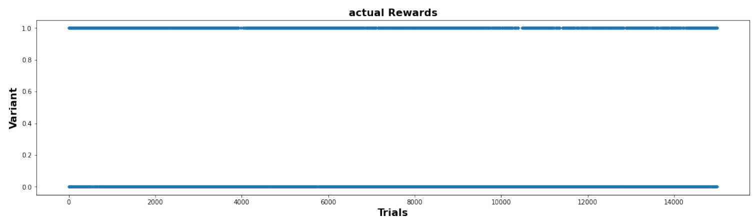

Figure 32: Frequency of reward being 1, Epsilon greedy full feedback, epsilon =1/t

- Total rewards: 6166.0
- Total regret: 7.880375717432527

1.3.2 UCB1 Algorithm (Stochastic)

- As discussed in previous sections, UCB1 algorithm for full feedback is the same except observing the rewards of all the arms, and updating the expected rewards for all the arms.
- The instantaneous regrets for Full Feedback UCB1 with $\delta = 1/t^4$ is as follows:

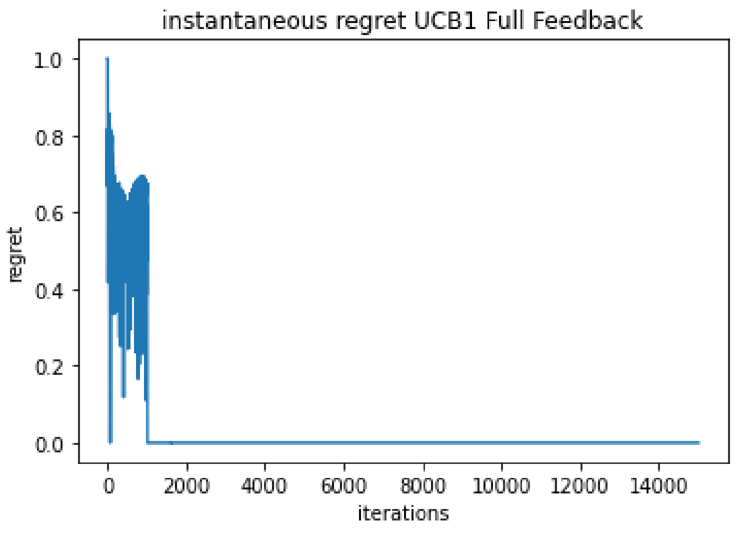

Figure 33: regret Full Feedback UCB1 with $\delta = 1/t^4$

- We observe that it oscillates only for the first 'k' iterations and then drops down to 0 and remains zero throughout since the best arm and the selected arm (with the maximum upper confidence bound) is the same.
- The cumulative regret of UCB1 with full feedback (alongwith Epsilon Greedy Full Feedback) is as shown:

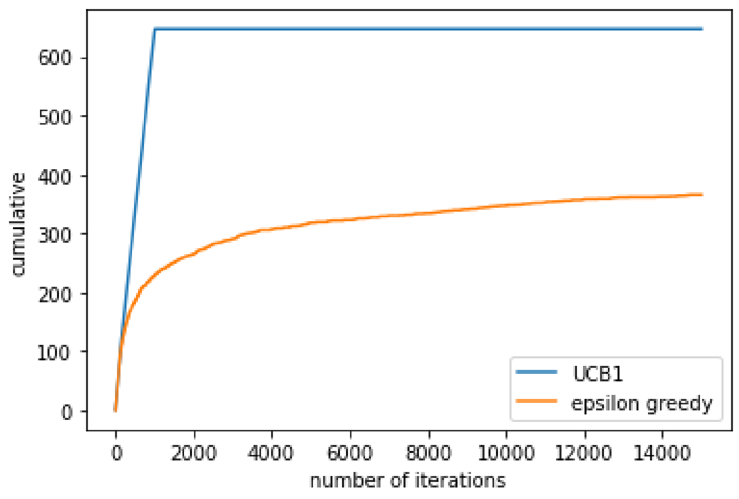

Figure 34: Cumulative regret Full Feedback UCB1 with $\delta = 1/t^4$

- Thus, the same pattern which was seen in intantaneous regret is also seen in cumulative regret, where regret oscillates a bit for the first k iterations thus increasing the cumulative regret, and then becomes zero, thus making the cumulative regret a constant value.
- The cumulative loss of UCB1 with full feedback (alongwith Epsilon Greedy Full Feedback) is as shown:

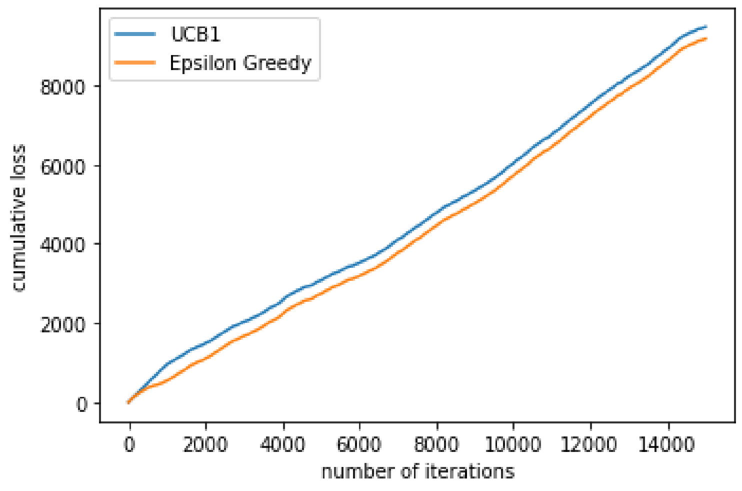

Figure 35: Cumulative loss Full Feedback UCB1 with $\delta = 1/t^4$

- Thus, we observe that there is not much difference in terms of performance of Full feedback UCB1 and Epsilon Greedy when measured using the cumulative loss.
- The frequency of the reward being one for the UCB1 full feedback algorithm is as follows:

