Local Sensitivity for Differentially Private Simple Linear Regression using Least Trimmed Squares

BENJAMIN HERZBERG, Khoury College of Computer Sciences, Northeastern University
HELENA DWORAK, Khoury College of Computer Sciences, Northeastern University

In this paper, we introduce a novel function for calculating the local sensitivity of an data set Y on which linear regression is being performed. To do this, we execute Least Trimmed Squares on a particular data set and then, our novel function looks at how much Y would have to change to produce an specific Least Trimmed Squares solution. This local sensitivity function can be used as the score function for Report Noisy Max, which will then output a differentially private estimate of linear regression on Y. The code can be found here https://github.com/bhherzberg/DP_LTS

1 INTRODUCTION

1.1 Motivation

We consider a common situation of analyzing and compiling course feedback in a university setting. Professor Pry would like to compile data about his students so that he may reflect on the results for future semesters. Specifically, he wants to know what each student received in the course and what they think about his teaching ability. The students, however, would like to guarantee that the professor will not be able to determine their exact feedback. They are not comfortable sharing the raw data as the professor would be able to track their feedback based on the associated grade. Instead, they look into common differentially private algorithms that may privatize their feedback before returning the results to the professor. The students find that many of the common differentially private algorithms will not work on their class size of fewer than one hundred students. They realize they must consider other algorithms that will privatize their data even with the limited number of data points they have. Our paper presents a possible solution to this problem by privatizing an estimate for linear regression.

Beyond this scenario, there exist many small-sized data sets that must be privatized, but traditional differential privacy algorithms distort the data beyond a reasonable amount. Because of this, we are motivated to find a algorithm that will privatize the data while still maintaining reasonably-approximate relationships.

1.2 Simple Linear Regression

Simple Linear Regression is often the default practice for analyzing trends amongst data, especially when data sets are small. Researchers want to see if some feature in the data set correlates with another highly and will often use linear regression as an estimator for proportional relationships.
Simple Linear Regression, commonly referred to as least squares regression, is the process of finding a line, $y = \beta x + \alpha$, represented by the terms (α, β), that best fits the data set. More formally, it is the line that

Authors' addresses: Benjamin Herzberg Khoury College of Computer Sciences, Northeastern University, herzberg.b@northeastern.edu; Helena Dworak Khoury College of Computer Sciences, Northeastern University, dworak.h@northeastern.edu.

minimizes the squared value of the vertical distance of each point from the line.

$$(\alpha', \beta') = argmin_{(\alpha,\beta)} \sum_{i=1}^{n} (\beta x_i + \alpha - y_i)^2 \quad \text{where} \quad (x_i, y_i) \in Y, \quad \forall i \in \{1, 2, \ldots, |Y|\} \tag{1}$$

Above, the pair (x, y) represents a point in the data set Y. Linear regression performs an optimization over the set of all possible α and β values for each of the value pairs. Conveniently, there is a closed form solution to this optimization problem.

$$\beta' = \frac{(\sum_{i=1}^{n} x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{(\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x_i)^2} \qquad \alpha' = \sum_{i=1}^{n} y_i - \beta' \sum_{i=1}^{n} x_i \tag{2}$$

This solution has a low time complexity as the number of operations grows with $|Y|$, such that its runtime is $O(n)$, where $n = |Y|$. Because of these properties, it is widely used, however, simple linear regression does not satisfy any privacy constraints and information may be exposed about specific individuals.

1.3 Differential Privacy

As the motivating example described, there is need to be able to privatize these linear regressions, especially involving small, focused data sets. The formal definition of differential privacy is as follows.

Definition 1.1. Differential Privacy: An algorithm A is ϵ-Differentially Private if, for all neighboring data sets x and x' and all Events E in the output space of A, the following equation holds;

$$\mathcal{P}(A(x) \in E) \leq e^{\epsilon} \cdot \mathcal{P}(A(x') \in E)$$

We define neighboring data sets as those that differ in at most one point, *i.e.* changing the data set x to x' only requires modifying one data point. Additionally, we define all events, E, in the output space of A, to mean that E_i is any event in the set of all possible outputs from the algorithm A. Together, this definition means that the likelihood of getting some specific output when performing $A(x)$ is not more than a factor of e^{ϵ} away from the likelihood of getting that same output when performing $A(x')$. This definition is essential to keeping sensitive information from an adversary as it provides a guarantee that when we run the algorithm, our results will not be prone to reconstruction.

In this paper, we will use the Report Noisy Max framework to produce a differentially private estimator for linear regression. Many algorithms have been developed for differentially private least squares regression, such as the NoisyStats algorithm [1]. Because the NoisyStats algorithm is a proven differentially private estimator, we will use it to compare against our algorithm's result. Our algorithm will be structurally different from NoisyStats, however, as it uses Least Trimmed Squares and our novel sensitivity function in order to create a estimate that is less sensitive to outliers and noise in the data. We hope that with the reduced sensitivity, we can limit the error in our estimate when compared to NoisyStats.

1.3.1 *Sensitivity.*

Firstly, we must define the sensitivity as we will be using in our estimator.

Definition 1.2. [1] Global Sensitivity: For a query q on the data set $X^n \rightarrow \mathcal{R}^k$ the global sensitivity is

$$GS_q = max_{x \sim x'} ||q(x) - q(x')||_1$$

The global sensitivity is the maximum value by which a query could vary from any data set to a neighboring data set. One can create a deferentially private algorithm by adding noise proportional to

the global sensitivity over ϵ, GS/ϵ; this is problematic for linear regression as that the global sensitivity is infinite. Because of this, we turn to local sensitivity [1].

