1 Experimental Setup and Results

In this section, we first present the experimental setup: the datasets we used, the implementation details, and the parameter space for evaluation. We then present the results obtained for the different datasets and algorithms tested.

1.1 Datasets and Settings

Dataset description and processing To conduct the experiments we used five real-world datasets along with two synthetic ones (Synthetic 1 and Synthetic 2) .

MNIST[1]**.** describes images of handwritten digits that are represented as vectors of dimension $d = 784$ (i.e grayscale images of size 28x28 pixels). For this dataset we follow the experimental setup described in Kuzborskij et al. (2019).

Steam[2]**.** The dataset originally contains reviews of games from multiple users. Games are characterized by different features such as the release date, the OS support, the tags of each game, the price and discounts available, etc.

MovieLens 25M[3]**,** The following dataset is composed of movie ratings from hundreds of thousands of users. Among the available features in this dataset are the year of release, the genres, the title.

Amazon books[4]**.** This dataset consists of reviews of books from two million users. Some of the available book features are the categories of a book, its price, and the number of reviews.

Yahoo[5]**.** This dataset contains users click logs for news articles recommendation. The proposed vectors of articles and users already available in the dataset are 6-dimensional vectors. Details on the construction of those vectors can be found on the dataset website.

Random Synthetic, Synthetic datasets are created such that each value of the action set of size $n \times d$ and the model parameter $\theta_* \in \mathbb{R}^d$ is sampled from a Gaussian distribution, $\mathcal{N}(0, 1)$. We select $n = 200$ actions of dimension $d = 100$ for Synthetic 1 and $n = 2000$, $d = 1000$ for Synthetic 2.

The full dataset details are available in 2.

We apply our bandit algorithms to the various datasets mentioned previously within a recommendation context. Consequently, in the case of real-world datasets, the actions (arms) are the vectorial representation of the items we can recommend. These vectors are constructed by concatenating some of the available features with the multi-label encoding of users who interacted with corresponding items. The reward observed corresponds to the adoption of the recommended item by a given user.

Moreover, as real-world datasets are characterized by a large number of users and items, we focused on the most relevant users – those who have interacted with a sufficient number of items – allowing us to estimate a realistic parameter vector θ_* within a reasonable execution time. This strategy, common in machine learning, highlights the importance of selecting relevant users to enhance computational efficiency in large datasets.

The complete description of the pre-processing of the datasets is available online.[6]

Evaluation settings Along with regret, we also consider the *click-through rate* (CTR), a measure commonly used to evaluate recommendation systems. The CTR is defined as the ratio

[1] https://pjreddie.com/projects/mnist-in-csv/
[2] https://kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam
[3] https://grouplens.org/datasets/movielens/
[4] https://amazon-reviews-2023.github.io/data_processing/index.html
[5] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=49
[6] `https://github.com/LiliZr/Stochastic-linear-bandits`

between the number of user clicks and the number of items recommended. To evaluate the algorithms on the above-mentioned datasets, two different settings were employed:

1. **Contextual setting.** (Figure **??**) The contextual bandit algorithms observe at each iteration a randomly chosen user (context). Each user has its own *itemset*. Hence, the set of items to be recommended changes with each context. User vectors are the contexts. If these are not explicit in the dataset[7], they are constructed by one-hot encoding each user. In this case, the reward r_t is equal to 1 if the user at the given iteration has already interacted with the recommended item and 0 otherwise. Consequently, the cumulative reward averaged over time is the CTR.

2. **Non-contextual setting.** (Figures **??**—**??** and Tables **??**—**??**) For non-contextual bandits, the action set is selected and fixed at the beginning of a run. For this, we assume that we select one user per run. The reward r_t observed at time t is computed as $r_t = \langle \theta_*, a_t \rangle + \eta$, with a_t the recommended action (item) at that time and $\eta \sim \mathcal{N}(0, 0.1^2)$.