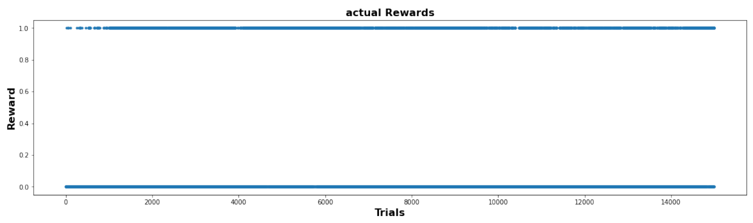

Figure 36: Frequency of reward being 1, UCB1 full feedback

- Total rewards: 5530.0
- Total regret: 646.9970918360648

1.3.3 Thompson Sampling (Stochastic)

- As discussed previously, Thompson Sampling with full feedback is the same except observing the rewards of all the arms, and updating the S and F values for all the arms, thus changing the beta distribution more effectively, for every arm.
- The instantaneous regrets for Full Feedback Thompson Sampling is as follows:

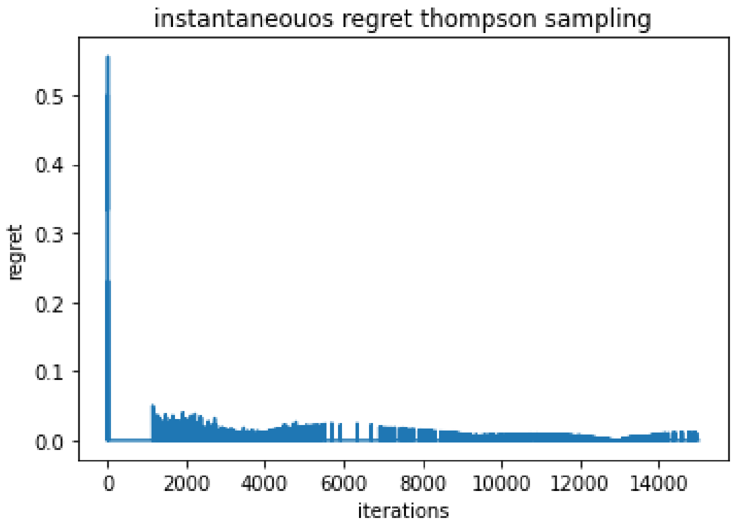

Figure 37: regret Full Feedback Thompson Sampling

- We observe that the regret remains really low throughout (well within 0.1) for all the iterations. Infact, this is the best performance we have observed till now for full feedback algorithms.
- The cumulative regret of UCB1 with full feedback (alongwith Epsilon Greedy and UCB1 Full Feedback) is as shown:

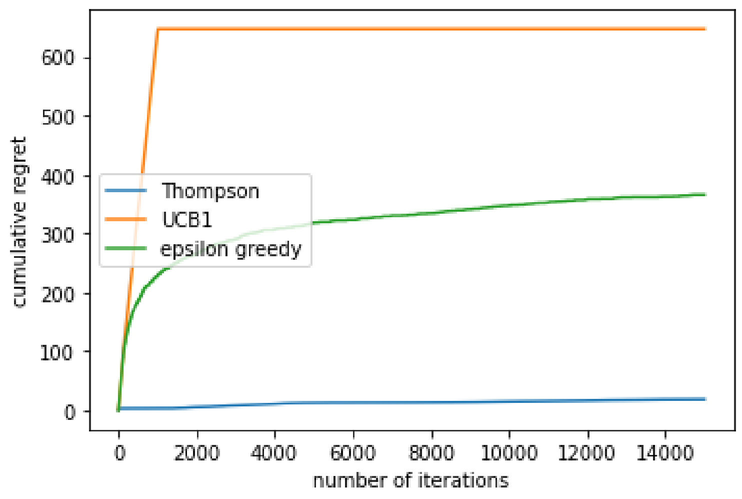

Figure 38: Cumulative regret Thompson Sampling Full Feedback

- Thus, Thompson Sampling with Full feedback gives the best results of all the Full Feedback Stochastic Algorithms, which is also evident from the cumulative regrets plot.
- The cumulative loss of Thompson Sampling with full feedback (alongwith Epsilon Greedy and UCB1 Full Feedback) is as shown:

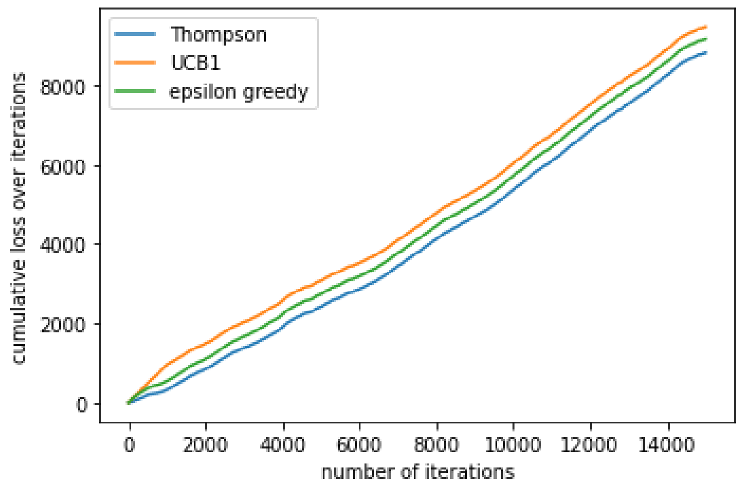

Figure 39: Cumulative loss Full Feedback Thompson Sampling

- Thus, we observe that the performance of Thompson Sampling is slightly better also in terms of the cumulative loss as compared to the other stochastic full feedback algorithms.
- The frequency of the reward being one for the Thompson Sampling with full feedback algorithm is as follows:

