Efficient Error Prediction for Differentially Private Algorithms

Citation for the original published paper (version of record):
Nelson, B. (2021)
Efficient Error Prediction for Differentially Private Algorithms
Proceedings of The 16th International Conference on Availability, Reliability and Security (ARES 2021)
http://dx.doi.org/10.1145/3465481.3465746

(article starts on next page)

Efficient Error Prediction for Differentially Private Algorithms

Boel Nelson
Department of Computer Science and Engineering,
Chalmers University of Technology
Gothenburg, Sweden
boeln@chalmers.se

ABSTRACT

Differential privacy is a strong mathematical notion of privacy. Still, a prominent challenge when using differential privacy in real data collection is understanding and counteracting the accuracy loss that differential privacy imposes. As such, the accuracy/privacy trade-off of differential privacy needs to be balanced on a case-by-case basis. Applications in the literature tend to focus solely on analytical accuracy bounds, not include data in error prediction, or use arbitrary settings to measure error empirically.

To fill the gap in the literature, we propose a novel application of *factor experiments* to create data aware error predictions. Basically, factor experiments provide a systematic approach to conducting empirical experiments. To demonstrate our methodology in action, we conduct a case study where error is dependent on arbitrarily complex tree structures. We first construct a tool to simulate poll data. Next, we use our simulated data to construct a least squares model to predict error. Last, we show how to validate the model. Consequently, our contribution is a method for constructing error prediction models that are data aware.

KEYWORDS

accuracy prediction, data privacy, differential privacy, empirical evaluation, error prediction, factor experiments, prediction model

1 INTRODUCTION

Adopting differential privacy in real systems is ultimately an issue of properly understanding the impact differential privacy will have on accuracy. In other words, if an analyst cannot predict the accuracy loss an algorithm will cause, they will be hesitant to use the algorithm. As such, understanding the accuracy loss induced by differential privacy is crucial to differential privacy being deployed in real systems.

In the literature, the accuracy of a differentially private algorithm is often evaluated analytically through Chernoff bounds, such as by Kasiviswanathan et al. [11]. Here, the authors introduce a metric for error, namely misclassification error, which is applicable in their domain. However, the general Chernoff bound they provide requires that there exists a definition for error, i.e. a unit of measurement for the inaccuracy introduced by differential privacy. As such, if the relationship between input variables and error is unknown, Chernoff bounds will not be applicable. As noted by Hay et al. [9], the more complex algorithm, the more difficult it is to analyze the algorithm theoretically. Consequently, some algorithms may be easier to investigate empirically instead of analytically.

In addition, previous research [9, 10] shows that the accuracy of a differentially private algorithm may be greatly influenced by the input data. Consequently, input data should also be taken into account when modeling error. So far, the current literature seems to model error from the algorithm without taking the input data into consideration. For example, Kasiviswanathan et al. [11] and Vadhan [27] use Chernoff bounds, but they do not include input data in their error model.

From the other end of the perspective, several papers including [1–3, 8, 12, 13, 29] investigate error empirically. Still, input values to the experiments are chosen seemingly arbitrarily. For example, Gao and Ma [8] use {0.005, 0.008, 0.012, 0.015, 0.02} as input values for a threshold variable, and {20, 40, 60, 80, 100} as input for query range size. While these values may be representative for their given domain, this approach requires the authors to rationalize both the chosen ranges and the amount of values used. Furthermore, if a variable is varied in isolation, it is not possible to capture interactions between variables. For example, in [3], the authors vary the number of dimensions, while setting cardinality and ε to fixed values. As such the trend for error when varying the number of dimensions is just captured at a fixed setting.

Hence, we identify three existing problems: 1) the relationship between error and an algorithm's input may be unknown, 2) data oblivious error may result in incorrect error predictions, and 3) choosing representative values for empirical experiments is difficult. To mitigate these problems we propose a novel application of *factor experiments* [18, 20, 24], a statistical approach, to the domain of differential privacy. Here, we show how empirical error measurements can be used to construct an error prediction model using (multiple) linear regression. As such, we are able to model the relationship between all input variables, including data, and error. Accordingly, for the example with ε and population as variables, the prediction model would be in the following format:

$$\begin{aligned} y =&\gamma_0 + \gamma_{threshold} \times \mathsf{threshold} + \gamma_{range} \times \mathsf{range} \\ &+ \gamma_{threshold:range} \times \mathsf{threshold : range} \end{aligned} \tag{1}$$

where y is the predicted error for a specific setting, γ_0 is the intercept, threshold and range are *coded value* representations of the factors, and threshold:range is the possible interaction between factors. Hence, the prediction model is able to predict the error for any value (within the model's span) of threshold and range.

More importantly, factor experiments provide a systematic way to choose the experiment settings where the most information can be extracted. Consequently, our methodology tackles all of the three identified problems by 1) modeling the relationship between variables and error, 2) involving all input variables in model creation, and 3) minimizing the samples required, allowing for efficient experiments.

We expect our methodology to be valid for any differentially private algorithm: factor experiments allow both numerical and categorical variables, and the analyst may choose any suitable error metric for their domain. To put our methodology into context, we will conduct a case study. In our case study, we run a poll where the algorithm traverses a tree structure before delivering a differentially private reply. Now, we will argue that our use case is particularly interesting in the context of our methodology. First, we have noticed that it is difficult to model the error correctly due to allowing for arbitrarily complex tree structures, where we identify six variables that need to be varied in experiments. Next, it is also difficult to argue for what constitutes a 'good' experiment setting in this case. As such, we believe the many variables' effect on error in our particular use case is difficult to investigate using methods from the current literature. Accordingly, we use Randori [14] as a use case where we create a prediction model for error. Randori is a set of tools for gathering poll data under *local differential privacy* [28]. So far, Randori can predict error analytically through Chernoff bounds, but this error is not data aware. In this paper, we extend Randori by adding a simulation tool where users can generate synthetic poll data and empirically evaluate error.

To summarize, prediction models created using our methodology will be able to answer the following questions:

- What is each variable's impact/effect on error?
- Are there any relationships/interactions between variables?

Hence, our contribution is a method for constructing accuracy/error prediction models.

2 BACKGROUND

In this paper, we join two well-known areas: differential privacy and *statistical design of experiments* (DOE) [16]. To provide the reader the necessary background, we describe the accuracy/privacy trade-off in differential privacy. As we expect our readers to mainly come from the area of differential privacy, we also introduce terminology used in DOE.

2.1 Differential Privacy

Differential privacy [5, 6] is a statistical notion of privacy that quantifies the privacy loss. Since differential privacy is a definition and not an implementation, differential privacy can be achieved in different ways, but must always satisfy Definition 2. To define differential privacy, we must first define neighboring data sets (Definition 1).