Definition 1.3. [1] Local Sensitivity: For a query q on the data set $X^n \rightarrow \mathcal{R}^k$ with respect to a data set x the local sensitivity is

$$LS_q = max_{x \sim x'} ||q(x) - q(x')||_1$$

Local sensitivity is the sensitivity of a specific data set. On its own, local sensitivity is not differentially private as it heavily correlates to the data set, but when combined more noise by considering neighboring data sets, we can hopefully add just enough noise to our estimate. This will then produce an estimate that is differentially private.

1.4 Least Trimmed Squares

Least Trimmed Squares (LTS) is a modified version of least squares regression. Instead of finding the line of best fit for all of the points, it attempts to find the line of best fit while eliminating a certain number of points from the regression, "trimming" them. LTS has some unique properties and is more resistant to outliers than least squares. Least squares regression can be easily modified by adding a large outlier, whereas LTS will resist the large deviation in its estimated slope. LTS manages this by trimming the additional outlier and instead includes a different point in the data set that does not affect the slope as much.
The breakdown point is a measure of robustness of an algorithm [2]: it measures how many data points can be moved before the estimate changes by an arbitrary amount. For instance, the breakdown point of regular least squares regression is $1/n$ as it only takes one point to move the estimate it produces to an arbitrary value. Least Trimmed Squares on the other-hand has a breakdown point of $(n-h)/n$ where h is its trimming factor. This property allows us to look at how output of Least Trimmed Squares changes as we modify the data set. The breakdown point of LTS is essential for our use of local sensitivity to create differential privacy, especially when using Report Noisy Max.

2 PROBLEM

The overarching problem we are trying to solve is to create an algorithm to produce a differentially private estimator of linear regression. To do so, we create an algorithm which takes a data set, Y; a trimming factor, h; and a privacy parameter, ϵ, as input and computes an differentially private estimate for linear regression on Y: $\hat{\beta}$.

input: data set Y, Privacy Parameter ϵ, Trimming factor h

output: Linear Regression Model $\hat{\beta}$

In order to be able to solve this problem, we need to look at a few sub-problems; First Least Trimmed Squares, Second Minimum Change Least Trimmed Square, and third Report Noisy Max. We can then combine these sub-problems and use them to solve our overarching goal.

2.1 Least Trimmed Squares

First, we formally define Least Trimmed Squares.

$$LTS(Y) = min_\beta \sum_{i=1}^{h} |r^2(\beta)|_i \qquad i \in \{1, 2, \ldots, n\} \tag{3}$$

In this equation, $|r^2(\beta)|_i$ is the absolute value of the squared residual for some point $i \in Y$ in relation to β. We also denote the points in an ordered manner from least to greatest $|r^2(\beta)|_1 \leq |r^2(\beta)|_2 \leq \cdots \leq |r^2(\beta)|_n$ where $n = |Y|$. Therefore, when we take the sum from 1 to h, we are able to find the h points with the smallest residual. [2]

The computation for LTS requires that we search over the set of all possible β to find the β with the smallest set of h residuals. There are $\binom{n}{h}$ options for β, and as such we need a more efficient way to do this calculation if we want to run it on any non-trivial data set. This is where the work by Li [2] and Asi et. al. [6] come in. They propose a solution that allows us to compute the LTS estimator in $O(n^2 log(n))$ time and with $O(n^2)$ storage complexity. Below, we will explore this algorithm in more depth.

2.1.1 Efficient Least Trimmed Squares Calculation. To understand the algorithm for Least Trimmed Squares, we require some of the following definitions which are adapted from [2]

Theorem 2.1. *α similarity: The residual of a point x_i on some (α, β) will be proportional to its residual on some (α', β).*

Proof. We can see that this is true by considering how the residual would change as we move α; all of the predictions of the points would move by the same amount. This means that their residuals would as well, as the residuals are the difference between the predicted value (created by the regression) and the true value. This will allow us to ignore changes in α while we calculate our estimation since all of the residuals will be proportional to the β they are calculated on. □

Theorem 2.2. *h Subset Theorem: For any pair (α, β) if we calculate the residuals (before squaring) for all of the points in the data set Y and sort them the LTS solution must come from a continuous subset of size h from within the ordered set of residuals.*

Proof. We have that $r(\alpha, \beta)_1 \leq r(\alpha, \beta)_2 \leq \cdots \leq r(\alpha, \beta)_n$, Since the residuals are in order, when we move though the list of residuals, we will be adding successively more positive points and removing more negative points forcing us to cross the set of smallest squared residuals when we are centered over zero. □

Definition 2.3. Size H Estimates: Given a data set Y a subset of size h, which we will call H will produce $LTS(H, h)$.

$$\overline{x} = \frac{1}{h}\sum_i x_i \qquad \overline{y} = \frac{1}{h}\sum_i y_i \qquad \overline{xx} = \frac{1}{h}\sum_i x_i^2 \qquad \overline{yy} = \frac{1}{h}\sum_i y_i^2 \qquad \overline{xy} = \frac{1}{h}\sum_i x_i y_i$$

$$\beta = \frac{\overline{xy} - \overline{x}\cdot\overline{y}}{\overline{xx} - (\overline{x})^2} \qquad \alpha = \overline{y} - \beta\overline{x} \qquad ss = \overline{yy} - \beta(\overline{xy} - \overline{x}\cdot\overline{y})$$

Here, we define that, if given a set of size h, we can then calculate its LTS estimate by performing simple linear regression. This is the same solution as the least squares model just with more of the steps declared as their own variables. One important addition is the *ss* variable. Variable *ss* represents the sum of squares for the data set. It is also important to look at how the LTS solution changes as we remove one datapoint and add another one in its place.