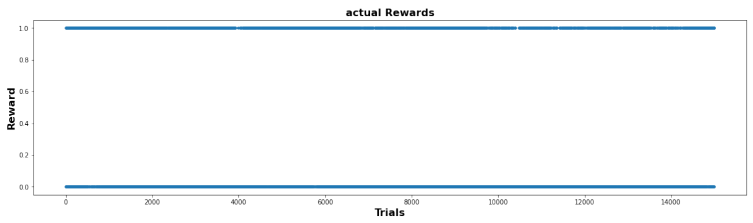

Figure 40: Frequency of reward being 1, Thompson Sampling full feedback

- Total rewards: 6133.0
- Total regret: 21.573091113672735

1.3.4 Multiplicative Weight Update Algorithm (Non Stochastic)

- As discussed previously, being a non stochastic algorithm, it even performs good when there is an adversary which can manipulate the losses, as the randomness helps in restricting the adversary's power in manipulating the losses.
- The algorithm assigns weights to each arm, and chooses an arm with a probability proportional to its weight. Next, the losses for all the arms are observed (Full Feedback), and the weight associated with each arm is updated multiplicatively by using a hyperparameter η and the loss associated with each arm in the current iteration.
- The instantaneous regrets for Multiplicative Weight Update Algorithms is as follows:

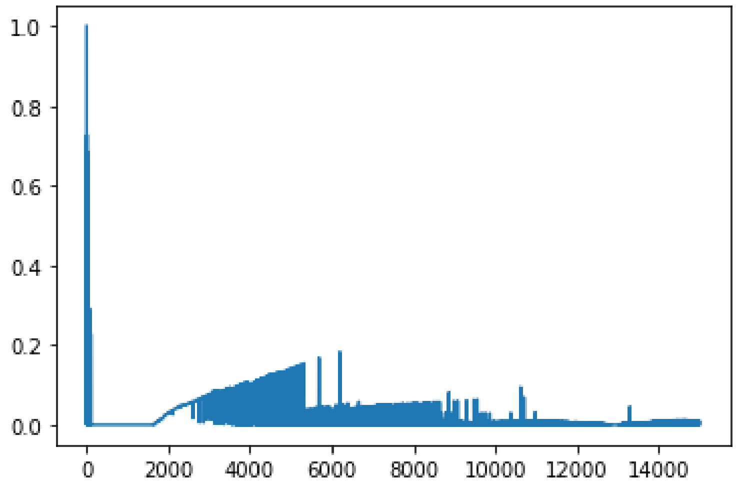

Figure 41: regret Multiplicative Weights Update

- We observe that the regret remains quite low for all the iterations, but still greater than the regrets for Thompson Sampling with Full Feedback.

- The cumulative regret of Multiplicative Weights Update Algorithm with full feedback (alongwith stochastic Full Feedback Algorithms) is as shown:

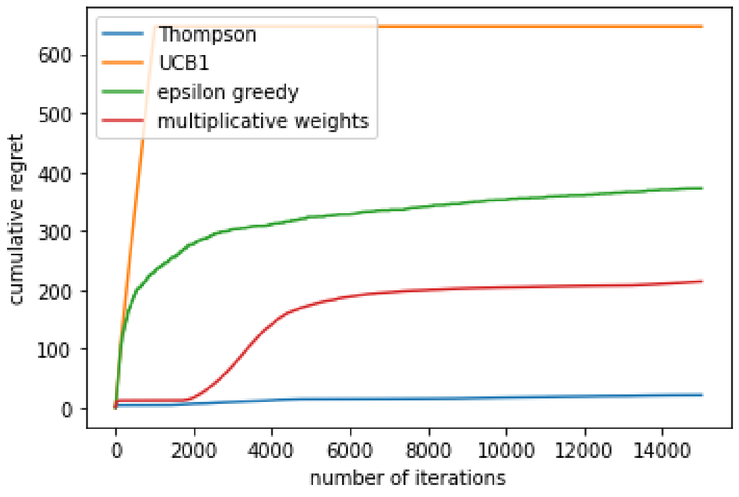

Figure 42: Cumulative regret all Full Feedback Algorithms

- Thus, Thompson Sampling with Full feedback gives the best results of all the Full Feedback Stochastic Algorithms, which is also evident from the cumulative regrets plot.
- The cumulative loss of all the implemented Full Feedback algorithms is as shown below:

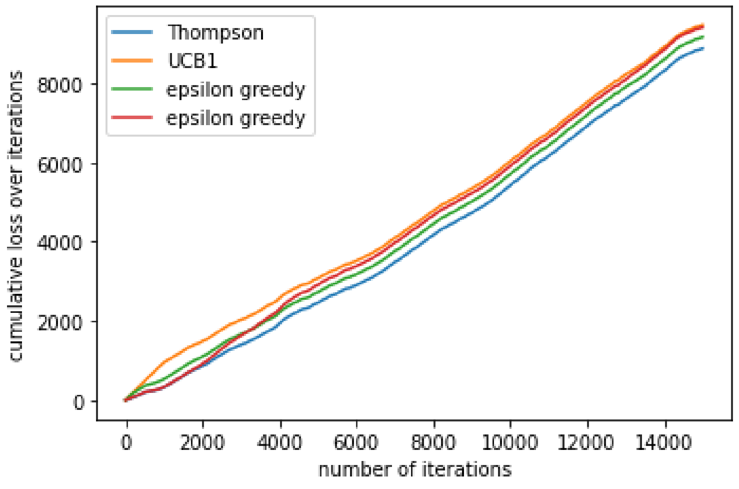

Figure 43: Cumulative loss Full Feedback All Algorithms

- Thus, we observe that the performance of Thompson Sampling is slightly better also in terms of the cumulative loss as compared to the other stochastic full feedback algorithms.