Definition 1 (Neighboring Data Sets). *Two data sets, D and D', are neighboring if and only if they differ on at most one element d. That is, D' can be constructed from D by adding or removing one single element d:*

$$D' = D \pm d$$

Definition 2 (ε-Differential Privacy). *A randomized algorithm f is ε-differentially private if for all neighboring data sets D, D' and for all sets of outputs $\mathcal{S}$*

$$Pr[f(D) \in \mathcal{S}] \leq exp(\varepsilon) \times Pr[f(D') \in \mathcal{S}]$$

where the probability is taken over the randomness of the algorithm f.

Although differential privacy gives strong mathematical privacy guarantees, implementations introduce some kind of error, relative to an exact but non-private algorithm, to achieve said privacy. The accuracy of a differentially private algorithm can be investigated through analytical accuracy bounds, such as Chernoff bounds. These analytical accuracy bounds are often expressed in general terms, i.e. they do not define error for a specific algorithm, such as the Chernoff bound given by Kasiviswanathan et al. [11] in Definition 3.

Definition 3 ((α, β)-usefulness). *Let X be a random variable representing the error of the output of a differentially private algorithm f', n is the population size and $\alpha, \beta \in (0, \frac{1}{2})$, where $\beta = 2e^{-2\alpha^2 n}$. Then with probability 1-β, the error X is bounded by at most error α:*

$$Pr[X \leq \alpha] \geq 1 - \beta$$

We say that f' is (α, β)-useful [30].

Note that this formula in particular does not define how to express error. That is, error must be defined on a per-algorithm basis. For example, Kasiviswanathan et al. [11] use misclassification error as their error metric. Still, the resulting accuracy bounds cover the entire *possible* range of error the algorithm can achieve. That is, such theoretical accuracy bounds focus on the worst case error [9]. In other words, the bounds do not describe how error is distributed within the bound. For example, high errors may have very low probability, but an analyst may still condemn the algorithm because the accuracy bounds are not tight enough. Consequently, predicting error using analytical methods can be overly pessimistic.

Furthermore, it can be difficult to properly model the error in order to construct a Chernoff bound. The data dependence of an algorithm's error is particularly important to capture. As Hay et al. [9] point out, a number of differentially private algorithms are indeed data dependent. Hence, data can have an impact on error, but the current literature offers no guidelines on modeling error correctly.

2.2 Designed Experiments

In this paper, we will empirically measure the error of a differentially private algorithm. As a consequence, we need to plan and conduct experiments. More specifically, we will conduct *factor experiments* [18, 24], which is a more efficient way of conducting experiments than changing *one factor at a time* (OFAT) [7]. Here, a factor is the same as a variable, and we will use these terms interchangeably.

With factor experiments, we are able to change several factors simultaneously, allowing us to run fewer experiments in total. Essentially, factor experiments is a way of designing experiments such that we can maximize what is learned given a fixed number of measurements [18]. For example, conducting an experiment with two different factors that each can take on 100 different values would require 10 000 measurements with the OFAT approach. Using these same factors but instead running *two-level* factor experiments, we only need to measure the *response* at each edge of the space. That is, only measurements from the black dots in Figure 1 are required for factor experiments, whereas the response from each coordinate in the space is required using OFAT.

Hence, two-level factor experiments with two factors ($k = 2$) require only $2^k = 2^2 = 4$ measurements. In summary, with two-level factor experiments, 2^k measurements are needed for an experiment

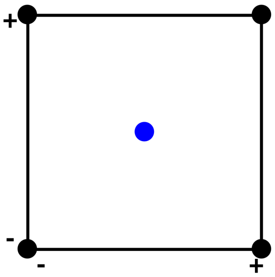

Figure 1: The space covered by a factor experiment with two factors. Black dots represents the factors at high/low respectively, and the blue dot is the baseline.

with k factors. Naturally, factor experiments are much more efficient than OFAT.

When running factor experiments, *coded values* are used to denote *actual values*. For example, two-level experiments usually have a low ('-' or '-1') and a high ('+' or '+1') coded value which are related to the actual values as follows:

$$v_{coded} = \frac{v - a}{b}, \tag{2}$$

$$\text{where } a = \frac{v_{high} + v_{low}}{2}, \tag{3}$$

$$b = \frac{v_{high} - v_{low}}{2} \tag{4}$$

So, in a real use case, with the high value (+1) 1000, and the low value (-1) 100, the actual value 500 is represented by the coded value $-\frac{5}{45}$.

Another point of interest in factor experiments is the *baseline*. The baseline is the center point (the blue dot in Figure 1) of the entire space that we cover. Consequently, the baseline always has the coded value 0 for each factor.

Using the 2^k responses from the factor experiments, it is possible to construct a prediction model. In this paper, we will construct a linear prediction model using (multiple) linear regression. Given two factors A and B, the linear model can be written as follows:

$$y = \gamma_0 + \gamma_1 A + \gamma_2 B + \gamma_{12} AB + \text{experimental error} \tag{5}$$

Where the constant γ_0 is the response at the baseline, and AB is included to capture the possible *interaction* between factor A and B.

Since the prediction model is linear, we will later show how to confirm these assumptions and validate the fit of the model. We also note that in case of non-linear systems, one can instead use three-level factorial designs [22], which are less efficient but are able to capture curvature.

3 METHODOLOGY

We propose a methodology consisting of four stages:

(1) Experiment design
(2) Data collection/generation
(3) Model creation
(4) Model validation

After going through all the stages, the prediction model is ready to be used.

3.1 Experiment Design

We propose using two-level factor experiments. This allows linear prediction models to be created. Note that it is important to not choose maximum or minimum values for the levels, as such values likely will be too extreme and not produce a valid model [4]. Instead, choose values that are feasible within the domain. Accordingly, the prediction model will be valid within the space the two levels span, but will not be able to make predictions for values outside. This step is necessary, as extreme values will likely break the assumptions about linearity that allow us to create a linear prediction model.

Next, the k factors involved needs to be identified. This can be done in different ways. The authors note that in software systems, this process is much more straightforward than in for example physical systems, since all possible factors are represented in code. As such, it should be possible to extract all factors from the code directly.

In cases where there are many factors, it might be a good idea to run *screening designs* first, using *fractional designs* [21] experiments to reduce the number of measurements needed. Basically, a fractional design only includes some of the 2^k points, but are chosen in a systematic way. With screening designs, it is possible to determine if there are factors that can be ignored without running the full 2^k experiments.

Our use case: In Randori, data is gathered in poll format. A poll consists of a number of questions and a fixed set of answer alternatives. We represent these questions as trees where a node is either an answer alternative or a question. Furthermore, we also allow follow-up questions in our poll. As such, some answer alternatives have question nodes as children.

Answers to the poll are then gathered using *randomized response* [28]. In randomized response, a respondent will answer truthfully with some probability, Pr[truth], and will otherwise choose a random answer according to a known distribution. In Randori, the known distribution is represented through weights attached to each answer alternative.