Definition 2.4. Size H Estimates change: Given a data set Y a subset of size h, which we will call H and a old point j and new point j', and a previous set of estimates we will produce the below values when removing j and adding j'.

$$\overline{x}' = \overline{x} + \frac{1}{h}(x_{j'} - x_j) \qquad \overline{y}' = \overline{y} + \frac{1}{h}(y_{j'} - y_j) \qquad \overline{xx}' = \overline{xx} + \frac{1}{h}({x_{j'}}^2 - {x_j}^2)$$
$$\overline{yy}' = \overline{yy} + \frac{1}{h}({y_{j'}}^2 - {y_j}^2) \qquad \overline{xy}' = \overline{xy} + \frac{1}{h}(x_{j'}y_{j'} - x_j y_j)$$

From these values β', α', ss' can be calculated.

In this definition, one j is swapped for j'. This will become essential for the run time of the algorithm as it will allow us to iteratively update the solution.

Definition 2.5. In order to find the LTS solution, we will have to check over the possible slopes created by each pair of points within Y and, if necessary, swap the points in the residuals and recheck for the smallest sum of squares.

Proof. The proof of this can be found under proposition 3 in [2] □

With all of these definitions and theorems under our belt, we combine them to construct an algorithm to solve Least Trimmed Squares. We will begin by sorting the data in Y by its x value. We can then create a list of all of the unique, unordered pairs in the data sets and calculate the slopes for each of the pairs, compiling a list of all slopes. We can then sort that list and store it temporally. We will then take the smallest slope, β_1, and pick any value less than β_1 to use as our initial β. Using this β, we can then calculate the residuals for all of the data points and sort them by their absolute values. We can then check all of the ordered subsets of size h to find the one with the smallest residual. Next we iterate over the slopes, swapping, when necessary, *i.e.* swapping when the points would result in a change in the calculation of any of the continuous subsets of size h in the sorted residuals. We only need to swap points if they will be excluded by any of the subsets. We then check again for the smallest continuous residuals over the new set with swapped points. Finally, we output the subset that had the smallest set of residuals, which is found by comparing their sum of squares values, and its α and β coefficients.

```
def findSmallestResidual(residuals, h) {
    val initialRM = RegressionModel(residuals, h, 0)
        (1 to n - h).foldLeft(initialRM)(
            (b: RegressionModel, p1: Int) => {
                val a = b.moveIntervalOver(p1)
                if (a.ss < b.ss) a; else b})}

def iterateThroughSlopes(slopes, residuals, h, defaultRM) {
    slopes.foldLeft(defaultRM)((rm, pp) => {
        if (not in central h) {
            residuals = swapResidualPostions(pp._1, pp._2, residuals)
            val rmNew = partB(residuals, h)
            if (rmNew.ss < rm.ss) rmNew else rm;
      } else rm })}

```

```
def LTS(Y, h){
  val orderedSlopes = Y.sortBy(point => point.x)
    .permutations().map((p1, p2) => (p2.y - p1.y)/(p2.x - p1.x).sort()
  val beta = orderedSlopes.head
  val residuals = Y.calculateResiduals(beta)
  iterateThroughSlopes(orderedSlopes, residulas, h, findSmallestResidual(residulas, h))
}
```

2.2 Minimum Change Least Trimmed Squares

In order to be able to analyze the local sensitivity, we need to come up with a function that measures how β changes when neighboring data sets are considered. To this effect, we created the MC-LTS algorithm. The MC-LTS algorithm heavily relies on the below conjecture:

Conjecture 2.6. *For a specific β, the smallest change you cam make to the data set Y in order to move its estimate to β is buy removing k points from Y*

We support our conjecture with the below theorem.

Theorem 2.7. *The largest change to the LTS solution, β, that can be made by adding or removing one point from Y comes from just the removal of 1 point*

Proof. We will begin by giving the proof for the removal or insertion of 1 element.
Here we have two cases: the first where there is a new element being inserted, the second where an old element being deleted.

(1) If we are only deleting a point from Y, then it is obvious that one of the deletions will cause the largest movement in β as we can break ties arbitrarily.
(2) Now we can consider the case where one element is inserted. LTS has interesting properties when points are inserted because of its ability to trim points with high residuals.
   (a) In the simplest case, we are adding a point p close to the current β therefore p has a smaller residual than at least one of the other points that used to be included. This will cause LTS to include our new point p and trim an older point with a higher residual p_{hr}. This will implicitly have a smaller effect than just removing a point; as when a point is removed from Y, a new point p^+ will be included and it will have a larger residual then p does, leading to a larger change in the slope.
   (b) In the other case, we add a point whose residual is not smaller than one of the h included points. Here, one of two things can happen, either:
      (i) It will not cause the LTS solution to move and will simply be trimmed. Leading to no change in the LTS solution.
      (ii) It will cause the LTS solution to move by making the LTS solution shifting towards a new set of points. In order for LTS to shift in this manner, it would need two bipartite sets of $(n-h)/2$ and $((n-h)/2)+1$ points.Like in Figure 1 This is true because the breakdown point of LTS. To create the largest change possible from an insertion, we would insert one of the points into the smaller set. This shift can also be represented by simply removing one of the points from the initial set, both of which cause a tie between the two sets. If we break ties in a intelligent manner, it is possible to get the same resulting β, thus producing the largest possible change through removal.

