PrivyPlot: Differential Privacy on Movie Ratings

Yuntian Yang
Brown University

Yiwei Zhang
Brown University

1 Abstract

The application of differential privacy to movie ratings presents a significant challenge within the realm of data privacy and personalized recommendation systems. The importance of this challenge lies in striking a balance between maintaining the privacy of individual users and preserving the accuracy and usefulness of the aggregated data for recommendation systems. The complexity of this problem is compounded by the diverse and dynamic nature of user preferences, as well as the vast quantity of data generated by online platforms.

We introduce PrivyPlot, a system designed to obscure the specific activity of individual users—specifically, whether a user has rated a particular movie—while still accurately reflecting overall rating trends and personal tastes. PrivyPlot leverages differential privacy algorithms to effectively anonymize user data. This approach allows us to maintain the integrity of user-specific information without compromising the collective insights needed for effective recommendation systems.

2 Introduction

Differential privacy is a rigorous standard for protecting individual data points within a dataset, ensuring that the removal or addition of a single record does not significantly affect the outcome of any analysis. This concept is particularly crucial in the context of movie ratings, where individual preferences can reveal sensitive information about a user's interests and demographics. As online platforms increasingly rely on user data to provide personalized experiences, the need to implement effective differential privacy measures without compromising the quality of recommendations becomes critical. This not only enhances user trust in privacy-preserving mechanisms but also adheres to evolving regulatory standards for data protection. Addressing this problem is vital for the sustainable growth of data-driven services and for safeguarding the digital privacy rights of individuals in the ever-expanding digital landscape.

In the specific context of movie ratings, individual preferences are not just about likes or dislikes; they are a reflection of personal experiences, cultural backgrounds, and even socio-economic statuses. The nuanced understanding of these preferences is what makes recommendation systems so valuable yet also raises significant privacy concerns. Differential privacy, in this setting, acts as a critical tool for decoupling the personal identifiers from these preferences, thus providing a layer of anonymity that is essential in today's digital age. However, the application of differential privacy in this domain is not without challenges. The primary challenge lies in maintaining the delicate balance between data anonymization and the retention of meaningful insights required for accurate recommendations.

In this paper, we present PrivyPlot to allow for the protection of individual data points without losing the essence of the aggregated data. We delve into various algorithmic strategies that can intelligently aggregate data, ensuring that the individual cannot be identified while the collective pattern remains intact. Moreover, our work acknowledges and addresses the dynamic nature of online platforms and user behaviors. PrivyPlot is designed as a stream processing system, handling each individual movie review in real-time. This approach allows for immediate application of differential privacy techniques as data flows in, ensuring that privacy is maintained at every moment of data interaction.

We evaluate the efficacy of PrivyPlot through a rigorous analysis of statistical dispersion using the Netflix Movie Rating Dataset [1]. The evaluation process involves comparing the statistical properties of the original dataset with those of the dataset processed by PrivyPlot, particularly examining how well PrivyPlot retains key trends and variances in user ratings while anonymizing individual contributions. This comprehensive analysis aims to validate the effectiveness of PrivyPlot in striking an optimal balance between data utility and privacy. The results of this evaluation not only demonstrate the feasibility of PrivyPlot but also contribute to the ongoing discourse on privacy-preserving techniques in data-driven environments.

3 Background

3.1 Threat Model

Our concern in the context of data exposure encompasses a variety of scenarios where user data may be compromised. These include:

1. The risk of malicious users deducing critical personal information from published rating statistics.

2. Potential information leakage during data transmission over networks.

3. The vulnerability of servers to infiltration by malicious hackers, leading to data leaks.

4. The possibility of an untrusted server collaborating with third-party entities.

5. The risk of user data leakage on client devices.

To mitigate the first risk, we implement a differential privacy algorithm on the server side. This approach ensures that each individual rating is noised before the aggregate statistics are published on the website. This method effectively obscures the details of individual ratings, thereby preventing malicious users from inferring personal information from the published data.