From our use case, we identify six factors to include in our experiment design. Here, Pr[truth] and relative alternative weight are due to randomized response. Tree depth and Number of alternatives are due to the poll's tree structure. Next, to make our model data aware, we include both the Population and the Number of answers which corresponds to the number of respondents that choose the answer alternative that we target in our measurements. We illustrate all of our identified factors in Figure 2. When we measure the error, we will choose one of the alternatives as our target, for example $A1_{Q1}$.

In Table 1 we show all our factors and define the levels for each factor.

Now, it makes sense to explain why we have not included ε among our factors. In our case, one thing we want to investigate is the impact of the poll structure on the error. However, there is not a one-to-one mapping between ε and poll structure. That is, while ε can be calculated from the structure of the poll, different structures can result in the same value of ε. As such, only varying ε would not allow us to deduce a unique poll structure.

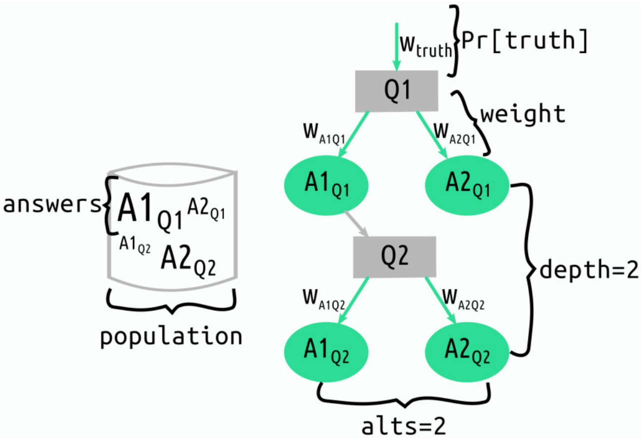

Figure 2: The factors used as input to Randori, including both data (to the left) and algorithm parameters (to the right). Here, question nodes are gray and answer alternatives are green.

Factor	+	-
Pr[truth]	High	Low
Tree depth	Deep	Shallow
Number of alternatives	Many	Few
Relative alternative weight	High	Low
Population	Many	Few
Number of answers	Many	Few

Table 1: Factors, and their respective levels

3.2 Data Collection/Generation

Data can either be collected from real experiments or generated synthetically. That is, responses from any differentially private algorithm can be used. Note that synthetic data does not make the model less valid: the prediction model will be valid for the entire space covered by the factors. In fact, if the algorithm can be simulated we recommend doing so, as this also eliminates the need to gather potentially sensitive data. Basically, the finesse of factor experiments is that we do not look to sample specific representative settings, but rather we want to be able to cover all values within a known space.

Since results from differentially private algorithms are probabilistic, it is also important to decide whether to measure an average error, or just one measurement per experiment setting. In this step, it is also important to decide which metric to use for error comparison.

Next, create a table for all the possible combinations of the k factors for a total of 2^k combinations. In physical systems, it is customary to produce the measurements in random order to avoid systematic errors.

Our use case: We construct a tool where we can generate synthetic data and measure the empirical error introduced by randomized response. This tool simulates respondents answering a given poll on Randori's format. We call this tool the Simulation Environment.

We decide to run each setting 30 times, i.e. $n = 30$, to measure the average error. We also decide to use *mean average percentage error* (MAPE) as our error metric:

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{x_t - x'_t}{x_t}\right| \times 100 \tag{6}$$

Here, we will calculate the MAPE for one target answer alternative. As such, we measure the distance between the actual percentage (x) of respondents that chose the target alternative, and the estimated percentage (x') calculated from the randomized responses.

3.3 Model Creation

From the measured error, it is now possible to create the prediction model. The prediction model is calculated using (multiple) linear regression. To create the prediction model, we suggest using the programming language R. In R, pass the data to the `lm` function and R will output a model. This model will include the effect of each variable and all present interactions between variables.

3.4 Model Validation

To test the fit of the model, we first check that the assumptions about linearity hold. Next, the predictions made by the model also need to be investigated. That is, more measurements need to be gathered and compared to the model's predictions for the same setting.

If the model has a good fit, the *residuals* will be small. We use the following formula to calculate the residual r_i when comparing a prediction y_i to a sample measurement s_i for some coordinate i:

$$r_i = y_i - s_i \tag{7}$$

A numerical measurement of the model's fit is the (multiple) R^2, the coefficient of determination. A high value of R^2 is necessary but not sufficient for concluding that the fit is good [17]. Next, compare the R^2 value to the adjusted R^2 (calculated as follows: $R^2_{adj} = 1 - (1 - R^2)\frac{N-1}{N-p-1}$, where N is the sample size and p is the number of predictors). The value of R^2 and the adjusted R^2 should be close. Otherwise, a difference indicates that there are terms in the model that are not significant [23]. Consequently, if R^2 and adjusted R^2 differ much, insignificant terms can be removed from the model. In this step, the programming language R can help with providing suggestions for which effects are significant.

Next, we recommend using visual methods to further validate the model due to NIST's recommendation [19]. These visual methods allow conclusions to be drawn that cannot be drawn from merely observing R^2.

We suggest the following three visual methods:

(1) Histogram
(2) Residual vs. fitted plot
(3) Q-Q normal plot

First, use a histogram to test the residuals for normality. Here, the residuals are expected to have the shape of a normal distribution, and to be centered around 0.

Next, for the residual vs. fitted plot, values should be randomly scattered around 0 on the y-axis [19]. We also expect the *locally weighted scatterplot smoothing* (LOWESS) [15] curve to be flat, since this shows that a linear model is reasonable.

Last, using the Q-Q normal plot shows if the residuals come from a common distribution as the prediction model. If the data sets come from common distributions, the points should be close to the plotted line.

Strategy if the model does not fit: To get quick feedback about the model's fit, pick the three points in Figure 3. Next, calculate the residuals for these points.

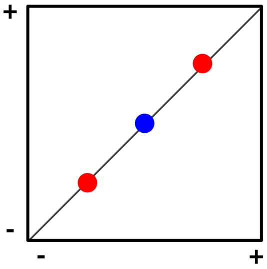

Figure 3: The center point, i.e. the baseline represented by the blue dot, and the red dots at (-0.5, -0.5) and (0.5,0.5) respectively

In cases where the residuals are high, re-use the samples from Figure 3 and add the remaining samples needed to create a new, smaller space. That is, systematically zoom in and target a smaller space to make the predictions on. We illustrate this new smaller space in 2D to be able to show a geometric explanation in Figure 4.

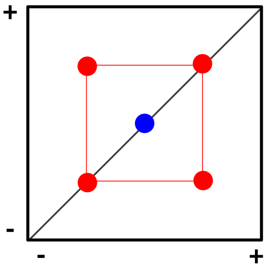

Figure 4: Adding the points (0.5,-0.5) and (-0.5,0.5) allows us to zoom in and find a new target space within the red lines

4 RESULTS

Next, we will apply our methodology to our use case where we estimate error for poll data. Here, we present the tool we used to generate data (the Simulation Environment) and then we show how we iteratively apply the methodology to reach an adequate prediction model.