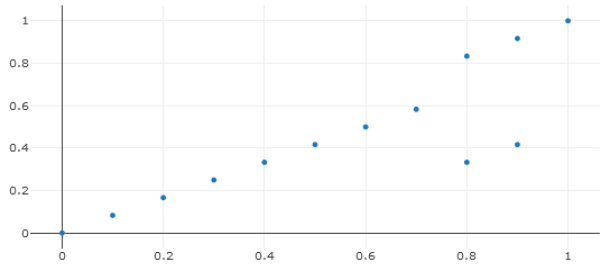

Fig. 1. A Bipartite graph

□

This theorem can then hopefully be applied recursively to the neighbors of the data to satisfy the conjecture. We can then define our new algorithm, Minimum Change Least Trimmed Squares or MC-LTS. It computes over a set of $\{\beta_1, \beta_2, \ldots, \beta_m\}$, for a given β_i $i \in [1, m]$, the minimum number of points which must be changed from our original LTS solution to satisfy a LTS computation for the given β_i or any β_j where $|\beta_{LTS(Y)} - \beta_j| \geq |\beta_{LTS(Y)} - \beta_i|$ and $|\beta_j - \beta_{LTS(Y)}| \geq |\beta_i - \beta_{LTS(Y)}|$ and $j \in [1, m]$. We want to find the minimum number of points that have to be changed to reach our β_i or any β_j that is on the same side of the $\beta_{LTS(Y)}$ as β_i and is father away from $\beta_{LTS(Y)}$ than our β_i. Our algorithm takes in the data set Y, the trimming factor, and a slope value to fit the points to, and the set of all β_i. Subsequently, the algorithm will return a value k that represents the number of data points that are no longer included in the new LTS solution that exist in Y.

$$\text{input: Data set } Y, \text{ Trimming Factor } h, \text{ slope to test } \beta_i, \{\beta_1, \beta_2, \ldots, \beta_m\}$$

$$\text{output: } k$$

$$\text{s.t.} \qquad k = min(|(Y - X)|) \qquad \forall X \in \{\beta_{LTS(X)} = \beta \text{ or } \beta_{LTS(X)} = \beta_j{}^{1}\}$$

The logic of our algorithm behaves as follows.

```
1 def mc_lts(beta, Y, h, betas) {
2     var LTS_solution = LTS(Y, h)
3     var lts_beta = LTS_solution.beta
4     betas.filter((b) => |lts_beta - b| >= |lts_beta - beta| && |b - lts_beta| >= |beta -
          lts_beta|).map((b) => {
5         val newSolution = Y.map((point) => (point, point.computeResiduals(LTS_solution.alpha,
              b)^2)).sortBy(_._2).trim(0,h).map(_._1)
6         h - LTS_solution.hpoints.intersect(newSolution).size() // the number of points that
              have changed
7     }).min}
```

This algorithm has a nice feature in which the ranges can be computed beforehand which greatly reduces the complexity when we run it over all β_i. Giving it a time complexity of $O(m * nlog(n))$, where n is the number of data points and m is the number of slopes (β values) we are checking. The $O(nlog(n))$ component

[1] As defined in the previous paragraph

comes from the sorting that is required when we find the h smallest residuals which is executed for every β value, of which there are m.

2.3 Report Noisy Max

Report Noisy Max (RNM) is a differentially private framework that takes in a set of values, scores the values, adds noise to the scores, and releases the value with the highest score. Conveniently, as long as the score function of RNM is δ sensitive under insertion and deletion, the output of algorithm will be epsilon-differentially private.
In our algorithm, the values that we are scoring are discrete intervals between the minimum possible slope and the maximum possible slope. We score those values with

$$q(Y, \beta) = -|\text{MC-LTS}(beta, Y, h)| \tag{4}$$

which means that the βs with the least number of points that have to be changed will be the most likely to be released with larger deviations from the true β returned from LTS, thus becoming exponentially less likely to be returned.

2.4 Bringing Them Together – DP-LTS-Regression

So far, we have looked at the separate parts of the algorithm in isolation, allowing us to understand them individually. Now, by bringing them together, we can develop a holistic algorithm. Firstly, we calculate the LTS solution. Secondly, we can generate the points over which Report Noisy Max will score. Thirdly, we pass the β and the LTS solution to the score function and produce the score values for RNM. Lastly, once the noise has been added, we can release the β corresponding to the highest noisy score. This algorithm we will call DP-LTS-Regression.

```
def DP-LTS-Regression(Y, h, epsilon) {
    val lts_solution = LTS(Y,H)
    val betas_to_score = (min_slope to max_slope by granularity)
    val scores = betas_to_score.map(beta => (MC-LTS(beta, Y, h, LTS), beta)) //use pre-
        calculated LTS
    val noisy_scores = scores.map(sc => (sc.score + Exp(2*delta / epsilon), sc.beta))
    return min_by(scores.score).beta
}
```

This algorithm has polynomial time complexity. The LTS step is $O(n^2 log(n))$, and all the scoring steps combined have a time complexity of $O(mnlog(n))$ where m is the inverse of the level of granularity, as the sorting step for finding the smallest squared residuals is the most expensive step. Additionally, the final min step is just $O(m)$. So combined this algorithm has a time complexity of

$$O(n^2 log(n) + mnlog(n) + m) = O((m + n)nlog(n)) \tag{5}$$

2.5 Accuracy

The accuracy of this algorithm does not have a formal proof at the time of writing this. We only have evidence from the experiments below. The accuracy hinges on weather or not MC-LTS is good measure of