- Total rewards: 5549.0
- Total regret: 946.320895564955

1.3.5 Summary of Full Feedback Algorithms

- Thus, to summarize the performance of full feedback algorithms, both stochastic and non stochastic algorithms, the above two plots (Fig 42 43) can be use to compare their performances in terms of cumulative regrets and cumulative losses.
- Thus, it is evident that the Thompson Sampling performs the best in the case of Full feedback algorithms, there is very slight difference in the actual loss values of the four algorithms, however the cumulative regret value of Thompson Sampling is way lesser than the other algorithms, which might be due to the Bayesian properties meaning that we can fully inspect the uncertainty of expected rewards. With uncertainty information explicitly captured via a posterior distribution the algorithm can decide when to exploit and when to explore its environment.
- To further summarize the performance of these full feedback algorithms, we can use the following table:

Algorithm	Type	Total Regret	Total Reward(actual)
Epsilon Greedy	Stochastic	371.89	5840.0
UCB1	Stochastic	646.99	5530.0
Thompson Sampling	Stochastic	21.57	6133.0
Multiplicative Weights	Non Stochastic	946.10	5549.0

Comparison of Partial Feedback and Full Feedback Algorithms

1. Epsilon Greedy Algorithm
    - The following plots were made for same value of epsilon but with the partial feedback and full feedback variants of the algorithm.

