Local Differential Privacy in Voting

Bernardo David, Rosario Giustolisi, Victor Mortensen and Morten Pedersen

IT University of Copenhagen, Denmark

Abstract
Voting allows groups of people to make collective decisions. Privacy is an important part of voting as it helps to prevent the manipulation of voters and control the outcome of an election. Local differential privacy (LDP) allows users to share private data with an aggregator and strictly limits how much information an aggregator can gain about individuals in a dataset. In this work, we test the potential use of multiple LDP mechanisms in a binary First-Past-The-Post voting system and a multiple option setting that, when combined with the binary setting, corresponds to a Two-Round voting system. Results show that Randomized Response works well in the binary setting for population sizes of 100.000 voters or more, with ϵ-values between 0.1-1. By allowing the system to reject close votes, the Randomized Response mechanism never altered the outcome of a vote, meaning that integrity was not broken. Results are different in the multi-option setting, which we test against data from the 2019 Danish general election. By using Basic One-Time RAPPOR with a population size of at least 1.000.000, and allowing for the mechanism to reject votes that are too close, it may be possible to use LDP in this setting as well. This will, however, depend on how often such a mechanism would end up rejecting a held vote.

1. Introduction

Elections are typically held in order for a group to make collective decisions. When the opinions of individuals within the group are seen as sensitive, the voting system needs to ensure privacy for its voters, as otherwise issues of coercion or self-censoring might arise. Traditionally, this is ensured by having people vote in secret and then dropping their ballot in a closed box, where it is mixed with the ballots of other voters. When voting electronically, this becomes slightly more complex, since a system needs to both verify the identity of the voter and keep their vote private [1].

Differential privacy (DP) is a method that offers rigorous privacy guarantees to individuals in a dataset by introducing noise [2]. It allows for a mathematical quantification of the privacy budget, independent of any external data that might exist. DP assumes that there exists a trusted data aggregator who stores the raw dataset and applies DP before releasing it. For voting, the repercussions for the voter might be grave if the aggregator misuses their data or accidentally leaks it. Local differential privacy (LDP) allows users to anonymize their data before sending it to the aggregator. LDP gives users plausible deniability for the values they report to the aggregator. This means that by the time the aggregator receives data from a user, they do not gain a significant amount of information about any individual user. However, this comes with the cost of greater levels of noise [3, 2]. This raises some questions about whether or not LDP can be used for voting systems, or if the introduced noise poses a too high risk of changing the

ITASEC 2023: The Italian Conference on CyberSecurity, May 03–05, 2023, Bari, Italy

result of the vote.

Contribution. This work examines the potential of applying LDP to voting, to see if it is possible to do so without violating the integrity of an election. In particular, we try to find suitable LDP mechanisms as well as make recommendations towards what values of ϵ should be used in combination with the number of voters needed. We discuss whether or not it makes sense to create LDP voting systems in practice, as well as how such systems should be applied. Although optimality and information theoretic bounds of LDP mechanisms have been studied, LDP mechanisms include constants and parameters that can affect the feasibility of a mechanism in a domain, such as voting. The goal is to examine empirically whether and when it makes sense to apply LDP to voting.

We test the performance of five different LDP mechanisms on synthetically generated voting data. The mechanisms are tested for two different voting settings: a binary First-Past-The-Post system and a multiple option system that, if combined with the binary setting, could represent a Two-Round voting system. We find Randomized Response works best in the binary setting, and that Basic One-Time RAPPOR works best in the multi-option setting. In the binary setting, we suggest having a population size of at least 10^5 and increasing this to 10^6 in the multi-option setting. ϵ-values of 0.5 to 1 work well depending on population size and voting setting. However, in both settings we find that close votes will have to be rejected, which poses further challenges.

1.1. Background

In the LDP model, each individual data holder adds noise to their own data before sending it to the aggregator. The drawback is that the total noise will be much larger than for the classic *central differential privacy* (CDP) model, which requires a trusted data aggregator to add noise. This means that either more users or lower privacy guarantees are required to get accurate results. LDP essentially ensures that even if a participant's data is leaked, they still have plausible deniability as the LDP mechanism might have changed the values of their original data. In practice, each participant applies a randomization mechanism on their dataset of size $n = 1$ before sending it to the aggregator. A randomized algorithm, M, satisfies ϵ-local differential privacy where $\epsilon > 0$ for any input v_1 and v_2 iff

$$\forall y \in Range(M) : Pr[M(v_1) = y] \leq e^{\epsilon} \cdot Pr[M(v_2) = y]$$