4.1 Simulation Environment

We have built a simulation environment using a Jupyter notebook [25] that takes input on a portable JSON format. The Simulation Environment is an additional tool to the Randori [1] (Figure 5) set of open source tools.

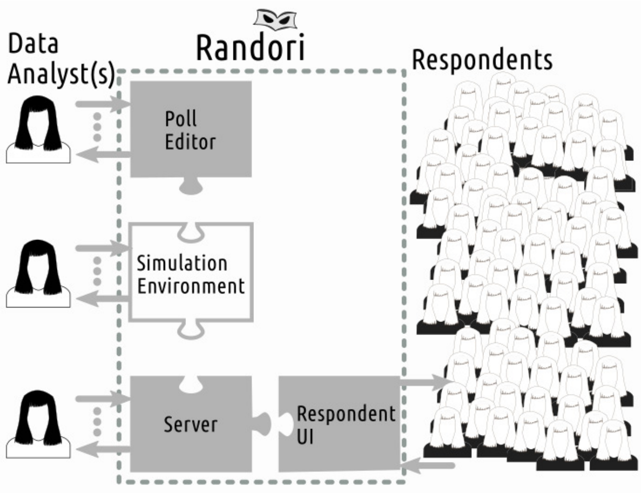

Figure 5: The Simulation Environment (white puzzle piece) in relation to existing Randori tools

Here, a user can construct an arbitrarily complex poll using Randori's Poll Editor. Next, the poll can be imported into the Simulation Environment where the user can tweak all the input variables. In addition, the Simulation Environment is able to simulate the respondents' answers either based on probability distributions or a deterministic distribution, although we only use deterministic distributions in this paper.

4.2 Experiments

We run a factor experiment with $k = 6$, and calculate the error as MAPE. We run each experiment $n = 30$ times.

Using the actual values in Table 2 we produce the measurements in Table 6 (in Appendix due to length).

Factor	Baseline	+1	-1
Pr[truth]	50%	90%	10%
Tree depth	3	5	1
Number of alternatives	6	10	2
Relative alternative weight	50%	90%	10%
Population	50500	100 000	1000
Number of answers	50%	90%	10%

Table 2: Factors and the actual values used for corresponding coded values. In the case of weight and pop the percentage is used for the target alternative, and the remainder is uniformly distributed among siblings.

[1] https://github.com/niteo/randori

We enter our data in R and create a prediction model using the `lm` function. Calculating the residual for the baseline, we get a significant error of 384.6646. We pick two additional settings and measure them (Table 3) to convince ourselves that the model is indeed a bad fit.

Setting	y_i	s_i	r_i
(0, 0, 0, 0, 0, 0)	418.7087	34.04411	384.6646
(0.5, 0.5, 0.5, 0.5, 0.5, 0.5)	124.8765	14.41732	110.4592
(-0.5, -0.5, -0.5, -0.5, -0.5, -0.5)	731.8813	38.23649	693.6448

Table 3: Residuals calculated using the prediction model for the first experiment

As a result, we move on to sample the 2^6 points that covers half the previous space i.e using the settings from Table 4. The measured MAPE is in Table 7 (in Appendix due to length). We then use these measurements to construct a new prediction model over the smaller space.

Factor	Baseline	+0.5	-0.5
Pr[truth]	50%	70%	30%
Tree depth	3	4	2
Number of alternatives	6	8	4
Relative alternative weight	50%	70%	30%
Population	50500	75750	25250
Number of answers	50%	70%	30%

Table 4: Factors and the values used for calculating residuals

From entering our measured values into R's `lm` function, we get a model with 64 coefficients. Using the model, we notice that the prediction for the baseline has improved significantly. The updated prediction is 32.89371, which gives us a residual of $34.04411 - 32.89371 = 1.1504$. Hence, we move on to validate our model.

5 ANALYSIS

Next, we move on to validate our model according to our methodology. After validating the model, we will interpret the model.

5.1 Evaluating the Model

In order to validate the model, we need to investigate the behavior of the residuals. Hence, we need more measurements. We have decided to pick settings to sample from two sets:

(1) The corners (2^6 points) of the middle of the model (like in Figure 4) and the center point
(2) Any coordinate in the space

We randomly pick 20 points (except that we always include the center point in the first set) from each of the two approaches, giving us a total of 40 samples to calculate residuals from. Be aware that you may also need to adjust values in certain cases. In our case, we need to take into account that some of our factors are discrete. For example depth is a discrete value and our corner values 0.25 and -0.25 would correspond to a depth of 3.5 and 2.5 respectively. Consequently, we chose to fix depth to 3. The points and their corresponding MAPE is shown in Table 5.

	truth	depth	alts	weight	pop	answers	MAPE
0	0	0	0	0	0	0	34.04411
1	0.25	0.00	0.25	0.25	0.25	0.25	20.17603
2	-0.25	0.00	0.25	-0.25	0.25	-0.25	48.18286
3	0.25	0.00	-0.25	-0.25	0.25	-0.25	31.06755
4	-0.25	0.00	0.25	-0.25	-0.25	0.25	50.33476
5	0.25	0.00	-0.25	0.25	0.25	0.25	19.59611
6	-0.25	0.00	0.25	0.25	0.25	0.25	27.66037
7	-0.25	0.00	-0.25	-0.25	0.25	-0.25	46.24753
8	-0.25	0.00	-0.25	0.25	0.25	0.25	26.60268
9	0.25	0.00	-0.25	0.25	0.25	-0.25	17.30670
10	-0.25	0.00	0.25	0.25	-0.25	-0.25	25.07704
11	-0.25	0.00	-0.25	-0.25	0.25	-0.25	46.36067
12	-0.25	0.00	-0.25	-0.25	0.25	-0.25	46.18749
13	0.25	0.00	-0.25	0.25	-0.25	0.25	19.71108
14	0.25	0.00	-0.25	-0.25	-0.25	0.25	33.26383
15	-0.25	0.00	0.25	-0.25	-0.25	-0.25	48.09976
16	-0.25	0.00	0.25	0.25	-0.25	0.25	27.58968
17	-0.25	0.00	-0.25	0.25	0.25	-0.25	22.55290
18	-0.25	0.00	0.25	0.25	0.25	-0.25	24.97823
19	0.25	0.00	-0.25	0.25	0.25	0.25	19.61443
20	-0.50	-0.50	-0.50	0.03	-0.46	0.28	8.42964
21	0.16	0.25	0.00	0.32	-0.25	0.38	28.34642
22	-0.06	-0.25	-0.50	0.03	-0.31	-0.32	8.82148
23	-0.50	0.25	-0.25	0.03	0.03	-0.29	53.20864
24	0.21	0.50	0.00	0.12	-0.17	0.34	36.71494
25	0.31	0.50	0.25	0.34	-0.02	0.39	29.04886
26	-0.49	0.25	0.25	-0.22	-0.12	0.07	63.40224
27	-0.27	-0.50	0.00	0.35	0.29	0.34	65.43967
28	0.39	0.25	0.50	0.21	-0.03	0.38	25.73380
29	0.39	-0.25	0.00	0.30	0.13	0.28	3.46581
30	-0.45	0.50	0.50	0.06	-0.04	-0.21	59.91642
31	-0.00	0.50	-0.25	-0.36	0.05	-0.02	47.62934
32	-0.20	-0.25	-0.50	-0.03	0.16	0.42	21.80034
33	-0.14	0.25	0.50	-0.40	0.11	0.46	53.57877
34	0.11	0.00	-0.25	-0.48	-0.35	-0.21	39.38831
35	0.14	0.00	0.00	-0.37	0.15	0.02	38.41253
36	-0.09	-0.50	-0.50	-0.41	-0.47	-0.39	5.75857
37	-0.19	0.50	0.25	-0.08	0.44	-0.19	52.70103
38	0.42	-0.50	-0.25	-0.19	0.00	-0.01	2.18997
39	-0.47	0.50	-0.25	0.33	-0.33	0.35	51.42151