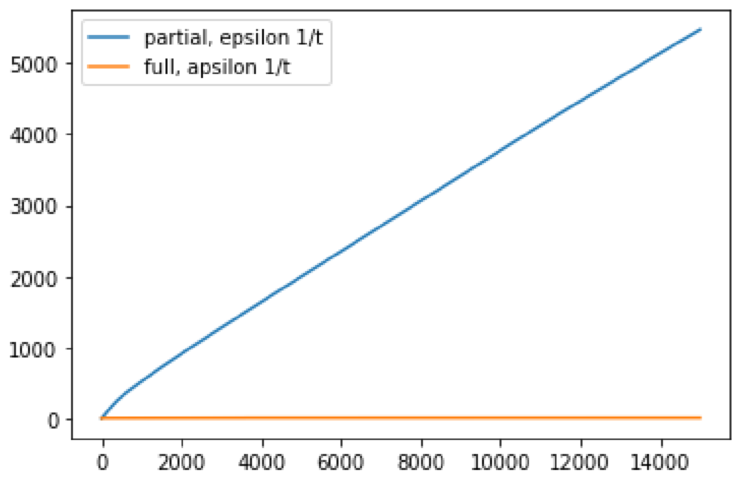

Figure 44: Cumulative Regret Full vs Partial, same epsilon

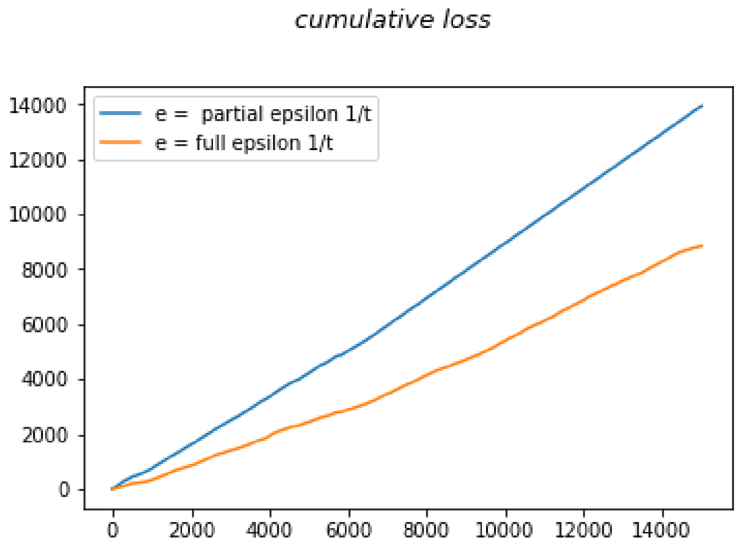

Figure 45: Cumulative loss Full vs Partial, same epsilon

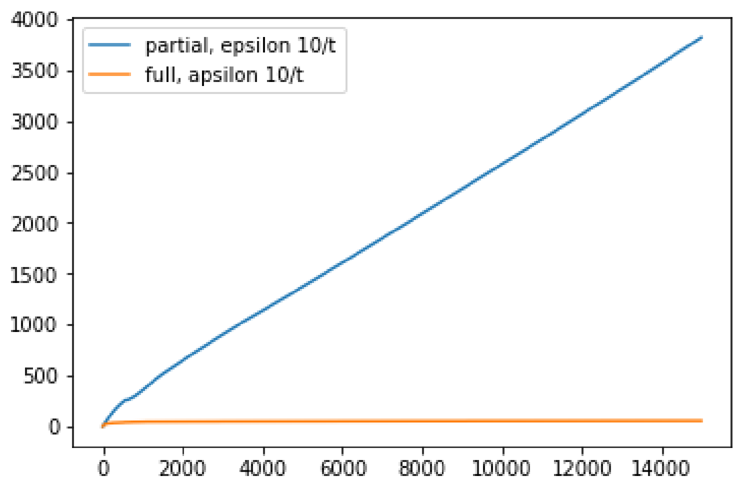

Figure 46: Cumulative regret Full vs Partial

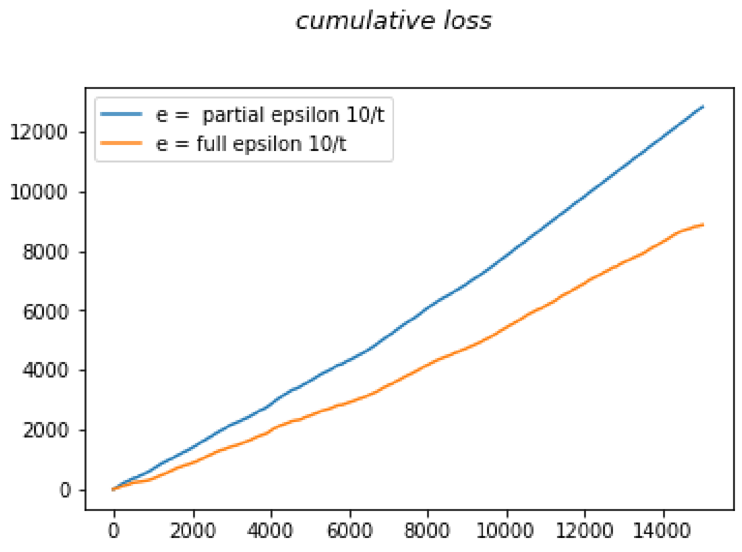

Figure 47: Cumulative loss Full vs Partial

- Thus, it is clear that for Epsilon Greedy algorithm the Full Feedback performs better than the partial feedback algorithms in terms of both, loss and regret.

2. UCB1

- The following plots were made for same value of delta but with the partial feedback and full feedback variants of the UCB1 algorithm.

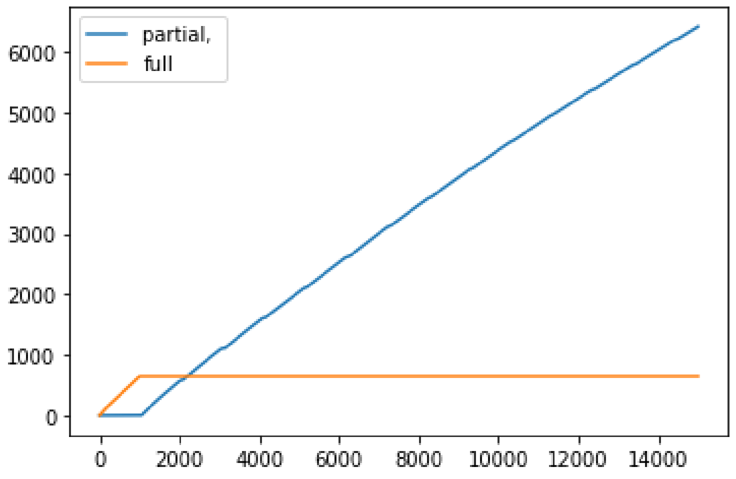

Figure 48: Cumulative regret Full vs Partial

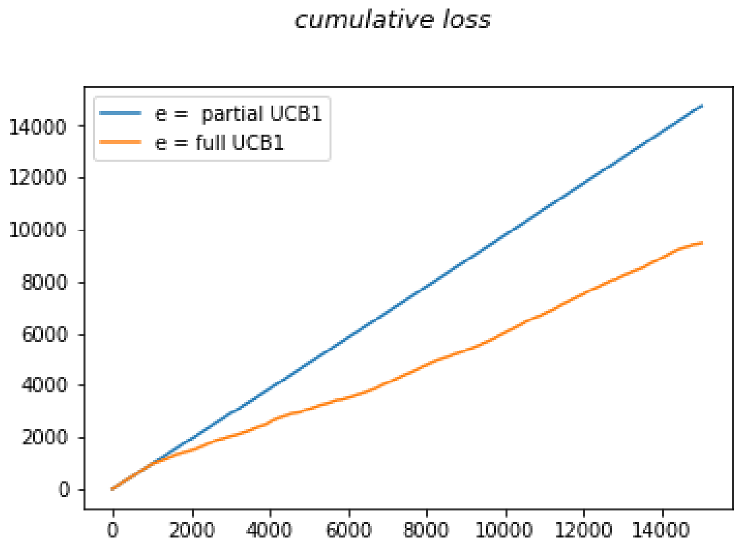

Figure 49: Cumulative loss Full vs Partial

- Thus, it is evident for UCB1 also, the full feedback version performs way better than the partial feedback version.

3. Thompson Sampling

   - The following plots were made with the partial feedback and full feedback variants of the Thompson Sampling algorithm.