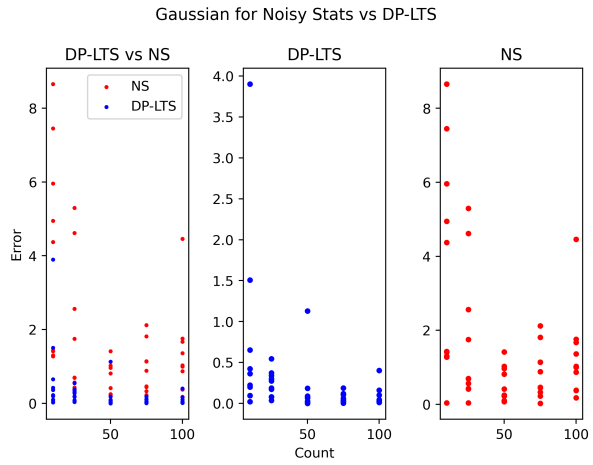

Fig. 2. Gaussian Distribution, 10 - 100

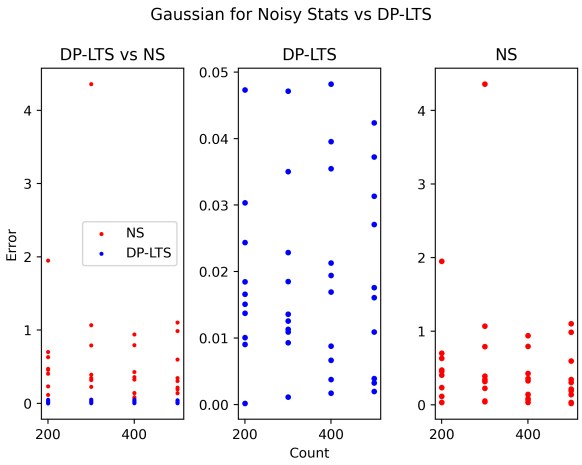

Fig. 3. Gaussian Distribution, 100 - 1000

local sensitivity for the data and on weather the conjecture from the MC-LTS section holds. If MC-LTS underestimates the possible change in β, then the output will not be deferentially private. Conversely, if it is a good estimate or overestimate of the true possible change in β, our result will be differentially private.

3 EXPERIMENTS

In order to test our accuracy, we ran our algorithm on on data sets of size 10, 20, 50, 75, 100, 200, 300, 400, and 500 with varying distributions: Gaussian, bimodal, and outlier. For each of these size and distribution combinations, we generated 10 unique data sets. DP-LTS-Regression was then performed on all of these generated data sets 10 times so that we can compare the real and average β values after completion.

In addition, we also opted to run the NoisyStats [1] algorithm on each of the distributions, and we compared the values we found from the Noisy Stats algorithm to DP-LTS to potentially determine if one was able to better approximate the true β. We found that for data sets of size less than 100, our DP-LTS algorithm was often slightly closer in approximating slope than Noisy Stats, and on data sets of size 100 to 1000, DP-LTS had significantly less error.

3.0.1 Gaussian Noise. We generated our linear Gaussian distribution by randomly selecting x values from a Gaussian distribution and calculated y by adding noise from a new Gaussian distribution to the caluculated x. Without the additional Gaussian noise added to the y value, all of our generated points would sit on the line $y = x$ by this method. This is visible in figure 8. As mentioned above, we generated 10 unique data sets per data set size and ran our DP-LTS computation those points 10 times and our Noisy stats computation 10 times.

We found that for data sets with size less than 100, our DP-LTS distribution had error about 10-times smaller than the error from NoisyStats, as illustrated by figure 2. The first graph in this figure shows the comparison between DP-LTS and NoisyStats. Notably, the values for DP-LTS are demonstrably lower than those for NoisyStats, as we can see the error is no more than 5.5013 for DP-LTS on data sets less than 100, but ranges from 0.0278 to as high as 8.6514 for NoisyStats.