Table 5: The sampled points used and their measured MAPE

First, we check the value of our R^2. For our model, the R^2 is 0.8419. However, we notice that the adjusted R^2 is significantly lower, 0.5929. Seeing as we have 64 coefficients, it seems reasonable to simplify our model to avoid overfitting. We update our model in R to only involve the effects that R marks as significant. To do this, we enter the suggested effects in R, which in our case are: `lm(formula = MAPE ~ truth + alts + weight + truth*depth+depth*weight + truth*depth*weight + depth*weight*answers )`. Now, we end up with a R^2 of 0.7846, and an adjusted R^2 of 0.7562. These values are still high, and since they are now significantly closer, we move on to validate the model visually.

Next, we plot the residuals as a histogram in Figure 6. From the histogram, we see that our residuals are indeed centered around 0. The histogram indicates a normal distribution. Hence, we move on to the next test.

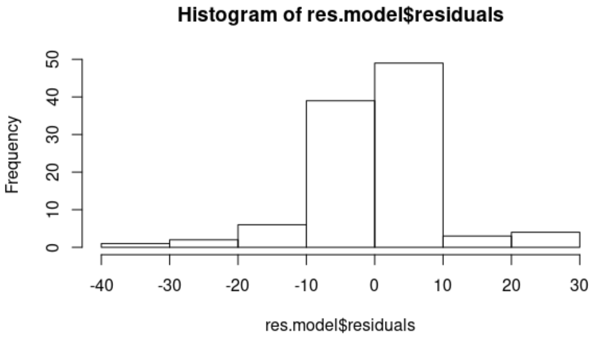

Figure 6: A histogram of the residuals

Now, we want to investigate the relationship between fitted values (measurements) and the model's prediction. Then, we plot fitted values vs. predictions in Figure 7. We observe that the residuals appear to not have a specific shape around the y-axis. We also conclude that the LOWESS fit curve appears to be almost flat.

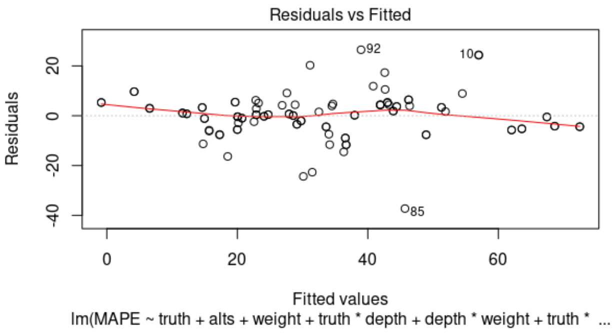

Figure 7: Residuals represented as circles, fitted values as the dotted line. The red line represents the LOWESS fit of the residuals vs. fitted values.

Finally, we investigate the normal Q-Q plot (Figure 8). We see that most points follow the plotted line, indicating that our predictions come from the same distribution as the measured values. Hence, we conclude that our prediction model is valid for our use case.

5.2 Interpreting the Model

The model is now ready to be used. That is, any value within each factor's range [high,low] can be plugged in to produce an error prediction. It is also possible to set $y \le c$, with c being our maximum tolerable error, and then find which settings satisfy the inequality. Our final error prediction model is as follows:

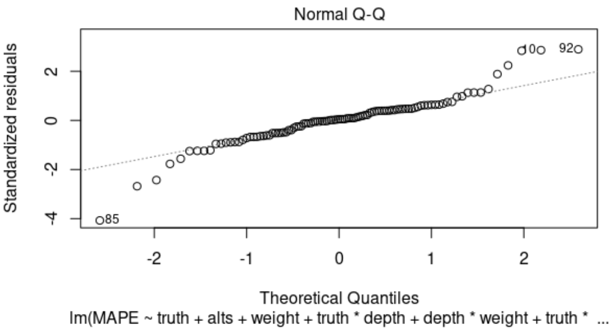

Figure 8: The normal quantile-quantile plot

$$\begin{aligned}
y = {} & 32.501266 - 29.023493 \times \text{truth} \ + 5.037411 \times \text{alts} \\
& - 16.562410 \times \text{weight} \ + 1.449934 \times \text{depth} \\
& + 1.856916 \times \text{answers} \ + 10.044302 \times \text{truth} : \text{depth} \\
& - 28.397984 \times \text{weight} : \text{depth} \\
& + 4.175231 \times \text{truth} : \text{weight} \\
& + 8.535667 \times \text{depth} : \text{answers} \\
& - 8.402531 \times \text{weight} : \text{answers} \\
& + 51.134829 \times \text{truth} : \text{weight} : \text{depth} \\
& + 25.945740 \times \text{weight} : \text{depth} : \text{answers} \qquad (8)
\end{aligned}$$

We note that the simplification step has allowed us to completely eliminate pop from our factors. As such, we draw the conclusion that the population size itself does not have a significant impact on error.

To get an overview of our model, we use a Pareto plot [26] (Figure 9) which allows us to visually compare all effects at once. Here, effects are ordered by magnitude.

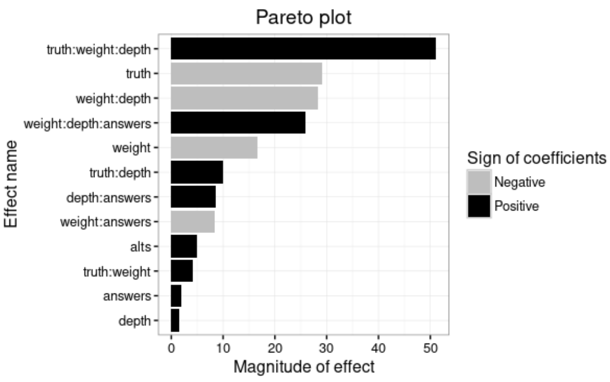

Figure 9: The Pareto plot of the simplified model

From the plot, it is clear that truth:weight:depth affects error the most. Maybe most notably, truth:weight:depth increases error whereas its components truth and weight:depth both decrease error. From examining the Pareto plot, it seems that truth:weight is the interaction that causes the increase in error.