Implementation details Where possible, we used the same setting to compare the various algorithms. We set the time horizon $T = 3000$ in the contextual setting and $T = 2500$ in the non-contextual one. We report the average and standard deviation of the results obtained by running each experiment from 20 to 50 times using the same random seeds to eliminate bias due to randomness. The metric of performance of each algorithm will be the *Click-Through Rate (CTR)* in the contextual setting and the *expected cumulative regret* in the non-contextual one. To measure the efficiency of different algorithms in terms of time and memory, we also compute the *CPU time* and *peak memory allocation*.

We perform a parameter grid search for each algorithm that we evaluate. We search the parameter $\lambda \in \{2 \times 10^{-11}, \ldots, 2 \times 10^{2}\}$, the parameter *scale* (i.e., a factor used to rescale β_t in the algorithms to avoid suboptimal results due to β_t increasing too rapidly) is varied as $scale \in \{10^{-6}, \ldots, 10^{1}\}$. We also vary the size of the sketching matrix m for the CBSCFD and SOFUL algorithms, along with the size of the projection matrix k in CBRAP and ConfidenceBall1-FJLT. For each combination and for each algorithm, we choose the best value combination and use it in the experimental results.

We conduct all experiments on a Linux server bi-pro Intel Xeon E5-2609v2 2.5GHz which contains 8 processors and has total memory of 64GB. The open-source Python implementation is available online[8]. Raw experimental results are also available online[9].

2 Datasets Complete Description

2.1 Non-contextual

Random Synthetic. We create synthetic datasets to represent our set of actions $\mathcal{A}$. Each dataset will be composed of a set number (200 and 2,000) of vectors (actions) of $d \in \{100, 1000\}$ dimensions and where each value is sampled from a Gaussian distribution, $\mathcal{N}(0, \sigma^2)$. The resulting vectors are normalized, i.e., $\forall a \in \mathcal{A}, \|a\|_2 = 1$. Finally, we generate a vector of size d representing θ_*, and we ensure the linear form of the gain obtained at a specific time t: $r_t = \langle \theta_*, a_t \rangle + \eta$. We set $\eta = 0.1$, in line with experimental evaluation in the literature.

[7] Only the Yahoo dataset contains explicit user vectors.

[8] `https://github.com/LiliZr/Stochastic-linear-bandits`

[9] `https://www.lri.fr/~izri/results.zip`

MovieLens 25M.[10] The following dataset is composed of movie ratings over 162,541 users and 62,423 movies. We first filter the users such that we keep only the users that rated between 1,500 and 3,000 movies. To build the vector representation of a movie, we extract the following features: (1) year of release, (2) genres, (3) statistical measures (mean, median, std, min, max) of the movie's ratings, and (4) the users that rated the movie (one-hot encoding). For each user in the dataset, we construct an itemset of movies rated by them. The resulting movie vectors are of dimension $d = 813$. For each round of the bandit algorithms, we proceed as follows: we select a user randomly, recommend a movie from the itemset and observe the reward as the real rating the user assigned the movie, plus a small noise $\eta \sim \mathcal{N}(0, \sigma^2)$.

MNIST.[11] We follow the experiment setup described in Kuzborskij et al. (2019), by adapting the classification problem to the bandit case. Given a dataset of C classes, we first select a target class. Then, at each iteration, the environment randomly chooses one sample from each class ($C = 10$ in this case, corresponding to the digits in the dataset), and then creates a set of C samples. The learner then selects one sample from the set and observes the reward, which is 1 if the selected sample belongs to the target class and 0 otherwise. The digits are represented as vectors of dimension $d = 784$ and we have a total of 60,000 samples.