The difference between CDP and LDP is thus that in CDP, the randomization mechanism is applied to a dataset of n users, whereas in LDP, each user applies the randomization mechanism to their own data. The reason why one would prefer LDP to CDP for voting systems is that it would provide stronger privacy guarantees for each individual voter, as LDP gives them plausible deniability if the aggregator misuses or accidentally leaks the data. When using LDP for voting, each participant inputs their vote to an LDP-mechanism, and the locally differentially private vote is sent to the aggregator. In particular, we test the performance of Randomized Response [4], Laplacian and Gaussian mechanisms [2], RAPPOR [5], and Local Hashing [6] on synthetically generated voting data. In each setting, we run several experiments with varying vote distributions, population sizes and ϵ-values. We run each experiment multiple times and measure the average absolute error of each mechanism, as well as how often the mechanisms successfully rank the voting options in the right order.

2. Related Work

Research on voting has been tackled from many different perspectives. In social choice theory, Tactical voting [7] examines voter behavior in a First- Past-The-Post and Two-Round system to see how prevalent tactical voting is within each system. Plenty of research papers have been focusing on e-voting. A summary of development of e-voting, its current state, as well as how it might develop in the future is available in [8]. Several proposals for secure and private e-voting systems have been advanced using cryptographic protocols [9, 10] including multi-party computation schemes [11, 12]. Few major elections, such as national elections, are currently conducted using e-voting. One example of a place where e-voting is used in major elections is Estonia, whose voting system uses ID cards for voter identification and allows voters to cast their vote multiple times in order to reduce coercion and vote buying [13]. To ensure the privacy of the voter, no part of the system should posses the digitally signed e-vote and the private key. This is ensured by key management procedures, which means voters have to trust that they are followed.

Differential privacy is a technique that allows for rigorous privacy guarantees. An extensive introduction to the topic can be found in [2] and [3]. LDP is typically used in settings where the data aggregator cannot be trusted, and many LDP mechanisms have been proposed and compared in various studies [5, 14, 6]. To the best of our knowledge, there has been no work that combines LDP with voting. Cryptographic schemes such as secure multi party computation can be used to build highly secure voting systems, but they are often much slower then LDP when dealing with million of voters, which is why it would be interesting to examine how much potential LDP offers in terms of voting.

3. Experiments

In each of the settings, we test multiple LDP mechanisms using different population sizes and ϵ-values in an attempt to empirically find the settings where LDP could potentially work for voting. Note that the goal of this work is to examine whether or not it is feasible to apply LDP to voting data without violating the integrity of the result. Thus, we do not qualitatively test the viability of applying LDP to voting in the sense that we do not examine the problem from a user perspective, i.e., we do not test whether or not it is possible to build an LDP voting system that users understand and trust.

3.1. Assumptions

This work does not consider the complete system one would need to build when implementing a real-world voting system. It does not attempt to create a real-life voting system with all it entails, such as the information that would be needed to be given to voters regarding how LDP works, in order to ensure that people trust the system. Neither does it attempt to deploy a secure system to hold a vote on. We leave it to others to create a full-scale voting system using LDP. We foresee that one main challenge is how to make such system end-to-end verifiable [15]. For simplicity, we also assume that everyone in the voter population casts a vote, and that no one casts a blank vote. It is also assumed that everyone who participates applies the LDP mechanism to their vote before sending it to the aggregator. Furthermore, we assume that only one vote is held such that ϵ-values do not add up. This simplifies our setup as we do not need to take composition into account when setting ϵ-values for our experiments. Using these simplifications, we test the effect that LDP mechanisms have on the outcome of a vote.

3.2. Setting

For binary voting, we measure the usefulness of a mechanism as the number of times where the mechanism does not change the overall result of the held vote. The result of a vote is a boolean measure of whether or not at least 50% of the population has voted in favor of the proposal. For multi-option voting, we instead focus on how many times the overall ranking of the options does not change after applying LDP, e.g., if options A, B, and C, are placed first, second, and third in the original ranking, we want to measure the number of times that our mechanism also rank parties A, B, and C as first, second, and third respectively. We measure this for all positions, i.e., we measure the number of times a mechanism gets the order of the ranking up to position $1, 2, \ldots, d$ correct.