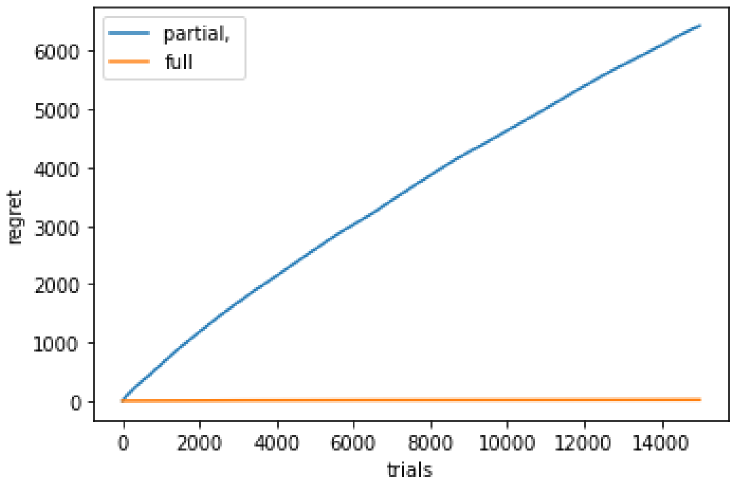

Figure 50: Cumulative regret Full vs Partial

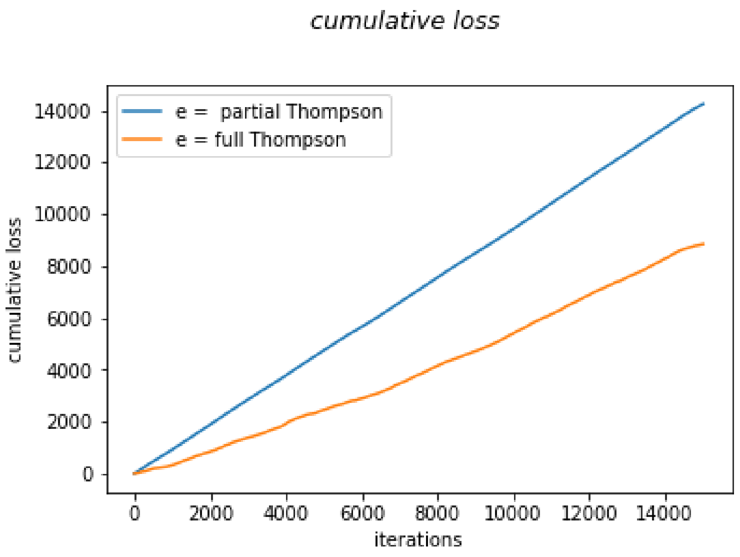

Figure 51: Cumulative loss Full vs Partial

- Thus, it is evident for Thompson Sampling also, the full feedback version performs way better than the partial feedback version.

Thus, it can be safely said that for all the algorithms the full feedback version of the algorithms performs better than the partial feedback version of the algorithm since it has a lesser cumulative regret in all the cases and also lesser loss values. Also, this is very intuitive, since the partial feedback algorithms use very limited information (that of the selected arm) to select the arm in a given iteration, whereas the full feedback algorithms, have observations of all the arms irrespective if which arm was selected, thus is certain to make better selections of arms.

1.4 Problem 4

Tracking Recommendation Probabilities of top ten movies

- The metric which we used to find the top ten movies was the obvious metric of highest actual reward values summation over all the trials.
  We found the top ten movies by using the information of all the arms(movies) at all time intervals, found out the actual reward for each movie at the end of 14999 days, and picked up the top ten movies with the highest values to find the best 10 movies.

- The top ten indices which we got from this method were:

$$[806, 778, 15, 211, 406, 160, 968, 887, 328, 107]$$

  with their corresponding summation of actual reward values over the time of 14999 days being

$$[3558., 3812., 4172., 4369., 4684., 4730., 5243., 5510., 6016., 6174.]$$

- Further, we used the non stochastic full feedback algorithm (Multiplicative Weights Algorithm) to keep a track of the recommendation probabilities of the top ten movies. The plot obtained was as shown below:

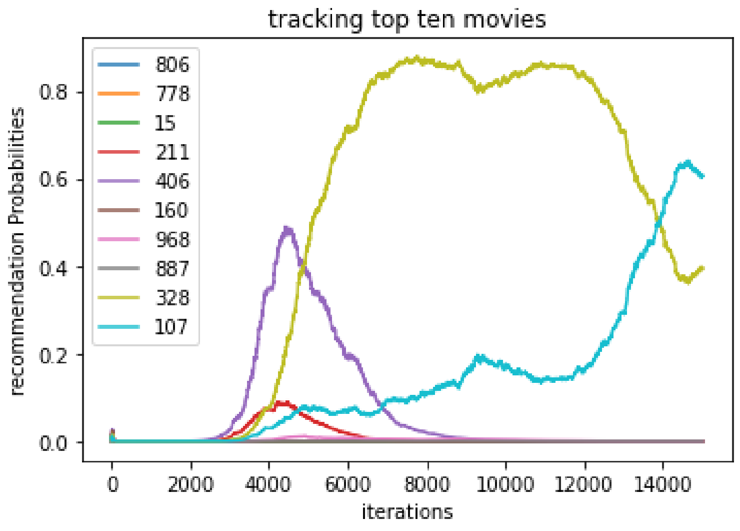

Figure 52: Recommendation Probabilities of Top Ten movies

- ANALYSIS: From the plot above, it can be said that although these were the top ten movies in terms of total actual rewards, but their individual behaviour in terms of being suggested by the algorithm varied a lot with time. This might be due to the fact that the algorithm which we are considering here is a non stochastic algorithm.

  However various trends about these different movies can be observed. The Movie with index 806 started very gradually and rose over the days to achieve the highest recommendation probability.

The movie numbered 328 had a very good start and was at the peak of being suggested for a majority of time, but later saw a downfall in the probability.

Other movies such as 406 and 211 were started gaining some traction but did not work very well and thus their recommendation probability dropped down to zero as days progressed. Whereas majority of the movies have almost flat curves but are spread over the whole time with very low values, which is the reason why they have high total reward values over the 14999 days period.

An important thing to be noted here is that all these behaviours are due to the nature if the provided dataset, and the algorithm used (multiplicative weights update) which penalizes the arms with large loss, thus decreasing their weights and probability of being picked.

All these behaviours and trends are often seen in real life examples, thus such algorithms can definitely be used to recommend products in real life scenarios as well.

1.5 Problem 5

We were getting the following regret values for the following algorithms. The analysis and resulting choice of hyperparameters has already been explained in Problem 2 and in much detail within each of the algorithm. Please look into the Partial Feedback Algorithm Section for details on the design decisions for the algorithms.

Algorithm	Type	Total Regret	Total Reward(actual)
Epsilon Greedy	Stochastic	580.15	5895.0
UCB1	Stochastic	4954.52	1419.0
Thompson Sampling	Stochastic	6580.03	714.0
EXP3	Non Stochastic	6725.10	457.0

2 Module 2 : Clustering

2.1 Problem 1

Preprocessing of Dataset

- The dataset was downloaded and read as a Pandas Dataframe. The dataframe had about 1865473 rows and 34 columns originally. Each row of the dataset corresponds to a movie and each column corresponds to a unique attribute of that movie. Some basic preprocessing steps were applied to the dataset in order to clean the data.