Steam.[12] The dataset originally contains reviews from 7,495,040 users of 37,032 games. As in the Movielens dataset, we first filter the users such that we keep only users that recommended (positive or negative recommendations) more than 200 games. This reduces the number of users to 1,347. Then, we construct the game vector features using: (1) the release date, (2) OS support (Windows, Mac, Linux), (3) general appreciation of the game (Very positive, ..., mixed, ..., very negative), (4) ratio of positive ratings, (5) number of reviews, (6) prices (original price, discount if any, final price), (7) one-hot encoding of tags, (8) one-hot encoding of the users that played the game (only out of the 1,347 users we kept). This results in game vectors of dimension $d = 1{,}807$. We estimate the parameter θ_* corresponding to a user using linear regression between the vectors of games played by this user and their corresponding recommendations (positive/negative). We then construct the itemset of a user using the 2,000 most reviewed games that the user has not yet played. At each run, we select a random user, the algorithm recommends then a game a_t from the corresponding itemset and observes a reward which is computed as $r_t = \langle \theta_*, a_t \rangle + \eta$.

Amazon books.[13] This dataset consists of reviews from 2,878,115 users for 7,797,481 books. We filter the users such that we keep only users that reviewed more than 350 books. This reduces the number of users to 648. The book vectors are then constructed by concatenating one-hot encoding of users and book features (categories, price, number of reviews, minimum/maximum/median/mean of ratings.). The parameter θ_* corresponding to a user is estimated using linear regression between books vectors reviewed by them and their corresponding ratings. Similar to Steam, we then construct the itemset of a user using 1,000 most reviewed books that the user have not yet reviewed. At each run, we select a random user, the algorithm recommends then a book a_t from the corresponding itemset and observes a noisy linear reward of similar form to Steam.

Yahoo.[14] This dataset contain reviews of 271 articles from 29,849,370. Users are filtered such that only users that interacted with more than 50 articles and clicked on more than 5 articles are kept, reducing the number of users to 1,550. The article vectors consist of one-hot encoding of the user(s) having interacted with the article and the article features directly taken from the dataset. The parameter θ_* corresponding to a user is estimated using linear regression between article vectors which the users interacted with and their corresponding interactions. The itemset

[10] https://grouplens.org/datasets/movielens/
[11] https://pjreddie.com/projects/mnist-in-csv/
[12] https://kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam
[13] https://amazon-reviews-2023.github.io/data_processing/index.html
[14] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=49

of a user consists of 200 articles with which the user has not interacted. At each run, we select a random user, the algorithm recommends then an article a_t from the corresponding itemset and observes a noisy linear reward between the article vector and the user vector.

2.2 Contextual

MovieLens 25M. We filter users such that we keep only 20 users. To build the vector representation of a movie, we extract the same features as in the non-contextual case. User vectors are context vectors and are constructed using one hot encoding. At each iteration, we proceed as follows: we select a user randomly (context vector), and construct a contextual itemset of 3000 most rated movies (concatenation of selected context vector and items). Then, the reward observed by an algorithm when recommending a movie is 1 if the user has interacted with the corresponding movie in the original dataset and 0 otherwise.

Yahoo.

We filter users such that we keep only 10 users. Article vectors are built as in the non-contextual case. User vectors are context vectors and are available in the dataset (6-dimensional vectors). We then proceed as in Movielens dataset (select a user, construct contextual itemset, observe a reward equal to 0 or 1). In this dataset, we select 200 most reviewed articles.

Steam. We filter users such that we keep only 10 users. Games vectors are built as in the non-contextual case. Users vectors are context vectors and are constructed using one hot encoding. We then proceed as in Movielens and Yahoo datasets (select a user, construct contextual itemset, and observe a reward equal to 0 or 1). In this dataset, we select the 2000 most rated games.

In general, as real-world datasets are characterized by a large number of users and items, we focused on the most relevant users – those who have interacted with a sufficient number of items – allowing a good application of the algorithm with a low number of iterations constraint. Also, these thresholds were set differently for each dataset, based on the number of already available features and to ensure reasonable execution times across various algorithms.

A Notation Table

The Table 1 summarizes the notation used in the paper.

B Algorithm Details

In this section, we present Algorithm 1 that shows how the sketching matrices are updated at each iteration in the case of linear bandits algorithms that use sketches (Kuzborskij et al., 2019; Chen et al., 2020).

