ESE 545 Project 3

Yibo Yang[a,b]
[a]Department of Mechanical Engineering and Applied Mechanics,
[b]Penn Institute for Computational Science,
University of Pennsylvania, 3401 Walnut Wing A, 518,
ybyang@seas.upenn.edu

Abstract

In this report, we will discuss how we would be able to do complete the project 3 including module 1: recommendation systems with ϵ greedy [6], UCB [2], Thompson sampling [4] and EXP3 [3]. For module 2: clustering, we implement the K-means [5] and K-means++ [1] to see and compare their performance.

Keywords: Recommendation systems, Clustering.

1 Clarification

- The submission consists of two functions: main1.py and main2.py. Together with these two functions, we also include the models functions: models1.py and models2.py.
- All the results could be seen by simply running the above functions main1.py and main2.py.
- The whole project including the codes and report is completely finished by the author himself (thanks for the stimulating discussions with all the TAs).

2 Module 1

2.1 Problem 1

We download the data sets from the website and find there are totally 1,005 movies and 14,999 days data. In this case, we can formulate a k-arm bandits problem with 1,005 arms and total time of test as $T = 14,999$. We also observe that the whole data set contains $191,404$ positive feedback in the whole data set. In this case, it would be sufficient to develop an online algorithm to solve this problem. We should note that because we need to compute the loss and regret every iterations, the code might be a bit slow as we would like to take summation of all the movies each iterations and make comparison. If we do not need to compute these quantities, it will be extremely fast and will finish everything in minutes.

2.2 Problem 2

We implement both stochastic and non-stochastic methods on the partial feed-back problem. For stochastic methods, we use: ϵ greedy, UCB, Thompson sampling. For non-stochastic method, we use multiplicative weight updates. For the ϵ greedy, we set the first αN (where α is chosen as 1 in our case) iterations as random choice as it is encouraging the exploration to collect more information as it is partial feed back. For the UCB, we can also tune the uncertainty bound to make sure it is reasonable as $\text{UCB}(i) = \hat{\mu}_i + \sqrt{\frac{\beta \log t}{n_i}}$ where β is some hyperparameter we can tune and make the performance better and we choose $\beta = \frac{1}{10}$. For the Thompson sampling method, we use the weights γ in the observations to emphasize the effect of observations. In other words, we want to model the Beta distribution with $\text{Beta}(\gamma * S_i + 1, \gamma * F_i + 1)$ where $\gamma = 10$.

We are reporting the regrets and losses for ϵ greedy in figure 1. We would be able to observe that the regret increase fast at the beginning and than become stable as it is able to find the good arm to pull. This is benefited from the initial choice of ϵ as big number so that the algorithm can fully explore all possible arms and collect information.

We are reporting the regrets and losses for UCB in figure 2. In this case, we can see the regret is os-

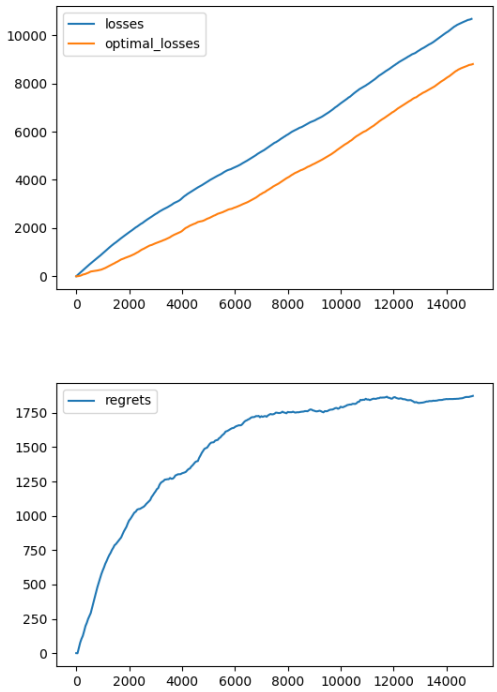


Figure 1: Regrets and losses for ϵ greedy with partial feedback.