- We observed the mean values of the attributes startYear and runtimeMinutes was 2001.8942000232648 and 44.27949962288385 which were comparatively large as compared to the mean values of the other columns which were all less than 1, thus decided to perform min max normalization to obtain values between 0 to 1, so as to avoid the bias of these values in distance calculation. Also, we dropped the string based columns such as originalTitle, tconst,titleType in order to have measurable distance metric for the clustering algorithm.

2.2 Problem 2

Online Version of K Means Clustering

- We try to formalize the K means clustering problem as an optimization problem and use minibatch gradient descent to update the centers thus learning them in an online manner.
- We randomly intialize our cluster centroids for this implementation and iteratively improve over the solution until a local minima is reached.
- This method is certainly better than LLoyd's Algorithm since LLoyd's Algorithm requires k passes over the whole dataset with N points and has the complexity of the order O(knd), where k is the number of centroids, n is the number of datapoints in the dataset, and D is the dimension of each datapoint.
- The basic steps in implementing the k Means algorithm by using minin batch gradient descent were
  - Random selection of centroids
  - For each iteration, random selection of datapoints in the mini batch to be considered.
  - Further for each point in the mini batch, distance is calculated from all the centers.
  - Note on Learning Rate η: We also maintain an array v, of size equal to the number of centroids or 'k', which is basically incremented for a centroid, when the centroid is the nearest centroid for a given point in mini batch among all the centroids. And for each point in the minibatch, this v value of the nearest centroid is used to set up the learning rate ets, which is taken as $\frac{1}{v[nearestcentroid]}$, thus for a centroid which is the nearest centroid to a lot of points in the mini batches, the value of v is large and thus the learning rate is small, and the centroid does not change much; whereas for a centroid which is not the nearest centroid to a lot of points, the eta value is large and thus changes a lot in order to reach convergence faster.
  - Update the centroid with the minimum distance using the gradient descent update equation.
- Since the results of K means clustering are heavily influenced by the initialization of the centroids and the value of k chosen, we ran the algorithm for different values of k and tried to find the distance of each data point to the nearest centroid and averaged over all the points. Further, We have plotted the minimum, maximum and mean over all the clusters using a custom evaluate function we wrote.
- We used the following values of k to initialize the number of centroids [5, 10, 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500] and based on these values of k, we calculate the centroids and compute the mentioned distance metrics.

- The results obtained for this are as shown below:

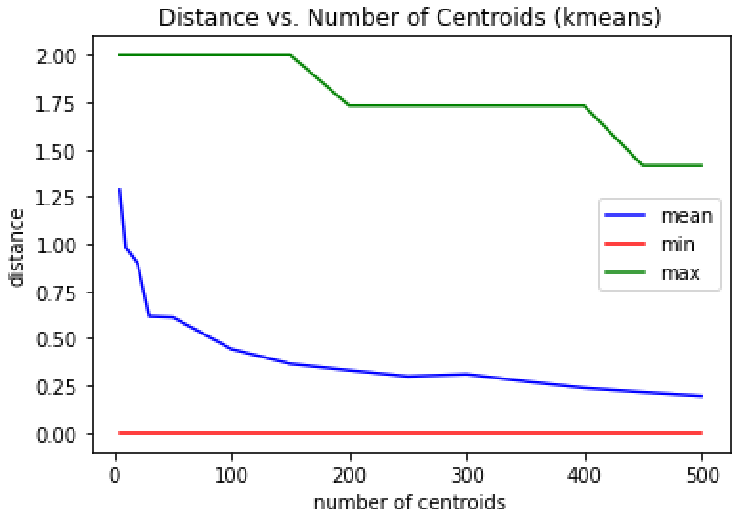

Figure 53: Min, Max and Average distances of datappoints from their nearest centroid for diff k

- We observe that as we increase the number of centroids for clustering i.e. k the calculated distances go down, and as the number of clusters increase, the sum of squared distances will tend to zero.

2.3 Problem 3

K Means++ initialization technique

- Since the optimization problem being solved here is highly convex, and many local minima exist, thus the initialization can dramatically influence the results. Thus, here we use a smart initialization technique i.e. K Means++, which is based on picking successive centroids from a frequency distribution which is proportional to the square of the distance between the datapoints and the nearest centroids. (D^2 sampling).

- The first centroid is initialized randomly by selecting one of the points from the existing datapoints.

- For the i^{th} centroid, the nearest already existing centroid is found out for every datapoint and this distance along with a normalization factor (which is the sum of this term for all the datapoints) is used to create the probability distribution, and the next centroid is randomly sampled from this probability distribution. Thus trying to pick points which are far away from the current points and having a better spread of centroids.

- Another advantage of stochastically choosing points from this probability distribution is that in case of an outlier which is very far from the actual clusters the outlier is not always selected as the centroid (which is actually a wrong choice of centroid).