For threats 2, 3, and 4, which involve potential vulnerabilities of an untrusted server (whether intentional or accidental), we propose the application of local differential privacy (DP) on the client side. In this setup, each user's rating is obfuscated with noise on the local browser before being transmitted to the server. This method ensures that the server plays no part in the computation of the privacy-preserving data, removing the need to trust the server with raw user data. We operate under the assumption that the server will faithfully publish the received results without any alteration.

Regarding the fifth threat, involving the potential leakage of user data on client devices, one viable solution is the use of advanced hardware technologies, such as Intel's Software Guard Extensions (SGX) [2]. This technology provides a secure enclave for data processing, effectively shielding sensitive data from unauthorized access, even if the device is compromised. However, in our threat model, we assume that the user's device can be trusted and, therefore, do not delve deeply into this aspect.

3.2 Differential Privacy

Differential privacy (DP) is pivotal in our system to ensure the privacy of user ratings while maintaining the utility of the aggregated data for recommendation systems. DP achieves this by adding controlled randomness to the data or queries, making it difficult to infer information about any individual record. The key metric in DP is the privacy loss parameter, denoted as ε, which quantifies the privacy level, with lower values indicating stronger privacy. We mainly considered three types of DP throughout our design.

Laplace Mechanism adds noise from the Laplace distribution to the output of a function. The scale of the noise is proportional to the sensitivity of the function (maximum change in output with the addition or removal of a single record) and inversely proportional to ε. Gaussian Mechanism is similar to the Laplace mechanism but uses noise from the Gaussian distribution. It is often used when dealing with numerical data and requires careful calibration to balance privacy and utility. Google's differential-privacy library [3] provides the implementation of both.

Exponential Mechanism (EM) is used mainly for non-numeric queries. It selects and outputs an answer with a probability that decreases exponentially with the amount of privacy loss it incurs. In our rating system, we have chosen to adopt the EM, a decision motivated by the discrete nature of movie ratings. Movie ratings typically fall into a finite set of categories (such as 1 to 5 stars), making them well-suited for the EM's approach to handling discrete data sets. We elaborate on the choice in Differential Privacy Algorithm.

4 Design

Our system is architecturally divided into two core components: the frontend and the backend, each serving distinct yet complementary functions. The frontend of our system is designed using React. The primary focus of the frontend is visualization, offering users an intuitive and engaging experience. Under circumstances where a stronger threat model is required, our system can shift some of the responsibilities to the frontend. Specifically, the frontend can take on the task of noise computation, an integral part of implementing differential privacy. This ensures that all data transmitted to the backend is already processed for privacy and provides users with the assurance that their personal data is being handled locally. We have intentionally modeled the layout to mirror that of the popular IMDB website, a decision driven by the familiar and user-friendly design of IMDB, which has been widely accepted and appreciated by movie enthusiasts. The frontend is based on an existing project, IMDB-clone-ReactApp [4].

The backend forms the backbone of our system and is developed using Django. Paired with a MySQL database, this combination offers robustness, scalability, and ease of data management. The backend is responsible for the heavy lifting of data processing, application logic, and interaction with the frontend. It provides efficient handling and querying of large datasets of movie ratings and provides RESTful API endpoints for the frontend to retrieve and display data. When a movie becomes a hotspot, leading to a significant influx of ratings in a short period, the backend is designed to switch from its standard real-time processing mode to a batch-process mode. This strategic shift is aimed at managing the increased computational load effectively while maintaining system performance and integrity.

4.1 Backend System Design

The backend system serves at a port on the server, listens to incoming requests from users or developers, and communicates with MySQL database to read or write data. It consists of 13 backend APIs. 6 of them are used to generate various charts to visualize the data and evaluate the system. For example, one of them is responsible for generating the scatter chart for the variance of all the movies, where one data point in the chart represents the variance for one movie's rating distribution. 3 of them are responsible for the IO (importing and exporting) of the data when we need to process movie ratings in batches. 3 of them are responsible for returning the information or ratings of a movie or a user in JSON format. They are consumed by the frontend system. The only API left is the "rate" API, the core of this system.