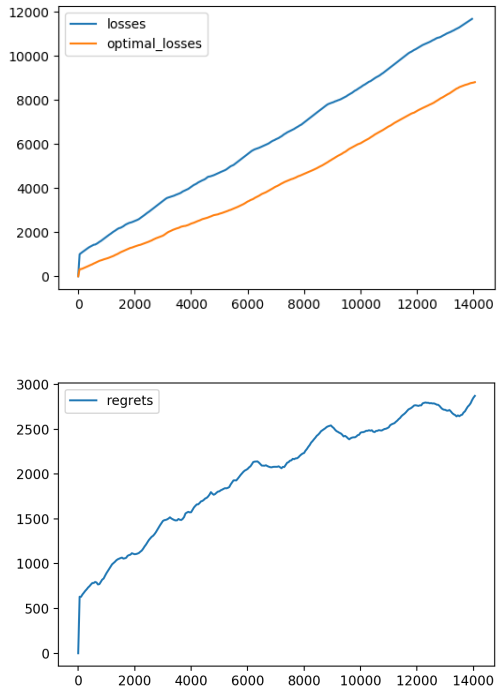


Figure 2: Regrets and losses for UCB with partial feedback.

cillating a lot as the number of classes is large and if the number of observations for a specific movie is small, the uncertainty related with that will be relatively large. This kind of explain why the oscillation happens sometimes. This is because we are accumulating data.

We are reporting the regrets and losses for Thompson sampling in figure 3. We can see Thompson sampling is providing a very good performance by having the regret increase at the beginning and quickly become stable. This is a probabilistic method, so all arms are possible to be pulled. The trick we used by increasing the emphasize of the arms help a lot the performance. So, we can see the final regret we obtained from this algorithm is not only stable but also having a relatively small value.

We are reporting the regrets and losses for multiplicative weight updates in figure 4. We observe that the regret is showing similar stable behavior in this case, and the final regret value is comparable with ϵ greedy, UCB and Thompson sampling.

We would observe very good performance of all the methods have very good convergence on the regret. The best algorithm from the stochastic method is Thompson sampling as it is providing very robust and consistent performance together with low regret value. For the non-stochastic method we are using, the performance is similar with what we have from the Thompson sampling.

2.3 Problem 3

We implement both stochastic and non-stochastic methods on the full feed-back problem. For stochastic methods, we use: ϵ greedy, UCB, Thompson sampling. For non-stochastic method, we use EPX3 and multiplicative weight updates.

We are reporting the regrets and losses for ϵ greedy in figure 5. In this case, we have full feedback, we pretty much are comparing the mean of the reward for each arm and pick up the best one. In this case, while time goes on, when the number of iteration becomes large, the ϵ becomes fairly small and lack exploration. That kind of well matching the data set we have and give very small regret.

We are reporting the regrets and losses for UCB in figure 6. In this case, we see very stable regret behavior because we have full information and is updating the mean and the uncertainty for each arm

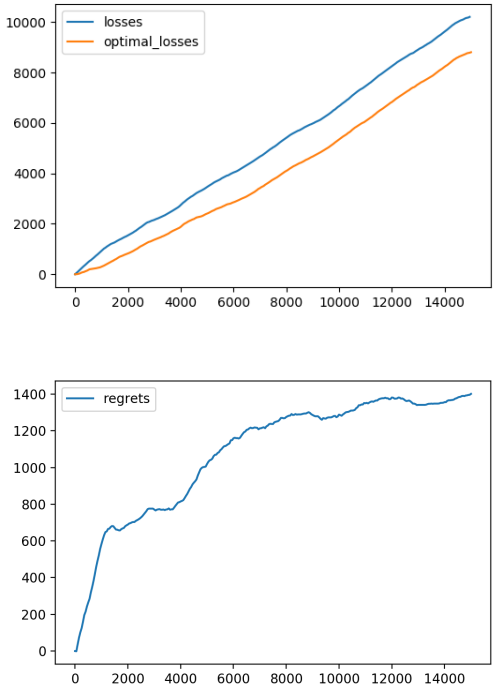


Figure 3: Regrets and losses for Thompson sampling with partial feedback.