The mechanisms are tested on multiple synthetic datasets with varying population sizes. The level of noise in LDP is generally higher than that of CDP [3], so we expect that a large number of voters is needed in order to achieve accurate results. In the binary setting, we therefore test our mechanisms on datasets with population sizes of 10^3, 10^5 and 10^6. Based on results from the binary setting, we use population sizes of 10^6 and 10^7 in the multi-option setting. In the binary setting, datasets are generated by flipping a biased coin n times, where n is the size of the population. This is preferred to storing the datasets locally in order to save storage space. The bias of the coin is set to different values in order to test how the LDP mechanisms perform when the vote is skewed in either direction. In this case, the worst-case dataset for our mechanisms is generated using a fair coin, meaning that the vote is split at roughly 50%. We expect this to be the worst case since having a close vote means that only a small amount of noise will be required in order to change the result. In the multi-option setting, we store a JSON-file where each key is the name of a dataset, and each value is a vector that contains the percentage of votes each option should receive, e.g., if the first position of the vector is 25, it means that the first option will get 25% of votes. This also saves storage space as we only need to store one number for each of the $d = 10$ voting options that a user has, rather than storing the actual vote of n users. In this setting, we create datasets with different distributions that we select in order to evaluate the performance of the LDP mechanisms. We test the mechanisms on data generated to have an almost uniform distribution, where there is a 0.1 percentage point difference between option i and i+1, as well as data generated such that each option gets half the votes of the previous option. We choose to test the mechanisms on these two datasets as we expect them to be close to the worst case and best case distributions respectively. We refer to these datasets as the *close* dataset and the *u_half* dataset.

To test the performance of the mechanisms on a more realistic voting distribution, we also create a dataset consisting of the percentage of votes achieved by the top 10 parties for the 2019 Danish general election. Since there were more than 10 parties to vote for in the 2019 Danish general election, we distribute the remaining votes evenly between all 10 parties. This dataset is referred to as the *dk_general_election* dataset. In Figure 1, we plot the distribution of each dataset.

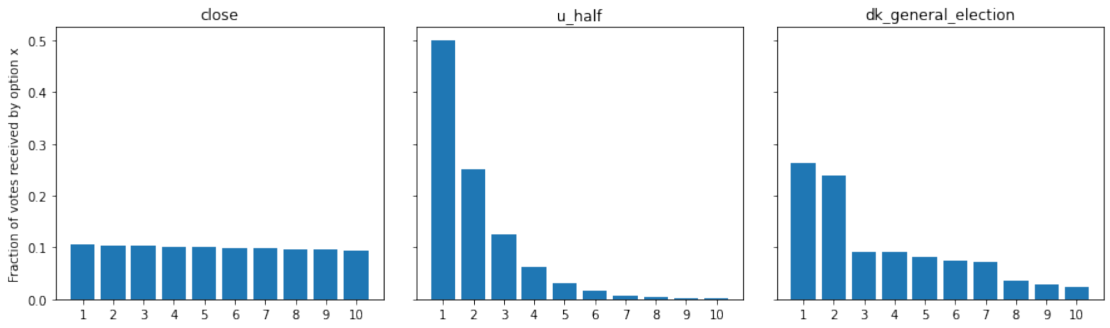

Figure 1: Distributions for the datasets used in the multi-option setting.

Finally, we also measure how many times a mechanism gets the top X options correct, where $X \in \{1, 2, \ldots, 10\}$. The general idea is that for a mechanism to be suitable, it should not change the ordering of what options got the most votes, as otherwise the result of the vote will change.

3.2.1. Application of mechanisms

For RR, we encode votes as integer values. In the binary setting, a vote of 1 correspond to a vote in favor of the proposal, and a vote of 0 correspond to one against the proposal. In the multi-option setting, a vote for i correspond to a vote for option i, where i is in the range $[0, d-1]$. In the binary setting, we create a variation of RR which we choose to call Threshold Randomized Response. The technique could potentially be implemented on any of the tested mechanisms, but it is only tested on RR as this is the mechanism that performs the best out of the ones we test. The idea behind the method is to create a threshold around 0.5, e.g., a threshold could be $(0.48, 0.52)$. Before deciding whether a majority is in favor of the vote, the mechanism checks whether its estimate of the number of votes in favor of the proposal lies within the threshold. If this is the case, then the mechanism rejects the vote, corresponding to a statement saying that the vote is too close for the mechanism to call. E.g., if a threshold of (0.48, 0.52) is used, and the RR mechanism estimates that a 0.485 fraction of people voted in favor of the proposal, then rather than concluding that the vote was not passed, it refuses to state the result of the vote. The idea is that a mechanism might be accurate enough for its result to be within a close threshold of the actual result, but if the actual result is that close to 50% voted in favor of the vote, then the mechanism might not be accurate enough to call the result of the election.

For the Laplace mechanism in the multi-option setting, a participant's vote is initially encoded as a one-hot vector where all but the i$^{\text{th}}$ index is set to 0 if the participant votes for option i. In this setting, the Laplace mechanism adds noise independently to each of the d indexes in the one-hot vector, i.e., a user first creates a one-hot vector, v, of length d, where all numbers are set to 0 except for the number at position i which is set to 1. This vector is passed on to the Laplace mechanism, which computes $v[j] = v[j] + Lap(0, \frac{2}{\epsilon})\ \forall j \in \{0, 1, \ldots, d-1\}$. In the binary setting, we encode each participant's vote as a single bit, corresponding to whether that user votes in favor of the vote or not. Here, one can interpret a vote cast by a participant as a one-hot vector of length 1. Such a vector only has L1-distance 1, which is why the scale of the added Laplace noise was smaller in this setting.