When the user submits a rating for a movie, "rate" API stores this rating in the MySQL database, opens up a new thread to add noise to this rating, and returns a message to client showing that the rating is stored successfully. In the new thread, we utilize a differential privacy algorithm to add an appropriate noise to one rating and store the result in the MySQL database once the noise for the rating is computed.

4.2 Differential Privacy Algorithm

There are several requirements for our DP algorithm:

1. We need to add noise to every single rating, not the overall ratings for a movie/user, as they might change significantly over time;

2. The noised result, noise + rating, must be within a reasonable range. Usually, the sum of noise and rating must be an integer between 1 and the maximum rating allowed for the rating system;

3. The noised result needs to reflect the rating trend of a movie or a user. It means that the average noise of a movie or a user needs to be not too far from 0;

4. The algorithm needs to be **online**: we need to add noise to a rating once it's sent from the client side and available.

Considering all those factors, we decided to choose the exponential mechanism as our DP algorithm. The exponential mechanism allows us to choose a noised result from a finite set. One way to realize the exponential mechanism is as follows: calculate a noisy score for each possible candidate in a finite set, then add random noise to each score, finally, we choose the candidate with the maximum score as the noised result [5]. This is called Report Noisy Max.

In our case, the finite set in the exponential mechanism is the set consisting of integers from the range [1, MAX_RATING], where MAX_RATING is a constant representing the maximum possible rating for the movies.

4.3 Noisy Score in Exponential Mechanism

How can we calculate the scores for each integer candidate in the range [1, MAX_RATING], given an available rating for a movie? Considering that the score of a rating should also reflect the trend of the average rating for both the movie and the user of the rating, we use the following formula to calculate the score for each candidate x:

$$score(x) = 3 * movie\_score(x) + 2 * user\_score(x) + Lap(1)$$

$Lap(1)$ means the random Laplace noise with scale=1.

The movie_score for a candidate depends on the proportion of ratings with score equals x in all the ratings for a movie. It reflects the trend of the rating distribution for a movie. Suppose the movie id of the rating is m:

$$movie\_score(x) = \frac{count\_ratings(movie = m, rating = x)}{count\_ratings(movie = m)}$$

The user_score is a score that makes sure that the noised rating, x, makes the average noise of a user closer to 0. To calculate the user_score, we first measure the farness of x to the average noise of a user's ratings. Suppose the average noise of a user's rating is $user\_avg\_noise$, then the best value of the noise for the current rating is $-user\_avg\_noise$, as in this way it balances out the extra noise and makes the average noise of a user close to 0. Suppose the current rating value is r:

$$farness(x) = e^{abs(x-r-(-user\_avg\_noise))}$$

So when the current noise is $-user\_avg\_noise$, $farness(x)$ will have its minimum value 1. However, we want the candidate closest to $-user\_avg\_noise$ to have a maximum, positive user score. One way to do so is to use a number bigger than all possible values of $farness(x)$ to minus $farness(x)$ in user_score. Because e^x grows too quickly, we set an upper bound for $farness(x)$ and drop those very big values (in this case, those bigger than 10 times the minimum possible $farness(x)$ values). Then we use the difference between the upper bound and $farness(x)$ as part of user_score:

$$uscore\_unnormalized(x) = max(\min(farness) * 10 - farness(x), 0)$$

Finally, we normalize the user_score array and get the user_score:

$$user\_score(x) = \frac{uscore\_unnormalized(x)}{sum(uscore\_unnormalized)}$$

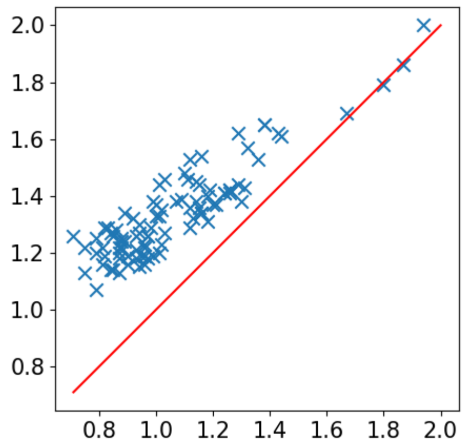


Figure 1: The variance of rating distribution for 94 movies, each movie has 1,000 to 100,000 ratings. Values on X axis are the variance of rating distribution calculated based on original ratings. Values on Y axis are those based on noised ratings.