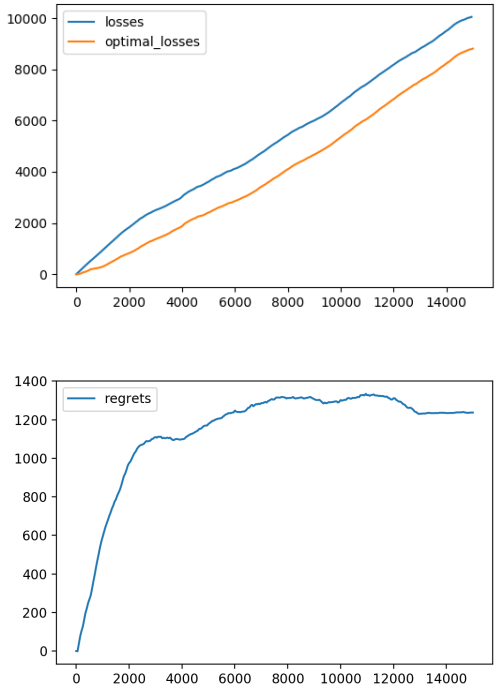


Figure 4: Regrets and losses for multiplicative weight updates with partial feedback.

every iterations. This significantly increase the stability of the algorithm and would help to mitigate the oscillating behavior in the regret.

We are reporting the regrets and losses for Thompson sampling in figure 7.

We are reporting the regrets and losses for EXP3 in figure 8. In this case, we use the $\eta_t = \sqrt{\frac{1}{t/10+1}}$ which is a very small modification but greatly improve the performance of the method by re-scaling the time. We observe that the regret is very small and not highly stable behavior. While the final regret value is small. More interesting results could be found in the probability figure in the next section.

We are reporting the regrets and losses for multiplicative weight updates in figure 9. We can see the regret converge very well as and to a reasonable value. This case, we do not have surrogate losses that would inevitably involve large number in the loss and cause some numerical unstable issue. As this method is a probabilistic method, we would not be able to expect the regret to be very small at the beginning. In this case, the regret we get makes good sense.

We would find that for the ϵ greedy and UCB algorithm, we would nearly find no regret from the implementation. I think this is because these algorithms are focusing on exploitation (with the parameters we use). The EXP3 algorithm with our modification on the η also shows very descent behavior and get the final regret around 50 which is relatively small. In the case of full feedback, we would definitely get very small regret. While Thompson sampling and multiplicative weight updates are somehow are encouraging the exploration, so their regret would be still descent and relatively small, but due to their probabilistic nature, we would find the regret would become stable once we see enough data.

2.4 Problem 4

We can easily find the top 10 movies among this data set by computing the mean of the rewards on each movie with respect to the number of day. This operation provides us the indices of these movies (index starts from 0) are $[107, 328, 887, 968, 160, 406, 211, 15, 778, 806]$. From the previous question, we have the full feedback and non-stochastic setting using the EXP3 algorithm and the multiplicative weight updates algo-

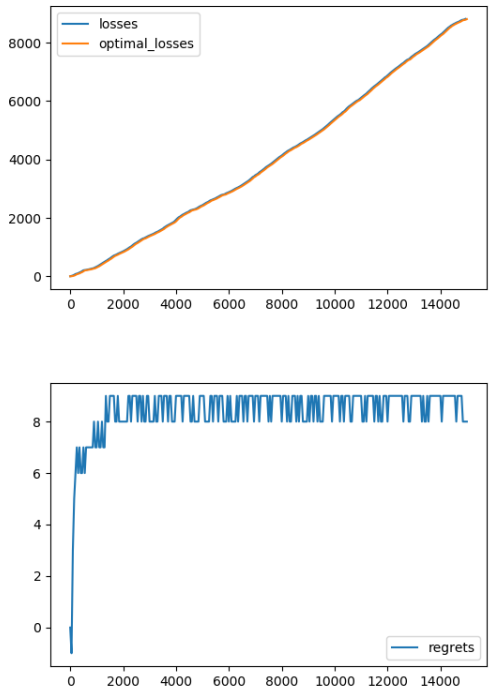


Figure 5: Regrets and losses for ϵ greedy with full feedback.