As expected, truth has a negative impact on error. That is, a high value of truth will reduce error. More surprisingly, truth is involved in several interactions which all increase error.

It may be tempting to completely ignore answers and depth as these two factors have the lowest magnitude of effect. However, ignoring these factors is dangerous: they are both involved in interactions that have significantly higher magnitude.

The factor alts is the only one that does not have interactions. It may seem counter-intuitive that having more siblings have such a small impact on error. Still, the magnitude of this effect may very well be due to our choice to input polls where we uniformly distribute the remaining weight among the siblings.

Hence, we can give RANDORI's users the following advice: use the model to find local minima or ranges. The model can also be used to find minima and ranges while accounting for known constraints such as for example Pr[truth] ≤ 0.5. When working in RANDORI's POLL EDITOR it is important to beware of the main effect truth:weight:depth and its building blocks. As weight primarily is involved in decreasing error, we recommend increasing weight before tweaking the other factors.

6 DISCUSSION, LIMITATIONS AND FUTURE WORK

A limitation in our work is that the prediction models we create are linear. As such, prediction can be off in cases where the error is in fact non-linear. Still, factor experiments can nevertheless be used to make predictions for non-linear systems. To facilitate for non-linear systems the factor levels have to be chosen differently: i.e. we would need 3 levels [22] instead of 2. Hence, our approach can be adapted to create non-linear models by running more experiments.

Additionally, we know that error should also depend on the, non-linear, term $exp(\varepsilon)$ from the definition of differential privacy. Still, it is not clear how the term $exp(\varepsilon)$ and other, algorithm specific, factors compare in order of magnitude. As such, more research is needed to see if ε can be modeled in a suitable way, or if perhaps ε needs to be transformed to be linear ($ln(exp(\varepsilon))$. Nevertheless, factor experiments still provide a systematic and efficient way to explore the impact of different variables on error. That is, factor experiments may still be used to explore the other factors' impact on error. Hence, while it may not always be possible to extract an accurate prediction model, factor experiments are still useful when determining which data points should be used as input to test the accuracy of a differentially private algorithm.

Furthermore, factor experiments provide a possible way to systematically predict error for *all* representative input data sets for a differentially private algorithm. That is, instead of using real data sets to predict error, factor experiments statistically emulate all possible data sets bounded by the experiment's levels (the high/low values for each variable in our case). Hence, using factor experiments to create prediction models can be more robust statistically than making predictions based on one real data set.

Whether the model is correct or not will be identified when testing the model according to our methodology. If the model is incorrect it can be due to error being non-linear, but it can also be due to not including all relevant factors. As such, an incorrect model requires further investigation.

Accordingly, correctly identifying relevant factors is crucial to building a correct model. Still, there exists no recognized way of correctly and efficiently identifying all factors. As mentioned in Section 3.1, it is nonetheless possible to try if a factor is relevant using *screening designs* before running a full factorial experiment. From our use case, it is nonetheless clear that some candidate factors rule themselves out by simply being impossible to implement. For example, we considered having the factor *number of parent siblings* together with depth, which results in the impossible combination of having no parents (depth=0) and also having parent siblings. Hence, we believe looking for possible contradictions among factors is important when designing the experiments.

In order to not create contradicting factors, we have also decided to only model the weight for the target alternative. That is, we set the weight for the target alternative (or the target's parent), and uniformly divide the remainder among the siblings. For example, when a target has weight 70% and three siblings, each sibling gets $\frac{100-70}{3}\%$ each. As such, we have not investigated settings where the siblings have non-uniform weight distributions.

One decision that may seem controversial is that we do not include ε as one of the factors in our model. While we do not tweak ε directly, we do in fact adjust ε by changing the structure of the poll. The reason we have chosen to indirectly tweak ε as to tweaking it directly is that one single value of ε corresponds to multiple poll structures, whereas one poll structure corresponds to exactly one value of ε. Hence, while it may seem unintuitive at first, indirectly tweaking ε makes more sense than tweaking it directly in our case.

Somewhat surprising is that population was eliminated from our prediction model in the simplification step. We argue that the elimination of population is because answers is related to pop (the probability of choosing some alternative Ai_{Qj} is $\Pr[Ai_{Qj}]$ = pop*answers), and population therefore becomes redundant. It is also possible that the choice of error measurement, MAPE in our case, contributes to making population irrelevant since it is a relative measurement of error as opposed to an absolute measurement.

Finally, we note that in this paper we have measured the error of leaf nodes in a tree. Still, with the known relationships between answers, it appears to be possible to further post-process and add accuracy to parent answers. We believe including the number of children as a factor would be an interesting path to explore next in order to better understand the effect of this post-processing. Put differently, the challenge here is properly modeling the factors without creating contradictions between factors.

7 RELATED WORK

As mentioned in Section 1, evaluating error empirically is not a new topic within differential privacy. However, creating prediction models from empirical data appears to be a novel approach.

The work closest to ours is DPBENCH [9], which is an error evaluation framework for differentially private algorithms. In DPBENCH, the authors propose a set of *evaluation principles*, including guidelines for creating diverse input for algorithms. Hence, DPBENCH has a strong focus on understanding the data-dependence of an algorithm's error. Still, DPBench does not produce an error prediction model like we do, nor does it minimize the number of experiments needed to conduct.

We also note that DPComp [10] is the closest work to our Simulation Environment. DPComp allows users to compare how the accuracy of a differentially private algorithm is affected by varying input data. Our work is similar in the sense that our Simulation Environment also is intended to be used to evaluate accuracy/privacy trade-offs. Our Simulation Environment is also inspired by DPBench's evaluation principles and consequently allows data following different distributions to be entered and evaluated. However, our simulation environment is less general than DPComp, since our solution uses one fixed algorithm.

8 CONCLUSION

We have presented a methodology for empirically estimating error in differentially private algorithms which 1) models the relationships between input parameters, 2) is data aware, and 3) minimizes the measurements required as input. Hence, prediction models created using our methodology allow for expressive, data aware, error prediction. Moreover, we conducted a case study where we apply our methodology to a setting where error is measured from poll structures. To support our use case, we have added a simulation tool to the Randori open source tool suite, adding the functionality of generating synthetic data and evaluating error empirically.

From our case study, we were able to create a prediction model for error using six factors. After evaluating and simplifying our model, we are able to answer the two questions from our introduction. First, there are 13 main effects on error. Next, there are seven interactions.