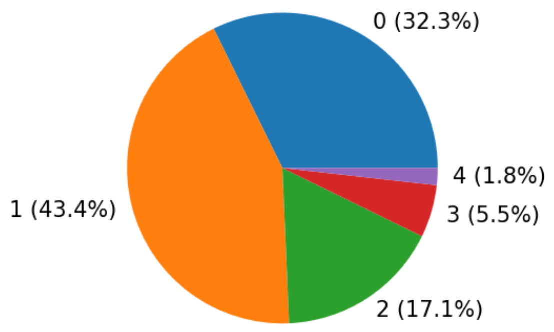


Figure 2: Noise value stats for Netflix Movie Rating Dataset. This pie chart shows the distribution of the absolute value of the noise value for 1,190,988 ratings.

5 Implementation

The backend system is implemented in 600+ lines of Python code based on the Django framework. The frontend system is based on an existing project, and we modified 100+ lines of code in Javascript.

6 Evaluation

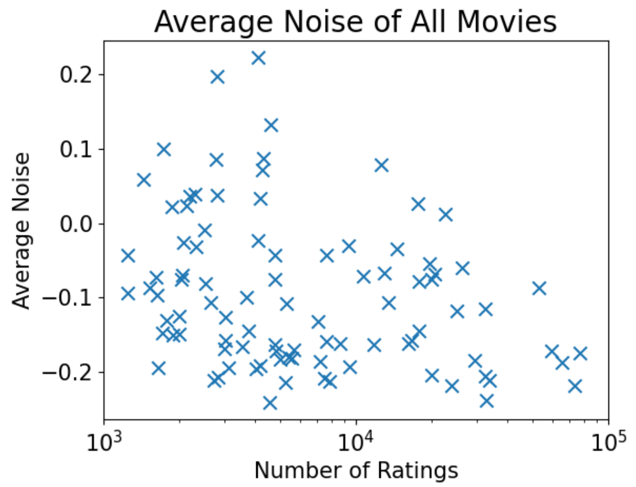
We evaluated our project using an existing dataset, Netflix Movie Rating Dataset [1]. We apply our algorithm to the ratings whose movie_id <= 305 (93 movies in total) or user_id <= 1894 (100 users in total) in the dataset. Finally, we added noise for 1,190,988 ratings in total. For the first 93 movies with the smallest movie_ids in the dataset, each movie has 1,000 to 100,000 noised ratings. In this dataset, MAX_RATING is 5, meaning possible ratings are 1, 2, 3, 4, 5 only.

Firstly, we evaluate the statistical dispersion of ratings for a movie by calculating the variance of their ratings based on original ratings and noised ratings. We showed the results in a scatter plot (Figure 1) where each scatter in the plot represents one movie. We also showed a line representing function $y = x$ as the baseline for comparison. The chart shows that 98% (92/94) of the noised movie ratings have a bigger variance compared to the variance of original ratings, proving the effectiveness of our differential privacy algorithm. As the original variance of movie ratings gets bigger, it's harder for us to increase the dispersion for the ratings.

Since we successfully made the rating distribution more dispersed, how many ratings are changed after we apply our differential privacy algorithm? How large is the change? For each possible absolute noise value, we count how many times it appears for all noised ratings and draw a pie chart (Figure 2). We found out that we added noise other than 0 for more than $\frac{2}{3}$ of the ratings, and most noise values are between [-2, 2], making it harder for hackers to guess the ratings of users.