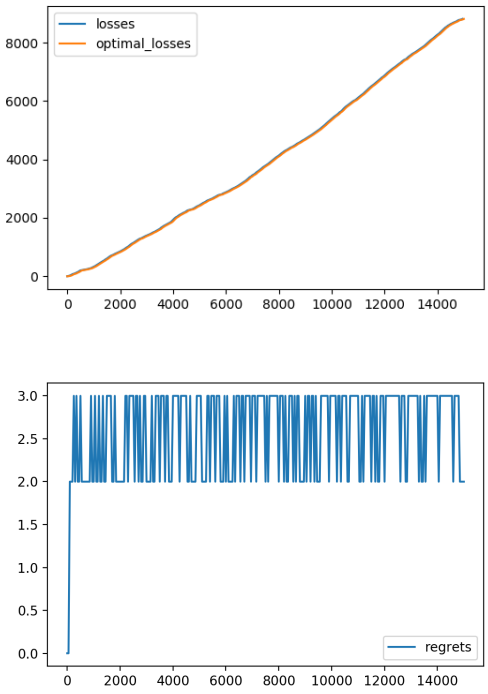


Figure 6: Regrets and losses for UCB with full feedback.

rithm. We are reporting the probabilities related with these movies from EXP3 in figure 10. We are reporting the probabilities related with these movies from multiplicative weight updates in figure 11.

This result from multiplicative weight updates is actually very interesting and can well justify our assumption. We can basically split the time period into 4 parts. At the very beginning ($t < 1000$), as the model at the very beginning has some probability for all movies, so the probabilities shown in this figure are all relatively small. At the second part ($1000 < t < 4000$), we would see these picked movies are having some high probabilities for each one and the summation of these movies are close to 1 means the other movies are neglected at this point. In the third part ($4000 < t < 12000$), all other movies are tending to have probability around 0 except the top 2 movies while the best is having slightly lower probability than the second one (this is due to the in-efficient number of observations). Then, at the final time period ($12000 < t$) we would see that the best one dominant nearly the whole probability measure to be recommended while the second one is still not 0. We would argues this behavior using the criteria that these two movies are essentially having similar probability (or both of them are liked by the audiences). While the order of the movies appears to have higher reward is also important. Although the second best movie has less total reward than the best movie bases on our selection rule, it still has the advantage that the rewards rise faster than the best movie at the very beginning. So, if we would like to reconsider the ranking rule we made (by just taking the mean and make the comparison) by using exponential or polynomial decay weights on these rewards, we may get some different rank. So, this is totally up to the selection rule we made to choose these top 10 movies. But in this figure, we already see the power of the proposed algorithm for picking up the best movies.

While we would like to see if we can further improve the performance of the multiplicative weight updates by using larger *eta* which we use $\eta_t = \frac{3}{\sqrt{T}}$ and actually observe significant improvement in the regret. This change make the model more sensitive with respect to each reward (or loss) it observe in each round. We are reporting the probabilities related with these movies from multiplicative weight updates with $\eta_t = \frac{3}{\sqrt{T}}$ in figure 12. In this case, we would see the probability is quite similar with the