For the Gaussian mechanism, each vote is encoded in the exact same way as described for the Laplace mechanism, after which noise drawn from $\mathcal{N}(0, \sigma)$ is added. Noise is added in the exact same way as it is for the Laplace mechanism, and the same unbiased estimator is used. In our implementation of the mechanism, we first compute c as $\sqrt{2 \cdot ln(\frac{1.25}{\delta})}$ after which we compute the scale of the noise as $\frac{c \cdot \Delta_2 f}{\epsilon}$. We set $\delta = \frac{1}{n}$. This is the recommended upper bound for the δ parameter, but ideally it should be lower in order to reduce the risk of disclosure [3]. We set δ this high to get an upper bound on the mechanism's usefulness, as setting it any lower will only yield more noise. If the mechanism does not work for a δ this high, then it is unrealistic that it can be used in a real setting, and hence there would not be a reason to test further values of δ. It is expected that adding Gaussian noise will yield more noise as compared to adding Laplace noise due to the shape of the distributions [2].

We test the basic One-Time RAPPOR, since the full version of RAPPOR uses Bloom filters and hashing because user inputs may be infinitely diverse. However, the number of voting options is assumed to be well-defined, hence there is no need to use Bloom filters to reduce the dimensionality of user inputs. Thus, each user hashes their values onto a d-length vector by using the hash function that hashes value i to index i of the array. By using the basic version of RAPPOR where Bloom filters are not used, we reduce both the complexity of the implementation as well as the running time of the estimation part. Furthermore, since we assume that only one vote is held, we do not need to use both permanent and instantaneous randomized response.

Our implementations of the Local Hashing mechanisms use the hash function family consisting of all (g-1)-degree polynomials modulo g, where g is a prime number. For Binary Local Hashing, $g = 2$, thus our hash function family consists of all first-degree polynomials, i.e., $H(x) = a_0 + a_1 \cdot x \ mod\ 2$ where the constants of the polynomial are integers in the range $[0, 1]$. This means that the universal hash function family used in our implementation consists of four different hash functions - one for each of the possible polynomials, i.e. $\mathbb{H} = \{H_1(x),\ H_2(x),\ H_3(x),\ H_4(x)\} = \{(0{+}0x\ mod\ 2), (0{+}1x\ mod\ 2), (1{+}0x\ mod\ 2), (1{+}1x\ mod\ 2)\}$. In the non-binary setting, we set g to be the smallest prime number that is at least as large as d. Since we use $d = 10$, it means that our Local Hashing mechanism uses $g = 11$. In both Binary Local Hashing and Local Hashing, we encoded a vote as an integer in the range $[0, d-1]$, where $d = 2$ in the binary setting and $d = 10$ in the multi-option setting. We test both Binary Local Hashing and Local Hashing in the multi-option setting. For our Local Hashing mechanism, we set g to the smallest prime number that is at least as large as d (i.e., $g = 11$ for $d = 10$), since this will allow us to hash every value in $[0, d-1]$ into a unique value. For the case where $d = 2$, this setting is equivalent to Binary Local Hashing.

4. Results

This section covers the results for the various mechanisms tested. It is divided into binary voting and multi-option voting.

4.1. Binary voting

We measure how many times a mechanism does not change the result of the vote out of the 1000 times that each experiment is run. Results for the tested mechanisms, for the three different population sizes they were tested at are shown in Figure 2. Note that we only show results for the synthetic datasets generated where 20, 40, 45 and 50% of voters voted "Yes" to the proposal. We only display these results since the results are symmetric, e.g., results for the dataset where 80 % of voters voted "Yes" to the proposal are similar to results for the data where 20% did so.