T	Maximum iterations
d	Original dimension
n	Number of actions (items)
m	Sketching dimension
k	Projection dimension
λ	Regularization factor of $l2-$least squares
η	Noise added to the reward $\eta \sim \mathcal{N}(0, \sigma^2)$
a_t	Selected action at time t
X_t	List of selected actions $X_t = [a_1, a_2, ..., a_t]$
V_t	Covariance matrix
θ_*	Optimal parameter vector
r_t	Reward observed at time t
$\hat{R}_T$	Cumulative estimated regret at time T
S_t	Sketched selected actions list
$\tilde{V}_t$	Sketched covariance matrix
ε	Projection error or distortion $\in (0, 1)$
Φ	$k \times d$ projection matrix
x_a	Projected action Φa
μ_*	Projected model parameter $\Phi\theta_*$
$[T]$	Set from 1 to T
$\hat{x}$	Estimator of parameter x
$\|x\|_{1,V}$	$\|A^{1/2}x\|_1$
$\|x\|_{2,V}$	$\sqrt{x^\top V x}$

Table 1: Notation used in the paper.

Algorithm 1 UpdateParameters

Require: S_t, H_t, $a_t \in \mathbb{R}^d$, α_{t-1}

$S_t \leftarrow \left[S_{t-1}^\top, a_t\right]^\top$
if S_t has $2m$ rows **then**
  Compute SVD: $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] \leftarrow \text{SVD}\left(S_t\right)$
  $\delta_t \leftarrow \sigma_m^2\left(S_t\right)$ ▷ Get the m-th singular value
  $\widehat{\mathbf{\Sigma}} \leftarrow \sqrt{\max\left\{\mathbf{\Sigma}^2 - \delta_t \mathbf{I}, 0\right\}}$
  $S_t \leftarrow \widehat{\mathbf{\Sigma}}\mathbf{V}^\top$ ▷ Update S_t and remove 0 valued rows
  $H_t \leftarrow \left(\widehat{\mathbf{\Sigma}}^2 + \alpha_t \mathbf{I}\right)^{-1}$
else
  Compute H_t using (1)
end if

The efficient update for the matrix H_t is given in Chen et al. (2020), such that:

$$H_t = \begin{bmatrix} H_{t-1} + \mathbf{p}\mathbf{p}^\top/\mathbf{q} & -\mathbf{p}/\mathbf{q} \\ -\mathbf{p}^\top/\mathbf{q} & 1/\mathbf{q} \end{bmatrix}, \tag{1}$$

where $\mathbf{p} = H_{t-1}S_{t-1}a_t$ and $\mathbf{q} = a_t^\top a_t - a_t^\top S_{t-1}^\top \mathbf{p} + \alpha_t$.

C Table Comparison of Algorithms

Table 2 summarizes the upper bound regret and the complexities in terms of time and space for each algorithm and T iterations.

Algorithm	Regret	Time	Space
Baseline algorithms			
ConfidenceBall1 (Dani et al., 2008)	$\tilde{O}\left(d^{\frac{3}{2}}\sqrt{T}\right)$	$O(d^2T)$	$O(d^2)$
LinUCB (Chu et al., 2011)	$\tilde{O}\left(\sqrt{dT}\right)$	$O(d^2T)$	$O(d^2)$
Covariance Matrix Sketching			
SOFUL (Kuzborskij et al., 2019)	$\tilde{O}\left(\begin{array}{l}\sqrt{T}\left(1+\Delta_T\right)^{\frac{3}{2}}\cdot \\ \left(m+d\log\left(1+\Delta_T\right)\right)\end{array}\right)$	$O(mdT)$	$O(md)$
CBSCFD (Chen et al., 2020)	$\tilde{O}\left(\begin{array}{l}\left(\sqrt{m+d\log\left(1+\Delta_T\right)}\right. \\ \left.+\sqrt{\Delta_T}\right)\sqrt{mT}\end{array}\right)$	$O(mdT)$	$O(md)$
Random Projections			
CBRAP (Yu et al., 2017)	$\tilde{O}\left(k\sqrt{T}+\varepsilon\sqrt{k}T\right)$	$O(k^2T)$	$O(k^2)$
ConfidenceBall1_FJLT (**??**)	$\tilde{O}\left(k\sqrt{T}+2\varepsilon T\right)$	$O(k^2T)$	$O(k^2)$

Table 2: Comparison of algorithms for T iterations.