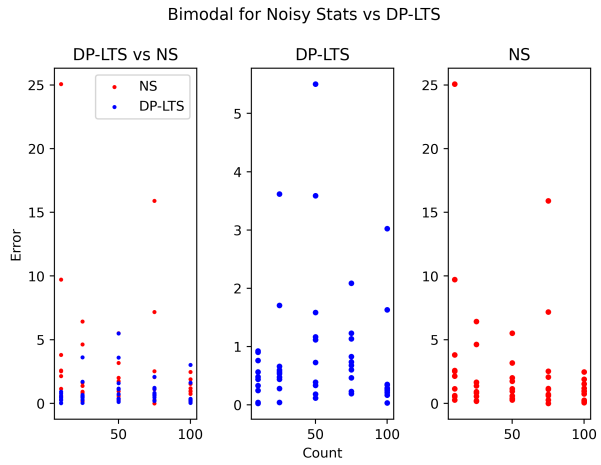

Fig. 4. Bimodal Distribution, 10 - 100

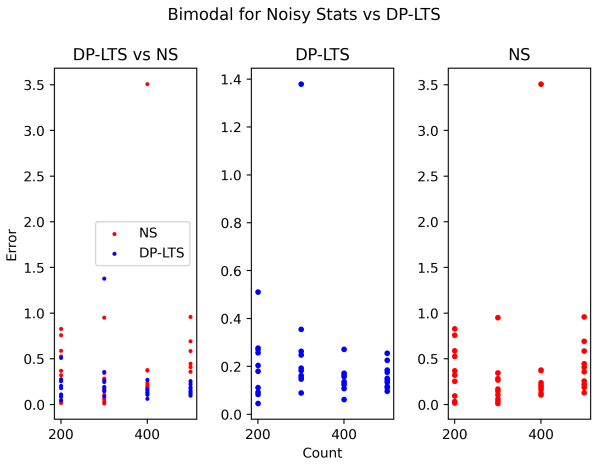

Fig. 5. Bimodal Distribution, 100 - 1000

For data sets larger than 100, we found that our DP-LTS algorithm had significantly less error as visible in figure 3. While the error level ranged as high as 3.800 for NoisyStats, the maximum error recorded for DP-LTS was 0.4031, which lies within the range of 0.0205 - 1.0684 that most NoisyStats results occur within.

Overall, we found that DP-LTS has less error than the NoisyStats algorithm on the same data sets when compared the the true value of β computed by the Least Squares regression on the data set.

3.0.2 Bimodal. We generated our bimodal distribution by generating x values from a Gaussian distribution and calculating y by taking the value of x, randomly selecting $(P = .5)$ if we multiply by 2 and add 10, and then add noise from a Gaussian distribution else generating points in the same way as the Gaussian experiment. This means that without the additional Gaussian noise added to the y value, a random half of our generated points would sit on the line $y = x$ and the other half would sit on the line $y = 2x + 10$. This is visible in figure 9.

For data sets with size less than 100, we found that DP-LTS performed similarly to NoisyStats for data sets of size 10 and 25, however, as data size increased, we found that our DP-LTS algorithm has less error than NoisyStats, as shown in Figure 4. The average error for DP-LTS was 0.8629 whereas the average error for NoisyStats was 2.4935. DP-LTS also has a smaller maximum error of 5.5013 compares to the maximum error of NoisyStats, 25.0613. Thus, on average, DP-LTS has less error than NoisyStats for data sets of size 100 or fewer.

For data sets of size greater than 100 and up to 500, the error for DP-lTS was less than or equal to half of the average error of NoisyStats for each of the data sizes as illustrated by figure 5. DP-LTS average error for all data sets between 200 and 500 is 0.2073 with a maximum 1.3789. For NoisyStats, average error was 0.3976 with a maximum 3.5066. While some of the errors in DP-LTS were greater than some of NoisyStats errors, the averages suggest that DP-LTS also showed lesser error for large bimodal distributions.

3.0.3 Outliers. Our comparison for outlier is unique in that we compare DP-LTS and NoisyStats against the Least Trimmed Squares solution, as opposed to the Least Squares regression. We compared the algorithms to LTS because its elimination properties will give a better estimate for the majority of the data compared to Least Squares regression.

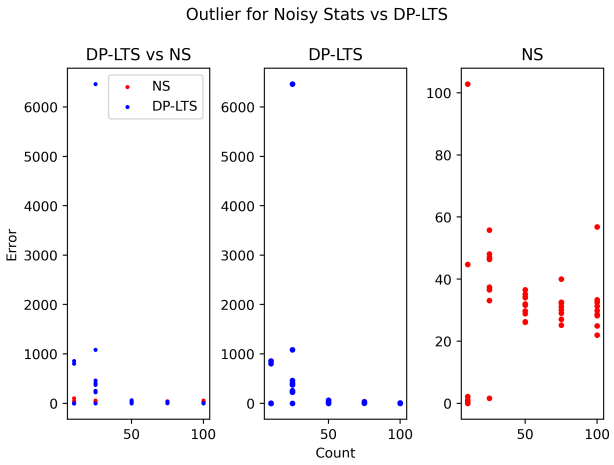

Fig. 6. Outlier Distribution, 10 - 100

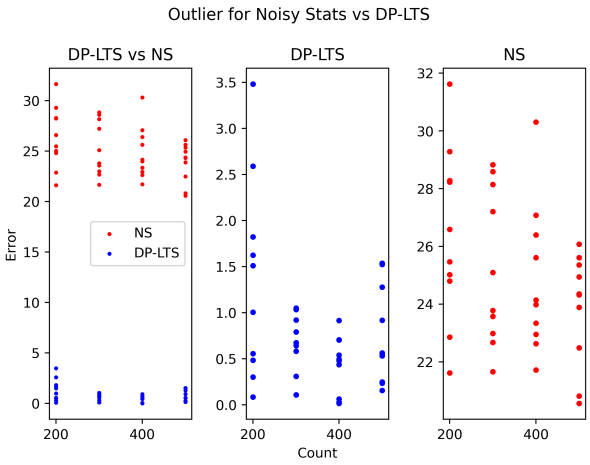

Fig. 7. Outlier Distribution, 100 - 1000

We generated our outlier distribution by again generating x values from a Gaussian distribution and calculating all but 3 y values by taking the value of x and adding noise to the value from a Gaussian distribution. The remaining 3 (x, y) value pairs are generated by generating two unique Gaussian values for x and y and adding 200 to x. This means that there will be large outliers away from the majority of the data as illustrated by figure 10.