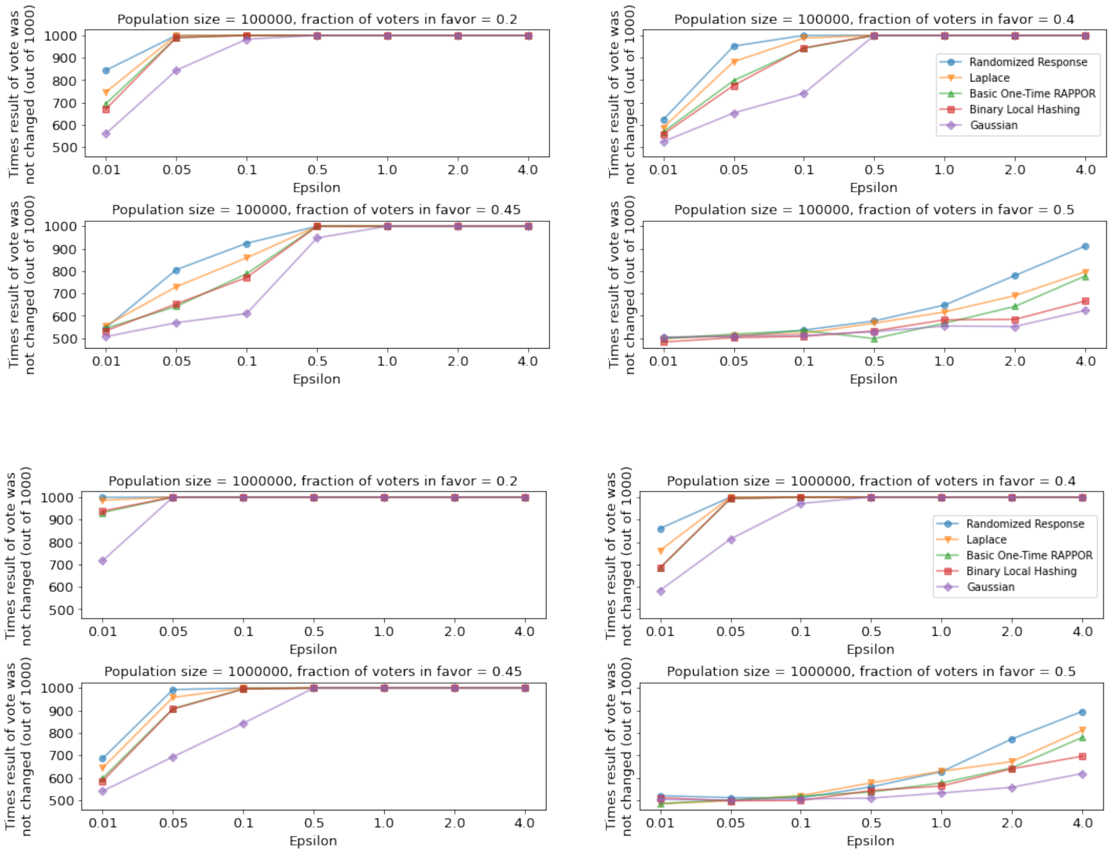

Figure 2: Number of experiments where result of vote was not changed after applying various LDP mechanisms at different ϵ-values.

The results generally show that once $\epsilon \geq 1$, then all mechanisms except for Gaussian noise never change the outcome of the vote, as long as the fraction of people voting in favor of the vote is within the range $[0, 0.48]$ or $[0.52, 1]$. With a population size of 10^5, ϵ can be decreased to

0.5 for all mechanisms except Gaussian without the mechanisms altering the result of the vote. ϵ can be further lowered to 0.1 when increasing the population size to 10^6. Since ϵ-values of more than 1 provide weaker privacy guarantees, these mechanisms should probably not be used in a voting system of only 10^4 people. However, for population sizes of 10^5 and 10^6, mechanisms are able to function somewhat well using ϵ-values of less than 1. We note that mechanism performance here correlates with their normalized average absolute error, i.e., Randomized Response performs the best, then the Laplace mechanism, followed by basic one-time RAPPOR and Binary Local Hashing, and Gaussian noise performs the worst. While results are promising, there is still one glaring issue: none of the mechanisms are able to perfectly estimate the result of the vote if close to 50% of voters are in favor of the proposal, regardless of population size and ϵ-value. This would be concerning in a real voting system, seeing as none of our tested mechanisms would be able to consistently estimate what the result of the vote should be if the held vote is close. Figure 3 shows the results of Threshold Randomized Response when run on