D Dataset Statistics

Table 3 describes statistics of the different datasets used in the experimental analysis of studied algorithms. The two columns *Users* and *Items*, describe statistics of initial raw data. While other columns describe the data we feed the algorithms (after processing).

			Contextual		**Non-contextual**	
Dataset	**Users**	**Items**	**d**	**n**	**d**	**n**
Synthetic1	-	-	-	-	100	200
Synthetic2	-	-	-	-	1,000	2,000
Amazon Books	2,878,115	7,797,481	-	-	1,366	1,000
MNIST	-	60,000	-	-	784	10
Movielens	162,541	62,423	833	3,000	813	1,500
Steam	7,495,040	37,032	1,818	2,000	1,807	2,000
Yahoo	29,849,370	271	1,561	200	1,555	200

Table 3: Dataset details.

E Experiments details

E.1 Code and Implementation

ConfidenceBall version As we use ConfidenceBall (CBall) in our experiments, we implement the ConfidenceBall-1 version by optimizing over the L1-norm ellipsoid, leading to a more computationally efficient algorithm. Indeed, when we use this norm we can optimize $\theta \in \mathcal{C}_t$ to solve the equation $a_t = \text{argmax}_a \max_{\theta \in \mathcal{C}_t} \langle \theta, a \rangle$ by generating $2d$ extremal points of the polyhedron rather than the infinite points present in the ellipsoid when using the L2-norm (ConfidenceBall-2), which would need to solve a convex optimization problem.

	MNIST		Movielens		Steam		Yahoo		Amazon	
Model	**Param**	**Time**	**Param**	**Time**	**Param**	**Time**	**Param**	**Time**	**Param**	**Time**
CBSCFD	m=50	41s	m=10	1071s	m=10	1183s	m=30	115s	m=50	533s
SOFUL		42s		950s		1232s		123s		530s
CBRAP	k=50	15s	k=50	22s	k=50	18s	k=100	10s	k=200	36s
CBall1_FJLT		35s		43s		42s		60s		241s

Table 4: Models parameters and CPU Times at T=2500 in the Non-contextual setting.

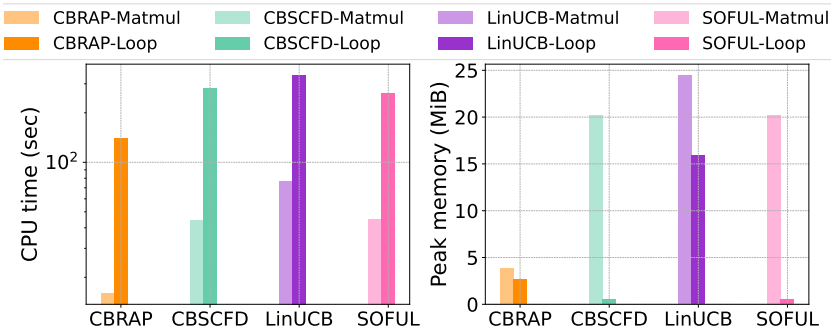


Figure 1: CPU time and Memory usage comparison per implementation for Movielens dataset.

`for` loop vs. Matrix Multiplication All results presented in this paper have been obtained using matrix multiplications only. Another possibility is to implement the presented algorithms more directly. Indeed, instead of performing a matrix multiplication involving the item set (all items at once), one can use a `for` loop to iterate through the list of items and estimate the gain of each individually.