We found that for our runs on data sets of size 10 and 25, NoisyStats severely outperformed DP-LTS with a average error of 27.228 compared to 569.827 and a maximum error of 102.776 compared to 6462.972. This suggests that our DP-LTS algorithm is weaker when it only has outliers as trimmed points and no points within the data set to trim. This issue could likely be solved by making h smaller but this is problematic because of the size of the data; 10 and 25 are small so the will each only have 4 and 5 points removed, respectively. In figure 6, this difference is visible where all DP-LTS error values are much larger than the NoisyStats values for size 10 and 25. However, when we run both algorithms on data sets of size 50, 75 and 100, we find the average error of DP-LTS is now 10.435 where NoisyStats is 31.586, and DP-LTS again has smaller error values than NoisyStats.

For data sets of size greater than 100, DP-LTS has significantly less error than NoisyStats as illustrated by figure 7. The first chart of this figure shows clearly that for all data set sizes, the error for NoisyStats is greater than that of DP-LTS. Each of NoisyStats has a average error of 25.072 maximum of 31.619 and DP-LTS has an average of 0.789 and a maximum of 3.4793.

Compared to all of the other distributions and larger sized data sets, DP-LTS has drastically less error when run on the outlier distribution.

4 CONCLUSION

Through our analysis, we found that when we analyzed a variety of data sets and perform our algorithm on them, the majority of slope values returned were within a reasonable window of the true value of linear regression. Additionally, DP-LTS frequently had less error than the NoisyStats algorithm in our experiments, which suggests that DP-LTS may be a better estimator for linear regression. The next iteration of our algorithm's development is to prove that it is differentially private in theory as well as practice. There is a good amount of room for future work on this subject as well; firstly, we will want to ensure that our

given conjecture is true, and secondly, we would want to analyze our algorithm on data sets that are more sporadic than the ones that we have supplied in this paper.

REFERENCES

[1] Daniel Alabi and Audra McMillan and Jayshree Sarathy and Adam Smith and Salil Vadhan, "Deferentially Private Simple Linear Regression," *ArXiv*, arXiv:2007.05157, 2020.

[2] Lei M. Li, "An algorithm for computing exact least-trimmed squares estimate of simple linear regression with constraints", *JCSDA*, j.csda.2004.04.003, 2004.

[3] Kobbi Nissim and Sofya Raskhodnikova and Adam Smith, "Smooth Sensitivity and Sampling in Private Data Analysis" *ACM*, ACM 978-1-59593-631-8/07/0006, 2007.

[4] Hilal Asi and John C. Duchi, "The importance of better models in stochastic optimization" *ArXiv*, arXiv:1903.08619v1, 2019.

[5] Hilal Asi and John C. Duchi, "Near Instance-Optimality in Differential Privacy" *ArXiv*, arXiv:2005.10630v1, 2020.

[6] Ola Hössjer, "Exact computation of the least trimmed squares estimate in simple linear regression", Computational Statistics & Data Analysis, Volume 19, Issue 3, 1995, Pages 265-282, ISSN 0167-9473, https://doi.org/10.1016/0167-9473(95)92697-V.