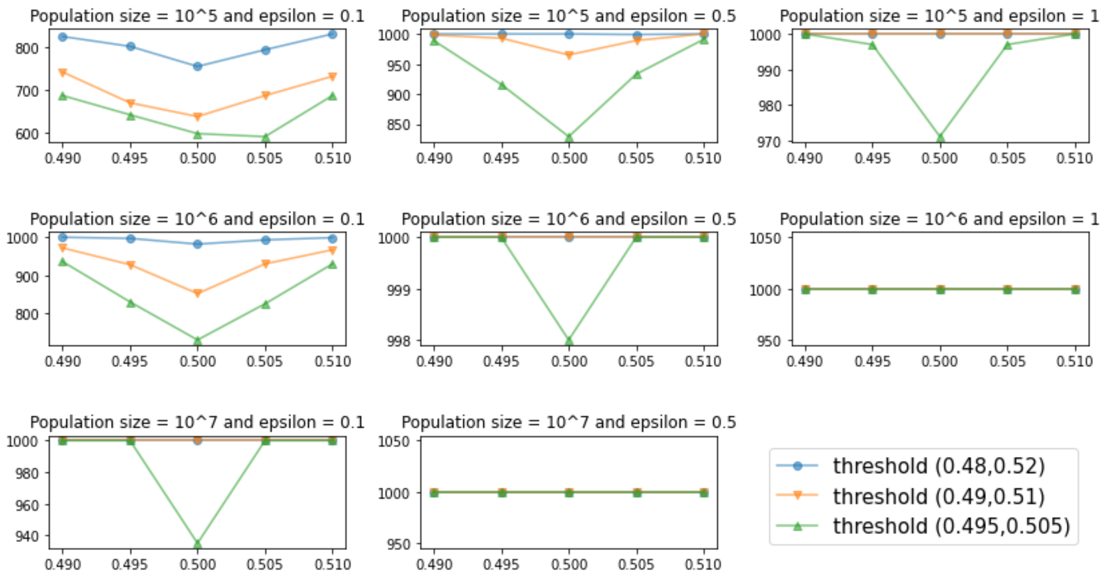

Figure 3: Number of times threshold randomized response successfully estimated the result of a vote for population sizes 10^5, 10^6 and 10^7, plotted for various values of ϵ and coin biases, and using different thresholds.

datasets of various sizes. For a population size of 10^5, we observe that the method only seems to reliably work when $\epsilon = 1$ with a threshold of (0.49,0.51) or wider. When the population size is 10^6, it can be seen that when ϵ is set to 0.1, the mechanism is unable to correctly estimate the result of the vote 100% of the time. We do observe that the (0.48,0.52) threshold nearly works, so by slightly widening the threshold, it might be possible to create a mechanism that functions at this privacy level with a population size of 10^6. When increasing ϵ to 0.5, the mechanism is able to use both a threshold of (0.48,0.52) and (0.49,0.51) without wrongly estimating the result of the vote, but the threshold of (0.495,0.505) is still too tight. At $\epsilon = 1$, all thresholds work. When the population size is increased to 10^7, we notice that both the (0.48,0.52) and the (0.49,0.51) thresholds work for $\epsilon = 0.1$. This means that as the population size increases,

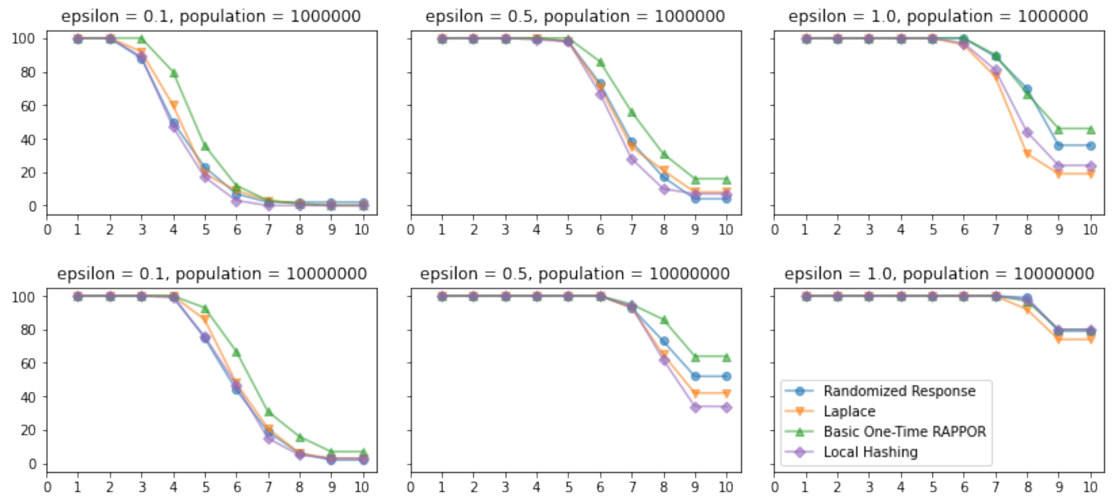

Figure 4: Number of times each mechanism correctly ranked the first x options of the *u_half* dataset for different population sizes and values of ϵ.

one can increase the privacy guarantees of their voting system by lowering the value of ϵ. We also notice that when $\epsilon = 0.5$, one would able to use the tightest bound we test for, namely (0.495,0.505), without violating the integrity of the vote.