In Figure 1, we compare the use of `for` loops implementation to matrix multiplications one, in terms of CPU time and memory usage for the Movielens dataset in the contextual setting and having $m = 10$, $k = 50$ and $T = 200$. As we can see the use of matrix multiplications leads to faster algorithms in terms of time, as shown in Figure 1, particularly for the CBRAP algorithm. However, we can also notice that this time efficiency leads to much higher memory consumption, especially for algorithms using sketches, as intermediate computations require the storage of matrices of order $n \times d$.

Consequently, the use of `for` loops appears to be a more memory-efficient approach, facilitating the manipulation of larger datasets on devices having low memory. Conversely, using matrix multiplications consumes more memory, especially if the data set is large (high dimension d and large number of items n), although this may be less of a concern if the memory is more plentiful.

E.2 Dimensionality Reduction Parameters Used

The two Tables 4 and 5 present the reduced dimensions used for each algorithm as well as the results in terms of CPU time and regret 4 (or Click Through Rate 5) at the end of experiments on different datasets. These values correspond to the ones of all figures presented in this paper.

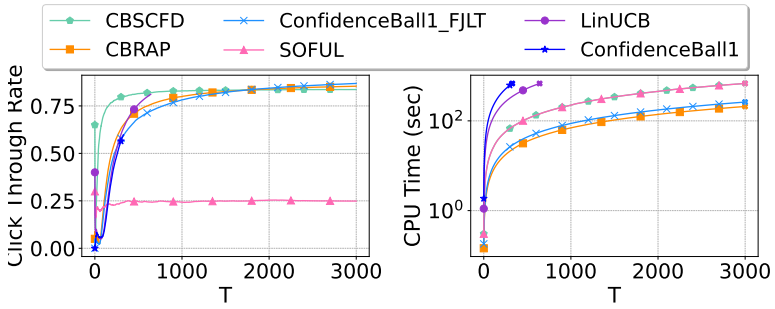
	Yahoo		Movielens		Steam	
Model	**Parameter**	**Time**	**Parameter**	**Time**	**Parameter**	**Time**
CBSCFD	m=50	117s	m=10	681s	m=10	917s
SOFUL		116s		668s		911s
CBRAP	k=50	25s	k=50	208s	k=50	270s
ConfBall1_FJLT		49s		263s		305s

Table 5: Model Parameters and CPU Times at $T = 3000$ in the Contextual Setting.

F Additionnal Results

F.1 Click Through Rate

Results on Movielens dataset that can be seen in Figure 2 show that CBSCFD appeared to converge more rapidly towards a near-optimal estimation. However, it was subsequently outperformed at the end by algorithms using random projections.



(a) Movielens

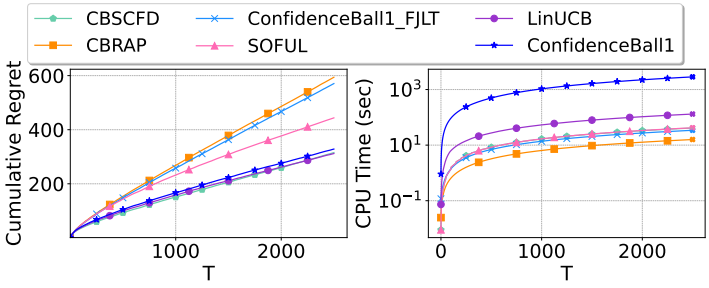
Figure 2: Click-through rate and CPU time per dataset.

F.2 Cumulative Regret and CPU time

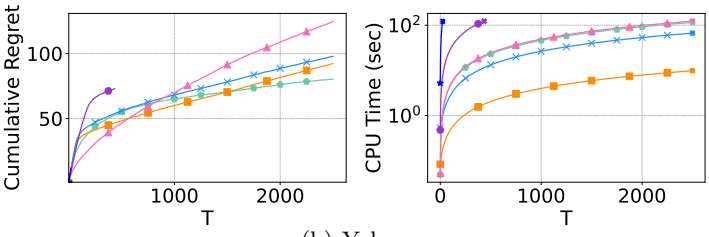
Cumulative regret evaluation on other datasets is presented in Figure 3. The best-performing algorithm in terms of mean of cumulative regret is the sketching algorithm CBSCFD as it can be seen in Figures 3a and 3b. However, sketching algorithms remain the slowest compared to algorithms that use random projections. Figure 3c shows that there exist some exceptions where random projections algorithms outperform sketching algorithms in terms of regret.