- The rest of the algorithm is the same as the previously implemented online version of kmeans algorithm, it is just that the initialization is different in kmeans++, which can significantly effect the performance of clustering.
- To observe the change in the results of clustering by using this method of initialization over the previous method of random initialization, we find the distance of each data to the nearest centroid, and the minimum maximum and mean distance over the clusters. We used the same values of k i.e. [5, 10, 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500] for a good comparison of the two methods.
- The Results obtained were as shown below:

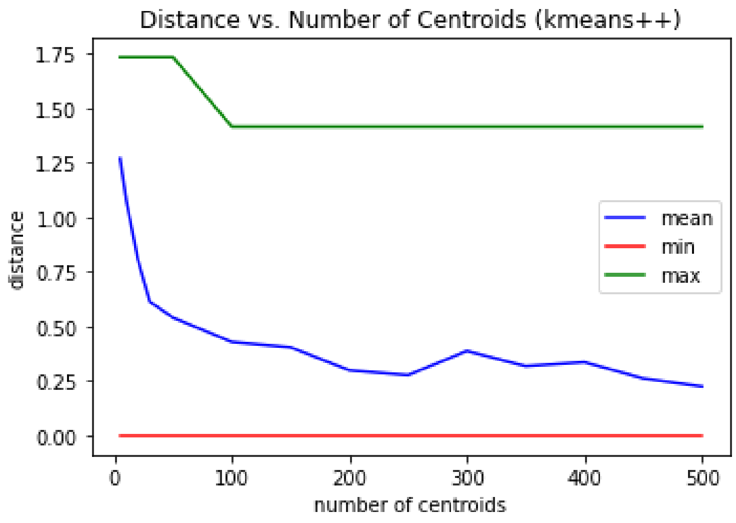

Figure 54: Min, Max and Average distances of datappoints from their nearest centroid for diff k

- When k is in the range [5, 250] as the number of centroids increases, the distance between the dataset and the corresponding closest centroids decreases. But when k crosses 250, the distance starts to increase slighlty but eventually decrease to a value around 0.25.

Comparison of K Means and K Means++

- As we have already seen from the plots in the respective sections, both k means and k means++ converge around a minimum value of around 0.25 for the mean distance. However, an observation to be made here is that the curve is steeper for k means++, thus we can say that convergence rate of k means++ is faster than that of k means.
- In both the cases, the values of go down as we increase the number of clusters and tend to almost zero as we keep increase the number of clusters or k.

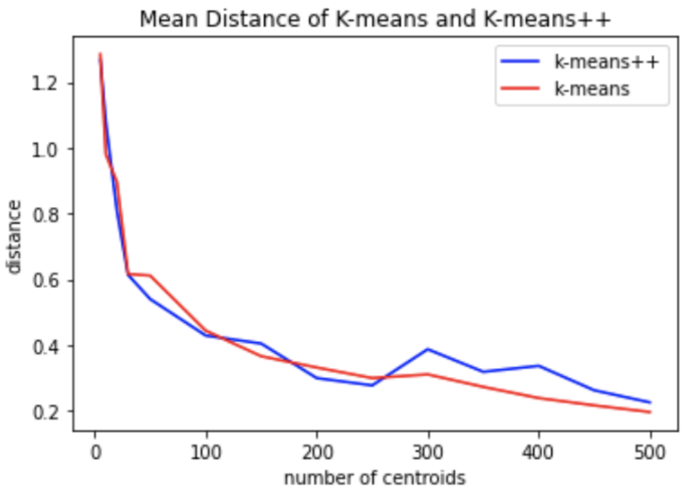

Figure 55: Average distances of datapoints from their nearest centroid for diff k

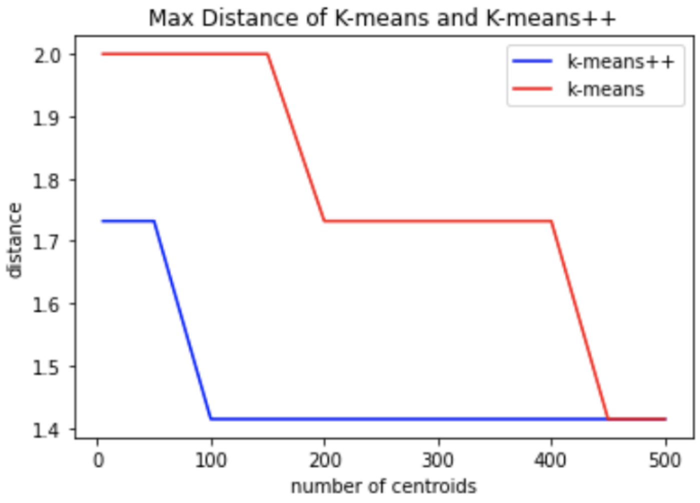

Figure 56: Max distances of datapoints from their nearest centroid for diff k

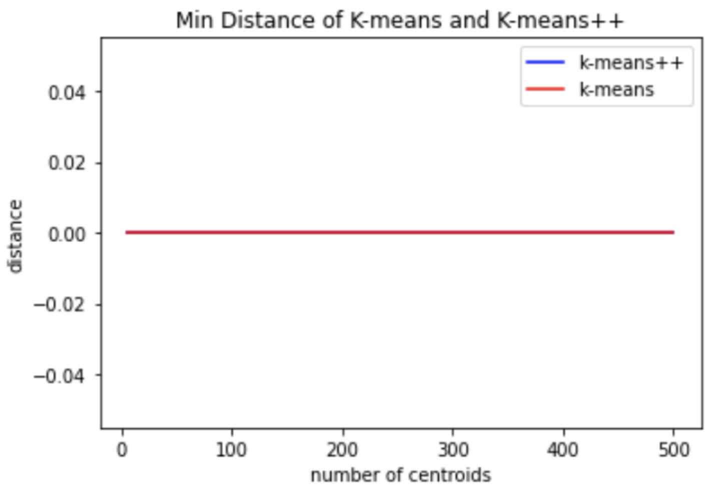

Figure 57: Min distances of datapoints from their nearest centroid for diff k

- We observe that the maximum distances for kmeans++ for a given value of k are significantly smaller than the distance values for kmeans, which signifies that the smart initialization was indeed useful, and the optimization algorithm reached a better minima value when initialization was done using kmeans++. The minimum distances for both the algorithms are zero, which might be the case that datapoints are same as the centroids.