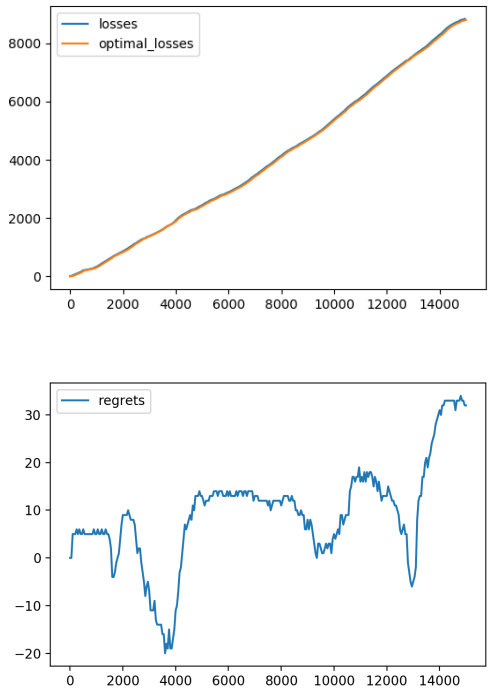


Figure 7: Regrets and losses for Thompson sampling with full feedback.

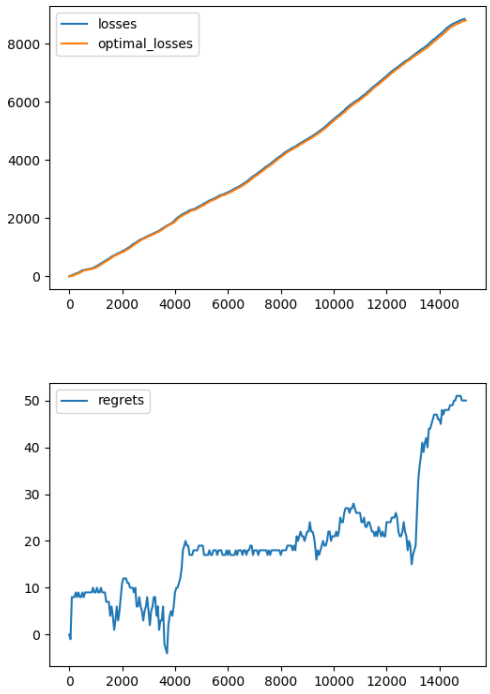


Figure 8: Regrets and losses for EXP3 with full feedback.

one we get from EXP3 (that kind of provide very small regret value). In this case, we see that the algorithm is recommending the best movie in each time period with very sensitivity to the reward of such movies. We would expect the sharp transition between the best movie telling us the performance of the method.

2.5 Discussion

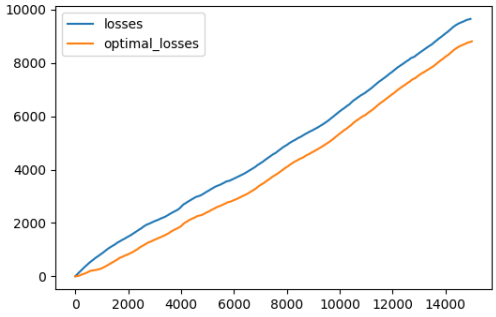
We would see that all methods get small regrets from the full feed back cases than the partial feed back cases. This is obvious and easy to understand because the algorithms see more information in the full feed back cases. So making decision with all the information known is definitely better than the partial information. Specially after fine tuning, we observe nearly 0 regret from full feed back cases.

3 Module 2

In this section, we would implement k-means and k-means++ for clustering the movies data set.

3.1 Problem 1

We load the data and find the number of movies contains in this data file is $1,865,473$ each with 34 features among whom we take the 29 binary features that we believe more characterize the main properties of the movies. We would like to comment on the reason why we throw away the year as it is not a good data preserving the order. For example, people maybe have different opinion on movies made at 2000 and 1950. But they may have similar opinion on movies made at 2000 and 1900. This relation is complicated and affected by a lot of properties. In order to keep the problem simple and do not introduce more continuous variables (as Euclidean distance taking into account of binary variable and continuous variable may mess things up). So, the final data input in our case is $X \in \mathbb{R}^{N\times D}$ where $N = 1,865,473$ is the number of movies and $D = 29$ is the number of features we will use to do the clustering task. We find that the data is nearly binary variable, so we choose to do not normalize the data as for each dimension they are already in a good range.