F.3 Impact of Parameters

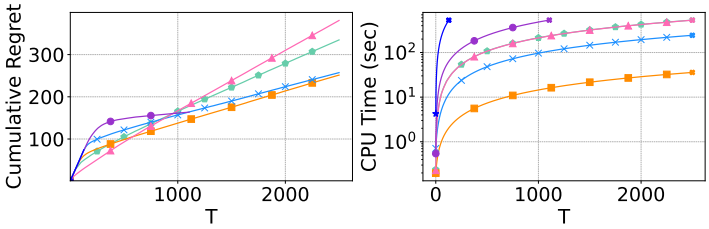
Reduced dimension Figure 4 shows how the reduced dimension impacts the cumulative regret and the CPU time. Increasing the dimensionality of projection for CBRAP algorithm leads to a more sublinear regret and reduce the variability as it can be seen in Figure 4a. For SOFUL 4b, increasing the sketch size does not appear to really improve the results.



(a) MNIST

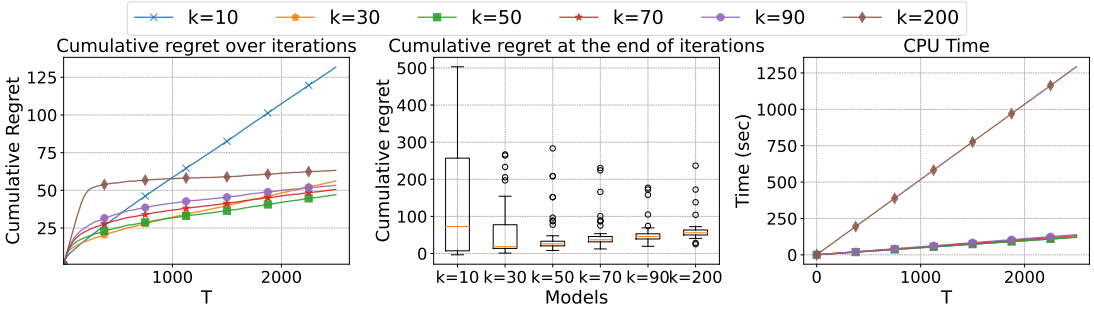


(b) Yahoo

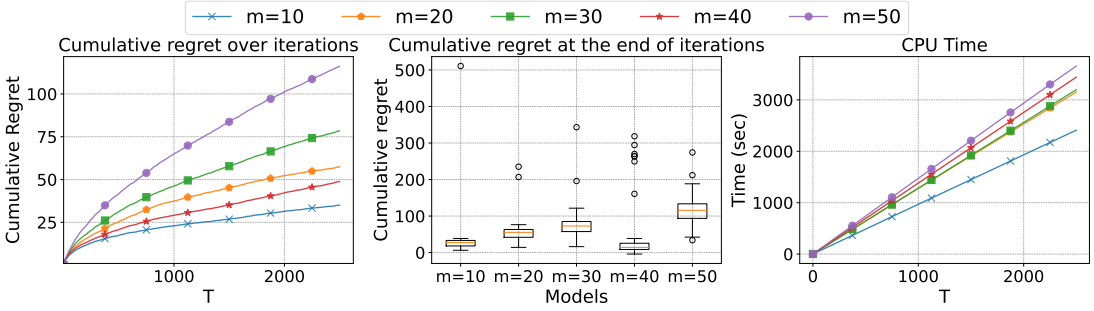


(c) Amazon

Figure 3: Regret and CPU time comparison per dataset.