4.2. Multi-option voting

It can be seen that all mechanisms achieve similar L2-errors. Basic One-Time RAPPOR performs the best out of all mechanisms, especially at the low value of $\epsilon = 0.1$. At higher values of ϵ, all mechanisms seem to perform similarly, although RR and Basic One-Time RAPPOR seem to be a little better. Finally, we show the results for how often each mechanism got the ranking of the top x parties correct, where x is in the range of [1,10]. Results for the *u_half* dataset are shown in Figure 4. Note that if a mechanism gets the ordering of voting options correct for position x, but has made an error in the ranking somewhere earlier, e.g., for $x > 5$, it might have ranked the option with the 5$^{\text{th}}$ most votes as number 4, then we do not count this towards a run where a mechanism got the top x correct. In order to count towards a correct top x, our mechanism would need to correctly rank all of the first x voting options. Performance-wise, we see similar results as before: all mechanisms perform similarly, but Basic One-Time RAPPOR seems to work slightly better than the others. For $\epsilon = 0.1$, the mechanisms are only able to consistently get the first 2 to 4 options correctly, depending on the population size and mechanism used. At $\epsilon = 0.5$, the mechanisms can rank the first 4 to 6 options correctly, which increases to 5 to 7 for $\epsilon = 1$. The difference between option 8 and 9 is about 0.2 percentage points, which suggests that if options are this close to each other, then mechanisms will fail to consistently make correct estimates of the rankings for the population sizes and ϵ-values we tested for. This is supported by results from the *close* and *dk_general_election* datasets.

5. Discussion

Our results show that LDP mechanisms producing lower levels of noise change the result of a vote less often than mechanisms that yield more noise, which intuitively makes sense. If a mechanism introduces less noise, then its estimate should be closer to the true result. In the binary setting, we saw that RR gave the best results, and that when combined with our threshold method, it was possible to create a mechanism that did not change the result of any of the 1000 votes it was run on. Based on our results, it seems that one would need to use a somewhat large ϵ of about 1 or higher if run on a population size of 10^4 or less. This could work, but to provide stronger privacy guarantees for voters, which is the core idea of using LDP in the first place, one would need to use a population size of around 10^5 people or more. For a population of 10^5, ϵ-values around 0.5 seem realistic, and for populations of 10^6 one may even use $\epsilon = 0.1$. We further observed that as the population size increases, not only can one guarantee stronger privacy for voters, but one is also able to set a tighter threshold around the 50% mark, reducing the risk of having to reject a vote. For the smaller population size of 10^5, one would need to use $\epsilon = 1$ with the tested thresholds in order to not violate the integrity of the vote. While we did not test the threshold method on other mechanisms, it seems plausible that the method would also work for mechanisms such as Basic One-Time RAPPOR, the Laplace mechanism and Binary Local Hashing - at least for some combinations of population sizes and ϵ-values. Depending on someone's needs, they can choose a mechanism accordingly.

When speaking of how realistic it would be to implement such an LDP voting system, the risk of having to reject a vote is something that has to be considered. Firstly, one would need to build a system that can somehow handle a rejected vote. This could for example be done by redoing the vote, but this would be inconvenient for voters and expensive for the people who organize the vote. Ideally, one should somehow build a system that relies on LDP by default, but where the system has something else to fall back on in case the vote is rejected by the LDP mechanism.

It should also be considered how often a vote would realistically be rejected. If all held votes are close, then our system is guaranteed to always reject them, rendering it useless. Looking at voting data from the US Senate elections, we see that the average margin of victory is around 20 percentage points, with exactly one election per election cycle (out of the around 33) being lower than 1 percentage point margin of victory. If this generalizes to other countries, then it seems that the threshold mechanism would not have to reject a lot of votes. However, whether this generalizes to other countries is currently unclear and would have to be examined further. We emphasize that having to reject a vote is a major drawback of our LDP system, since we generally cannot tell how close a vote is going to be before the election.

In the multi-option setting, Basic One-Time RAPPOR performed the best out of the mechanisms tested, although it tied with RR when $\epsilon = 1$. We saw that Binary Local Hashing performed far worse than any of the other mechanisms, and that its normalized average error remained at similar levels despite increasing both ϵ as well as the population size. This was due to the fact that Binary Local Hashing hashes input into a single bit. This meant that estimates for one half of the voting options were hashed to one value, and the other half was hashed to another, resulting in all options within each half having the same estimated number of votes. This makes the mechanism useless in the multi-option voting setting we tested.