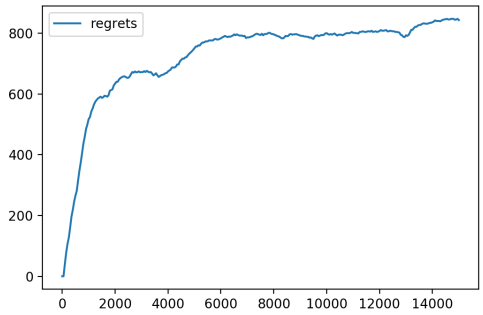


Figure 9: Regrets and losses for multiplicative weight updates with full feedback.

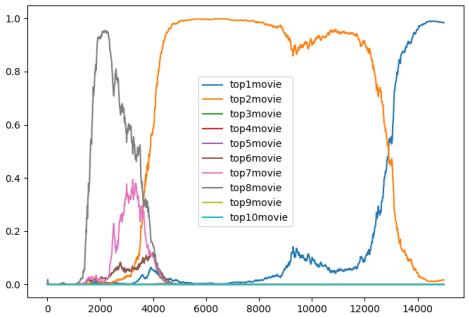


Figure 10: Probabilities related with the top 10 selected movies from EXP3.

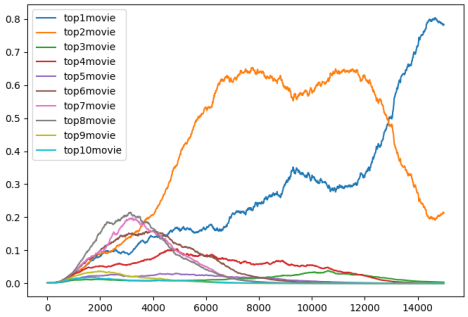


Figure 11: Probabilities related with the top 10 selected movies from multiplicative weight updates with $\eta_t = \frac{1}{\sqrt{T}}$.

3.2 Problem 2

For this question, we implement online version of the k-means clustering algorithm with random initialization of the K centers. Based on this large number of data set, we consider a mini-batch size as $10,000$ which we believe would be sufficient enough to capture the population properties of the whole data set. We also choose the learning rate $\eta_t = \frac{1}{t}$ to obtain the reasonable convergence rate. As this is a SGD approach, we would not expect the loss can decrease always without jumping back. In this case, we train the model for 500 iterations that we believe is good enough to converge. We test our model on different choice of K as $K = [5, 10, 20, 50, 100, 200, 400, 500]$ to see the performance. The figure summarizing the final error of the algorithm is given in figure figure 13. From the figure we would claim that starting from $K = 50$ the performance is not changing a lot. This tells us the K-means algorithm is poor due to the initialization. Specially in our case, the training data we have is binary variables with 29 dimensions. Bad initialization will always let the algorithm converge to a bad local minimum.

3.3 Problem 3

For this question, we implement online version of the k-means++ clustering algorithm which initialize the K centers iterative probabilistic according to the distance between the data and existing centers. Similar as the previous question, we consider a mini-batch size as $10,000$ and the learning rate $\eta_t = \frac{1}{t}$ to obtain the reasonable convergence rate. As this is a SGD approach, we would not expect the loss can decrease always without jumping back. In this case, we train the model for $1,000$ iterations that we believe is good enough to converge. We test our model on different choice of K as $K = [5, 10, 20, 50, 100, 200, 400, 500]$ to see the performance. The figure summarizing the final error of the algorithm is given in figure 14. From the figure we would also claim that $K = 200$ seems to be a good choice as the error is not changing dramatically when the cluster number is greater than 200. Also,

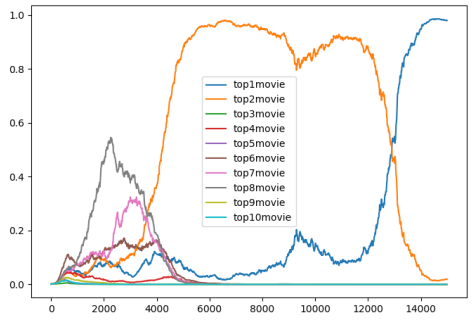


Figure 12: Probabilities related with the top 10 selected movies from multiplicative weight updates with $\eta_t = \frac{3}{\sqrt{T}}$.