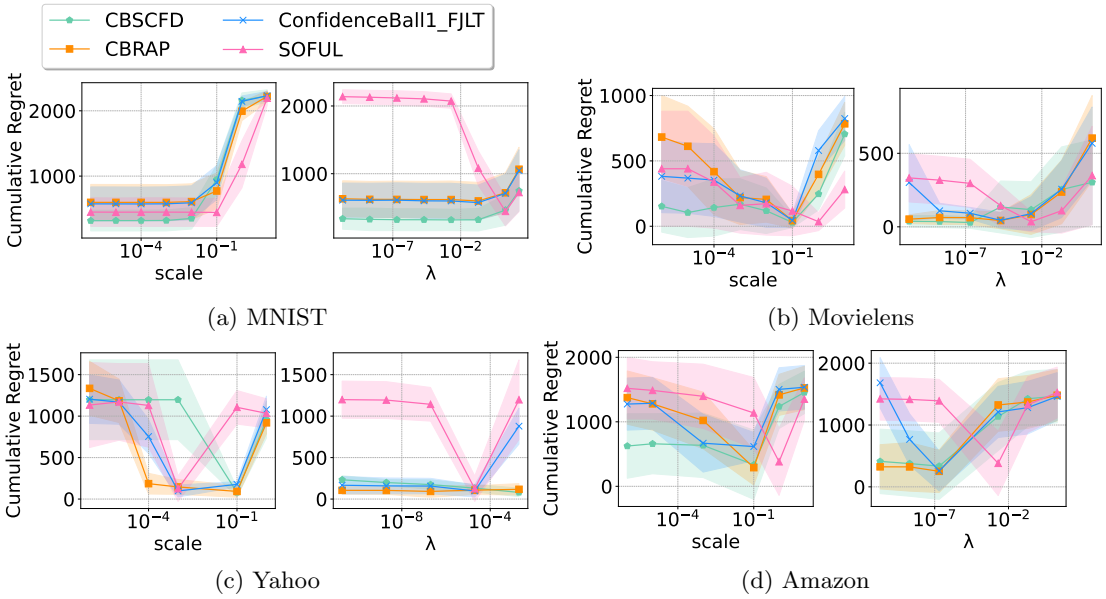
(a) CBRAP



(b) SOFUL

Figure 4: Impact of sketch and projection dimension on regret.

Other parameters Figure 5 displays the regret values at the end of iterations for different algorithms and datasets. We can see that the regret depend greatly on the regularization parameter λ and confidence bound scale, with each algorithm having an optimal value that minimizes cumulative regret and variance.



(a) MNIST (b) Movielens

(c) Yahoo (d) Amazon

Figure 5: Impact of parameters on regret.

References

Cheng Chen, Luo Luo, Weinan Zhang, Yong Yu, and Yijiang Lian. 2020. Efficient and Robust High-Dimensional Linear Contextual Bandits. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan, 4259–4265. `https://doi.org/10.24963/ijcai.2020/588`

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual Bandits with Linear Payoff Functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 208–214. `https://proceedings.mlr.press/v15/chu11a.html` ISSN: 1938-7228.

Varsha Dani, Thomas P. Hayes, and S. Kakade. 2008. Stochastic Linear Optimization under Bandit Feedback. `https://www.semanticscholar.org/paper/Stochastic-Linear-Optimization-under-Bandit-Dani-Hayes/551e19e5113cdff60a3c545d684fc4b9eb9a7306`

Ilja Kuzborskij, Leonardo Cella, and Nicolò Cesa-Bianchi. 2019. Efficient Linear Bandits through Matrix Sketching. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. PMLR, 177–185. `https://proceedings.mlr.press/v89/kuzborskij19a.html` ISSN: 2640-3498.

Xiaotian Yu, Michael R. Lyu, and Irwin King. 2017. CBRAP: Contextual Bandits with RAndom Projection. *Proceedings of the AAAI Conference on Artificial Intelligence* 31, 1 (Feb. 2017). https://doi.org/10.1609/aaai.v31i1.10888 Number: 1.