One more requirement for our differential privacy algorithm is that the rating trend for a movie or a user needs to be maintained, in other words, the average noise value for a movie or a user needs to be close to 0. We prioritize maintaining the average noise for movies compared to that of users in our algorithm. We draw 2 scatter charts (Figure 3) for the average noise of 93 movies which each has more than 1,000 ratings and 100 users who each has more than 50 ratings. The average noise for those movies is mainly in the range [-0.2, 0.2]. The average noise for users is mainly in the range [-0.5, 0.5]. This result shows that our algorithm keeps the average noised rating around the original rating for both movies and users with enough ratings.

Does this also work when the number of ratings is small for a movie? We studied the trend of average noise for some movies as the number of ratings increases for them. We found out that the average noise becomes stable for a movie as more data are available after the initial fluctuations when the number of ratings is small. We depicted the line chart showing the trend of average noise for a specific movie (movie_id=30) in our dataset (Figure 4).



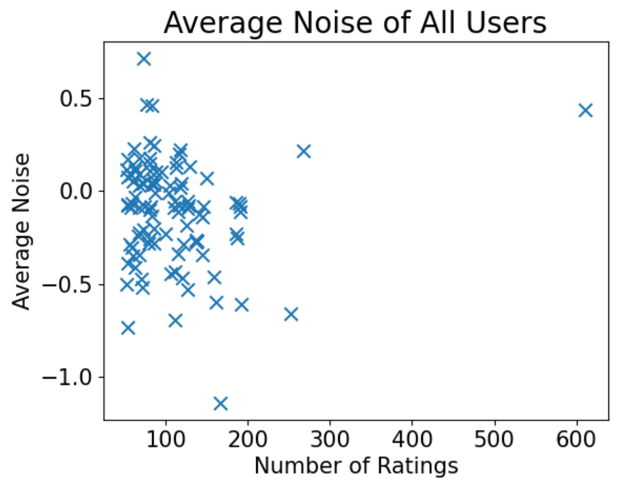


Figure 3: The average noise for 93 movies with more than 1,000 noised ratings and 100 users with more than 50 noised ratings.

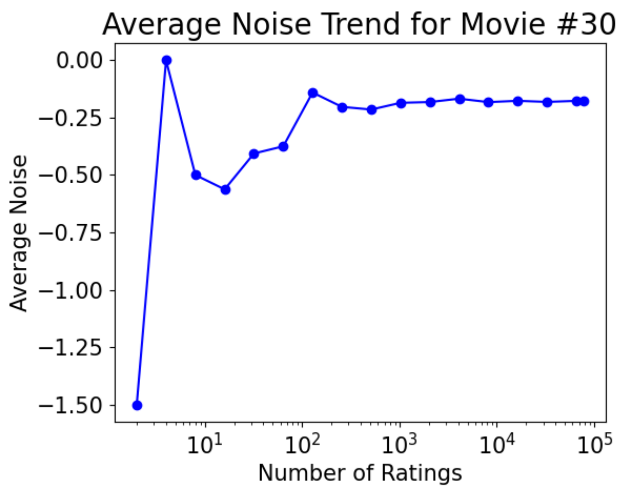


Figure 4: The average noise trend for a movie. We select the movie with movie_id=30 as our example.

7 Individual Contribution

Please see the GitHub commit history for our project at https://github.com/ZhangYW18/movie_rating.

References

[1] RISHIT JAVIA. *Netflix Movie Rating Dataset*. Kaggle, 2023. https://www.kaggle.com/datasets/rishitjavia/netflix-movie-rating-dataset.

[2] Phillip Nguyen and Alex Silence. *DuetSGX: Differential Privacy with Secure Hardware*. arXiv, 2020. https://arxiv.org/abs/2010.10664.

[3] google. *differential-privacy*. GitHub, 2023. https://github.com/google/differential-privacy.

[4] PrinceRaaaj. *IMDB-clone-ReactApp*. GitHub, 2023. https://github.com/PrinceRaaaj/IMDB-clone-ReactApp.

[5] Joseph P. Near and Chiké Abuah. *The Exponential Mechanism - Programming Differential Privacy*. GitHub, 2023. https://programming-dp.com/ch9.html.