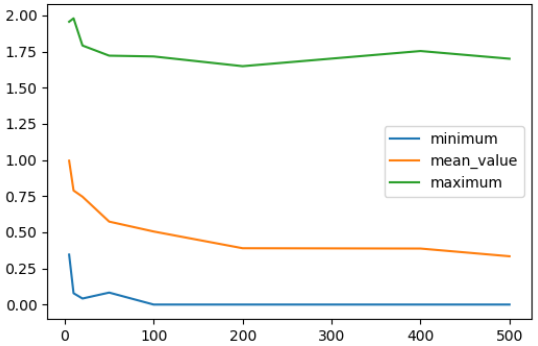


Figure 13: Maximum, minimum and mean distance as a function of number of classes in K-means algorithm.

noticing that the error decay from a relative smaller value than the k-means which means this initialization truly help a lot for initializing the model. In this case, k-means++ would not only give a better error at the end, it also provides a faster convergence rate due to the benefit of the initialization.

3.4 Discussion

Comparing the figure of error from k-means and k-means++, we would be able to make a conclusion that the probabilistic initialization related with distance significantly help a lot the performance of the algorithm to make sure the convergence more robust and would get to a good local minimum instead of some bad points. Final observation is that due to the sensitivity of the results on the initialization, we can see the k-means algorithm is oscillating a lot

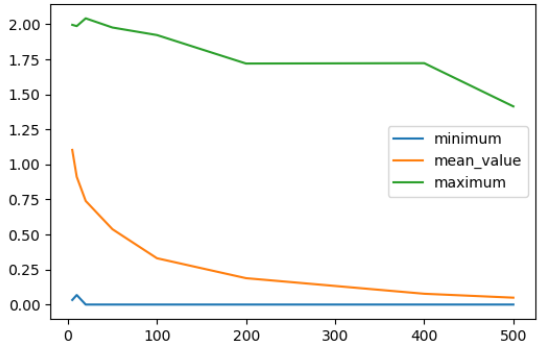


Figure 14: Maximum, minimum and mean distance as a function of number of classes in K-means++ algorithm.

with the change of K, while the k-means++ algorithm gives very stable decay trend as the number of cluster K goes up. So, we would definitely prefer to use k-means++ instead of k-means. However, the initialization of k-means++ is involving heavy computational cost, specially when the number of classes is large, e.g. $K = 500$. It takes long time to get the initialization done. But luckily there exist k-means MCMC algorithm to accelerate this process. We are not using this trick for this report, but one would expect to see that with the MCMC initialization, k-means++ would absolutely out perform than k-means. However, we can still consider a way to accelerate the initialization using a mini-batch. Say, we in order to get a new initialization center, we randomly sample M samples from the training data and compute the probability on these data to pick up the next center. This will definitely accelerate the speed.

Acknowledgement

We would like to thank professor Hassani and the TAs of this class for stimulating discussions. The codes and report is completely finished by the author himself.

References

[1] D. Arthur, S. Vassilvitskii, k-means++: The advantages of careful seeding, Tech. rep., Stanford (2006).

[2] P. Auer, Using confidence bounds for exploitation-exploration trade-offs, Journal of Machine Learning Research 3 (Nov) (2002) 397–422.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, R. E. Schapire, The nonstochastic multiarmed bandit problem, SIAM journal on computing 32 (1) (2002) 48–77.

[4] O. Chapelle, L. Li, An empirical evaluation of thompson sampling, in: Advances in neural information processing systems, 2011, pp. 2249–2257.

[5] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer Science & Business Media, 2009.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.