We observed that when two options were close, all mechanisms struggled to consistently rank them in the proper order. While it was not tested, it seems plausible that the threshold method could be applied in this setting to correct for this. In a Two-Round system, this would correspond to checking whether the second-most voted option is too close to the third-most voted option, since the two most popular candidates move on to the second round in this system. Whether it would be realistic to use such a mechanism would depend on how votes are typically distributed in such a system, i.e., ideally the second- and third-most voted options should only rarely be close. We leave it to future works to examine how votes cast in such a system tend to be distributed.

Results for the *u_half* dataset show that for a population size of 10^6 and $\epsilon = 1$, most mechanisms are able to get the rankings of options up to and including option 6 correct. This means that mechanisms begin to struggle when the difference in votes between two options is around 0.4 percentage points. For a population size of 10^7, mechanisms were able to correctly rank the first 7 options, meaning that they only began to struggle once the difference between two options were around 0.2 percentage points. This hints that the threshold method might work in this case. For a population size of 10^7, it might even be realistic to use $\epsilon = 0.5$, but this would depend on how many votes would need to be rejected. The same can be said for the population size of 10^6 and $\epsilon = 1$, as mechanisms performed similarly across those two settings.

6. Conclusion

We examined the possibility of applying LDP to voting data. Privacy is an integral part of holding anonymous votes, and differential privacy provides rigorous, quantifiable privacy guarantees for data holders. This work's aim was to assess whether LDP can be applied to voting, and if so, make recommendations for the choice of mechanisms, ϵ-values, and population sizes. While many voting systems exist, this work mainly focused on two settings: a binary First-Past-The-Post system, and a multi-option system similar to the first round of a Two-Round voting system. In both settings, we created synthetic datasets of votes. In the binary setting, RR was the mechanism that performed the best, although the Laplace mechanism, Basic One-Time RAPPOR and Binary Local Hashing were not far behind. All mechanisms struggled to correctly estimate the outcome of a held vote when votes were split almost evenly between the two options. We therefore created a threshold method which, when combined with RR, provided a mechanism that would never wrongly estimate the result of the vote. However, this came with a risk of having to reject the result of a vote entirely. The size of the threshold could be set based on population sizes and ϵ-values. In the multi-option setting, Basic One-Time RAPPOR performed better than all other tested mechanisms, except at $\epsilon = 1$, where it tied with RR.

Finally, while it seems technically possible to build LDP voting systems that do not violate the integrity of a vote, it requires an estimate of the number of people who are going to vote in order to properly choose an ϵ-value. If such an estimate cannot be found, one may end up setting an ϵ-value that yields too much noise.

Acknowledgments

Rosario Giustolisi is supported by the Villum Foundation, within the project “Enabling User Accountable Mechanisms in Decision Systems”

References

[1] C. Faulí, K. Stewart, F. Porcu, J. Taylor, A. Theben, B. Baruch, F. Folkvord, F. Nederveen, A. Devaux, F. Lupiáñez-Villanueva, Study on the benefits and drawbacks of remote voting (2018).

[2] C. Dwork, A. Roth, The Algorithmic Foundations of Differential Privacy, now publishers inc., 2014.

[3] S. Vadhan, The Complexity of Differential Privacy, Springer, 2016.

[4] S. L. Warner, Randomized response: A survey technique for eliminating evasive answer bias, Journal of the American Statistical Association (1965).

[5] U. Erlingsson, V. Pihur, A. Korolova, Rappor: Randomized aggregatable privacy-preserving ordinal response, CCS (2014).

[6] T. Wang, J. Blocki, N. Li, S. Jha, Locally differentially private protocols for frequency estimation, USENIX Security Symposium (2017).

[7] A. Blais, S. Labbé-St-Vincent, J.-F. Laslier, N. Sauger, K. Van der Straeten, Strategic vote choice in one-round and two-round elections: An experimental study, Political Research Quarterly (2011).

[8] J. P. Gibson, R. Krimmer, V. Teague, J. Pomares, A review of e-voting: the past, present and future, Annals of Telecommunications (2016).

[9] D. Chaum, P. Y. A. Ryan, S. Schneider, A practical voter-verifiable election scheme, in: ESORICS, Springer-Verlag, 2005.

[10] B. Adida, Helios: Web-based open-audit voting, in: Proceedings of the 17th Conference on Security Symposium, SS’08, USENIX Association, 2008.

[11] A. Kiayias, M. Yung, Self-tallying elections and perfect ballot secrecy, in: PKC, Springer, 2002.

[12] F. Hao, P. Y. A. Ryan, P. Zielinski, Anonymous voting by two-round public discussion, IET Inf. Secur. (2010).

[13] E. Maaten, Towards remote e-voting: Estonian case (2004).

[14] Apple, Learning with privacy at scale (2017).

[15] J. D. C. Benaloh, Verifiable Secret-Ballot Elections, Ph.D. thesis, 1987.