From evaluating the prediction model we found that our model has a good fit. As such, our novel application of factor experiments shows promising results as a methodology for error evaluation of differentially private algorithms.

Consequently, we have contributed with a novel application of a methodology that shows promise for error prediction of differentially private algorithms. In addition, we have also built a simulation environment that generates synthetic poll data and measures error through simulating randomized response.

One interesting path for future work is to investigate if, and how, the number of factors used in the model prediction affects the model's fit. Along a similar line of thought, it would also be interesting to attempt to create prediction models for well known differentially private algorithms and libraries. As such, we encourage the use of our methodology in order to construct error prediction models for other differentially private algorithms.

ACKNOWLEDGMENTS

This work was partly funded by the Swedish Foundation for Strategic Research (SSF) and the Swedish Research Council (VR).

REFERENCES

[1] Tarek Benkhelif, Françoise Fessant, Fabrice Clérot, and Guillaume Raschia. 2017. Co-Clustering for Differentially Private Synthetic Data Generation. In *Personal Analytics and Privacy. An Individual and Collective Perspective*, Riccardo Guidotti, Anna Monreale, Dino Pedreschi, and Serge Abiteboul (Eds.). Springer International Publishing, Cham, 36–47.

[2] Rui Chen, Yilin Shen, and Hongxia Jin. 2015. Private Analysis of Infinite Data Streams via Retroactive Grouping. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (Melbourne, Australia) *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 1061–1070. https://doi.org/10.1145/2806416.2806454

[3] Bolin Ding, Marianne Winslett, Jiawei Han, and Zhenhui Li. 2011. Differentially Private Data Cubes: Optimizing Noise Sources and Consistency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, New York, NY, USA, 217–228. https://doi.org/10.1145/1989323.1989347

[4] Kevin Dunn. 2021. Process Improvement Using Data. Release 0f428b.

[5] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Number 4052 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1–12.

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer, Berlin, Heidelberg, 265–284.

[7] Ronald A. Fisher. 1990. *Statistical Methods, Experimental Design, and Scientific Inference* (1st edition ed.). Oxford University Press, Oxford, England.

[8] Ruichao Gao and Xuebin Ma. 2018. Dynamic Data Histogram Publishing Based on Differential Privacy. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. IEEE, Melbourne, VIC, Australia, 737–743. https://doi.org/10.1109/BDCloud.2018.00111

[9] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, and Dan Zhang. 2016. Principled Evaluation of Differentially Private Algorithms Using DPBench. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 139–154. https://doi.org/10.1145/2882903.2882931

[10] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. 2016. Exploring Privacy-Accuracy Tradeoffs Using DPComp. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 2101–2104. https://doi.org/10.1145/2882903.2899387

[11] Shiva Kasiviswanathan, Homin Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What Can We Learn Privately? *SIAM J. Comput.* 40, 3 (Jan. 2011), 793–826. https://doi.org/10.1137/090756090

[12] Hui Li, Jiangtao Cui, Xue Meng, and Jianfeng Ma. 2019. IHP: Improving the Utility in Differential Private Histogram Publication. *Distributed and Parallel Databases* 37 (2019), 721–750.

[13] Haoran Li, Li Xiong, Xiaoqian Jiang, and Jinfei Liu. 2015. Differentially Private Histogram Publication for Dynamic Datasets: An Adaptive Sampling Approach. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (Melbourne, Australia) *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 1001–1010. https://doi.org/10.1145/2806416.2806441

[14] Boel Nelson. 2021. Randori: Local Differential Privacy for All. arXiv:2101.11502 [cs].

[15] NIST/SEMATECH. 2013. 4.1.4.4. LOESS (Aka LOWESS). https://www.itl.nist.gov/div898/handbook/pmd/section1/pmd144.htm. [Accessed: 2021-05-15].

[16] NIST/SEMATECH. 2013. 4.3.1. What Is Design of Experiments (DOE)? https://www.itl.nist.gov/div898/handbook/pmd/section3/pmd31.htm. [Accessed: 2021-03-02].

[17] NIST/SEMATECH. 2013. 4.4.4. How Can I Tell If a Model Fits My Data? https://www.itl.nist.gov/div898/handbook/pmd/section4/pmd44.htm. [Accessed: 2021-02-24].

[18] NIST/SEMATECH. 2013. 5.1.1. What Is Experimental Design? https://www.itl.nist.gov/div898/handbook//pri/section1/pri11.htm. [Accessed: 2021-02-17].

[19] NIST/SEMATECH. 2013. 5.2.4. Are the Model Residuals Well-Behaved? https://www.itl.nist.gov/div898/handbook/pri/section2/pri24.htm. [Accessed: 2021-02-17].

[20] NIST/SEMATECH. 2013. 5.3.3.3. Full Factorial Designs. https://www.itl.nist.gov/div898/handbook/pri/section3/pri333.htm. [Accessed: 2021-02-24].

[21] NIST/SEMATECH. 2013. 5.3.3.4. Fractional Factorial Designs. https://www.itl.nist.gov/div898/handbook/pri/section3/pri334.htm. [Accessed: 2021-02-24].

[22] NIST/SEMATECH. 2013. 5.3.3.9. Three-Level Full Factorial Designs. https://www.itl.nist.gov/div898/handbook//pri/section3/pri339.htm. [Accessed: 2021-05-15].

[23] NIST/SEMATECH. 2013. 5.4.4. How to Test and Revise DOE Models. https://www.itl.nist.gov/div898/handbook/pri/section4/pri44.htm. [Accessed: 2021-02-24].

[24] NIST/SEMATECH. 2013. NIST/SEMATECH e-Handbook of Statistical Methods. https://www.itl.nist.gov/div898/handbook/index.htm. [Accessed: 2021-02-17].

[25] Project Jupyter. 2021. Project Jupyter. https://www.jupyter.org. [Accessed: 2021-05-15].

[26] RDocumentation. 2021. Pareto.Chart Function - RDocumentation. https://www.rdocumentation.org/packages/qcc/versions/2.6/topics/pareto.chart. [Accessed: 2021-05-15].

[27] Salil Vadhan. 2017. The Complexity of Differential Privacy. In *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, Yehuda Lindell (Ed.). Springer International Publishing, Cham, 347–450. https://doi.org/10.1007/978-3-319-57048-8_7

[28] Stanley L. Warner. 1965. Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. *J. Amer. Statist. Assoc.* 60, 309 (March 1965), 63–69. https://doi.org/10.1080/01621459.1965.10480775

[29] Yonghui Xiao, James Gardner, and Li Xiong. 2012. DPCube: Releasing Differentially Private Data Cubes for Health Information. In *International Conference on Data Engineering (ICDE)*. IEEE, Arlington, VA, USA, 1305–1308. https://doi.org/10.1109/ICDE.2012.135

[30] Tianqing Zhu, Gang Li, Wanlei Zhou, and Philip S. Yu. 2017. *Differential Privacy and Applications*. Advances in Information Security, Vol. 69. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-62004-6

A EXPERIMENTS

Standard order	Pr[truth]	Tree depth	Number of alternatives	Alternative weight	Population	Number of answers	MAPE
N/A	0	0	0	0	0	0	34.04411
1	-	-	-	-	-	-	87.08667
2	+	-	-	-	-	-	3.49111
3	-	+	-	-	-	-	37.57905
4	+	+	-	-	-	-	4.90007
5	-	-	+	-	-	-	47.75
6	+	-	+	-	-	-	6.58
7	-	+	+	-	-	-	76.73124
8	+	+	+	-	-	-	8.56657
9	-	-	-	+	-	-	7365.33667
10	+	-	-	+	-	-	96.20333
11	-	+	-	+	-	-	1228.76234
12	+	+	-	+	-	-	19.77456
13	-	-	+	+	-	-	1456.40333
14	+	-	+	+	-	-	18.47
15	-	+	+	+	-	-	405.1528
16	+	+	+	+	-	-	3.74374
17	-	-	-	-	+	-	90.03673
18	+	-	-	-	+	-	1.21997
19	-	+	-	-	+	-	39.38121
20	+	+	-	-	+	-	4.38645
21	-	-	+	-	+	-	47.13567
22	+	-	+	-	+	-	7.02496
23	-	+	+	-	+	-	75.60747
24	+	+	+	-	+	-	8.34256
25	-	-	-	+	+	-	7362.4095
26	+	-	-	+	+	-	98.25777
27	-	+	-	+	+	-	1240.11986
28	+	+	-	+	+	-	19.7394
29	-	-	+	+	+	-	1466.18583
30	+	-	+	+	+	-	18.8858
31	-	+	+	+	+	-	403.33846
32	+	+	+	+	+	-	4.16551
33	-	-	-	-	-	+	61.83111
34	+	-	-	-	-	+	8.08626
35	-	+	-	-	-	+	88.29154
36	+	+	-	-	-	+	9.66657
37	-	-	+	-	-	+	63.75222
38	+	-	+	-	-	+	8.2323
39	-	+	+	-	-	+	89.69907
40	+	+	+	-	-	+	10.02583
41	-	-	-	+	-	+	811.41556
42	+	-	-	+	-	+	10.13037
43	-	+	-	+	-	+	310.01569
44	+	+	-	+	-	+	2.16437
45	-	-	+	+	-	+	738.71667
46	+	-	+	+	-	+	9.07111
47	-	+	+	+	-	+	300.02957
48	+	+	+	+	-	+	2.1328
49	-	-	-	-	+	+	61.99979
50	+	-	-	-	+	+	7.9004
51	-	+	-	-	+	+	88.42618
52	+	+	-	-	+	+	9.84616
53	-	-	+	-	+	+	63.75659
54	+	-	+	-	+	+	7.95395
55	-	+	+	-	+	+	89.82931
56	+	+	+	-	+	+	9.95786
57	-	-	-	+	+	+	810.22851
58	+	-	-	+	+	+	9.99809
59	-	+	-	+	+	+	310.55943
60	+	+	-	+	+	+	2.44021
61	-	-	+	+	+	+	737.21517
62	-	+	+	+	+	+	299.99379
63	+	-	+	+	+	+	9.01693
64	+	+	+	+	+	+	2.20558

Table 6: MAPE measurements for the experiment using -1 and +1 as coded value inputs

Standard order	Pr[truth]	Tree depth	Number of alternatives	Alternative weight	Population	Number of answers	MAPE
N/A	0	0	0	0	0	0	34.04411
1	-	-	-	-	-	-	38.23649
2	+	-	-	-	-	-	17.89185
3	-	+	-	-	-	-	58.33831

Standard order	Pr[truth]	Tree depth	Number of alternatives	Alternative weight	Population	Number of answers	MAPE
4	+	+	-	-	-	-	25.18673
5	-	-	+	-	-	-	48.15875
6	+	-	+	-	-	-	25.15229
7	-	+	+	-	-	-	64.44095
8	+	+	+	-	-	-	27.66351
9	-	-	-	+	-	-	81.467
10	+	-	-	+	-	-	13.00362
11	-	+	-	+	-	-	9.89232
12	+	+	-	+	-	-	9.41709
13	-	-	+	+	-	-	56.28555
14	+	-	+	+	-	-	9.56171
15	-	+	+	+	-	-	19.75423
16	+	+	+	+	-	-	12.79737
17	-	-	-	-	+	-	38.11988
18	+	-	-	-	+	-	17.97198
19	-	+	-	-	+	-	58.37657
20	+	+	-	-	+	-	25.14935
21	-	-	+	-	+	-	48.43102
22	+	-	+	-	+	-	25.08915
23	-	+	+	-	+	-	64.49147
24	+	+	+	-	+	-	27.73975
25	-	-	-	+	+	-	81.24882
26	+	-	-	+	+	-	13.02403
27	-	+	-	+	+	-	9.5234
28	+	+	-	+	+	-	9.65797
29	-	-	+	+	+	-	56.3261
30	+	-	+	+	+	-	9.79661
31	-	+	+	+	+	-	19.70136
32	+	+	+	+	+	-	12.57202
33	-	-	-	-	-	+	52.6255
34	+	-	-	-	-	+	23.3408
35	-	+	-	-	-	+	66.96285
36	+	+	-	-	-	+	28.56059
37	-	-	+	-	-	+	54.63909
38	+	-	+	-	-	+	28.61188
39	-	+	+	-	-	+	68.09695
40	+	+	+	-	-	+	29.17961
41	-	-	-	+	-	+	45.78992
42	+	-	-	+	-	+	4.45637
43	-	+	-	+	-	+	23.87327
44	+	+	-	+	-	+	13.78785
45	-	-	+	+	-	+	41.37552
46	+	-	+	+	-	+	13.85628
47	-	+	+	+	-	+	25.68611
48	+	+	+	+	-	+	14.47902
49	-	-	-	-	+	+	52.71001
50	+	-	-	-	+	+	23.2522
51	-	+	-	-	+	+	66.94767
52	+	+	-	-	+	+	28.70839
53	-	-	+	-	+	+	54.66564
54	+	-	+	-	+	+	28.71268
55	-	+	+	-	+	+	68.04705
56	+	+	+	-	+	+	29.16309
57	-	-	-	+	+	+	45.72794
58	+	-	-	+	+	+	4.47782
59	-	+	-	+	+	+	23.84796
60	+	+	-	+	+	+	13.90072
61	-	-	+	+	+	+	41.23229
62	-	+	+	+	+	+	25.70945
63	+	-	+	+	+	+	13.88817
64	+	+	+	+	+	+	14.41732

Table 7: MAPE measurements for the experiment using -0.5 and +0.5 as coded value inputs