5 APPENDIX

5.1 Experiment Types

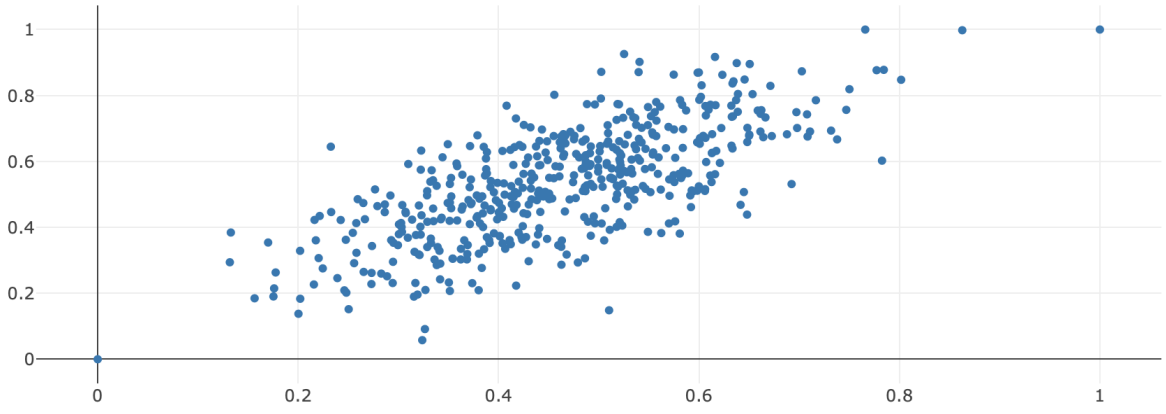

Fig. 8. Gaussian Distribution, data size 500

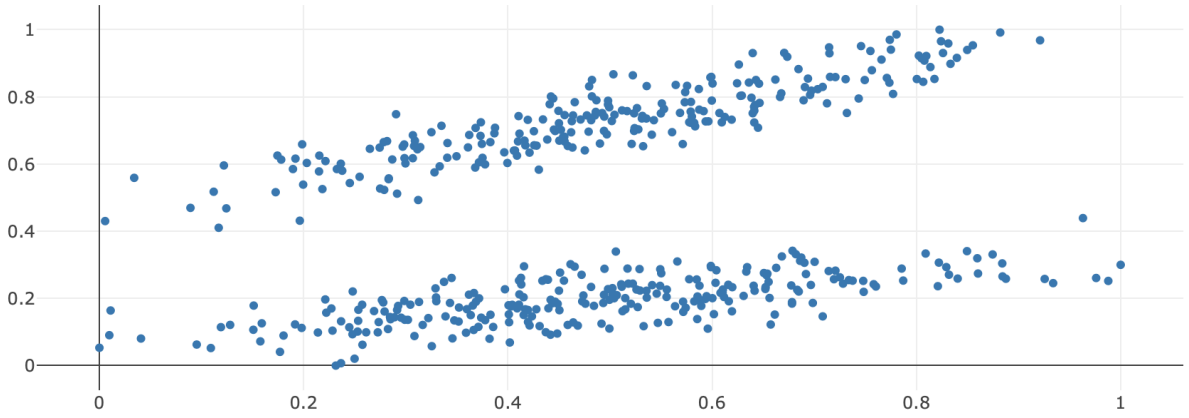

Fig. 9. Bimodal Distribution, data size 500

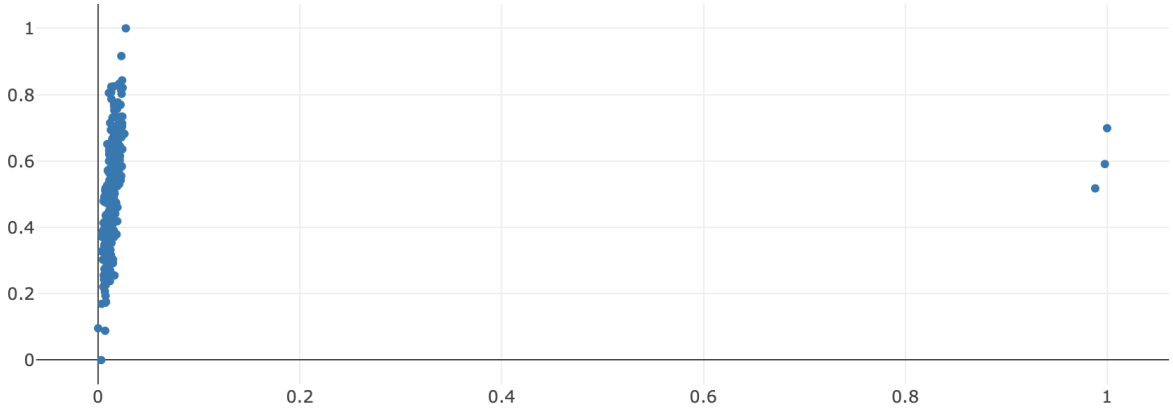

Fig. 10. Outlier Distribution, data size 500

5.2 STD Values for Experiments

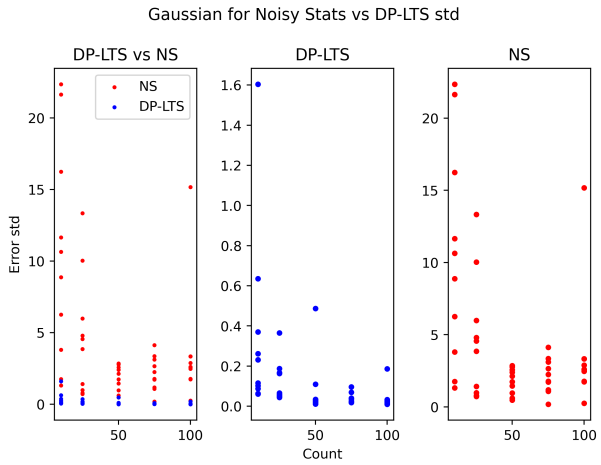

Fig. 11. Gaussian Distribution std, 10 - 100

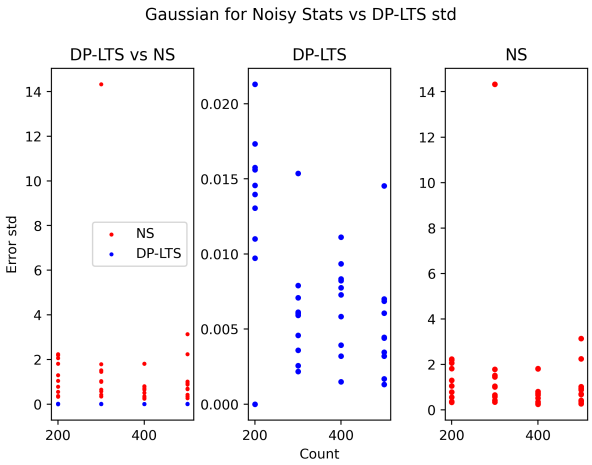

Fig. 12. Gaussian Distribution std, 100 - 1000

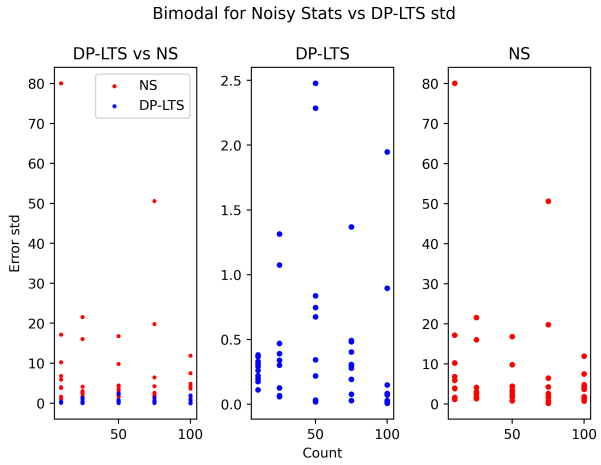

Fig. 13. Bimodal Distribution std, 10 - 100

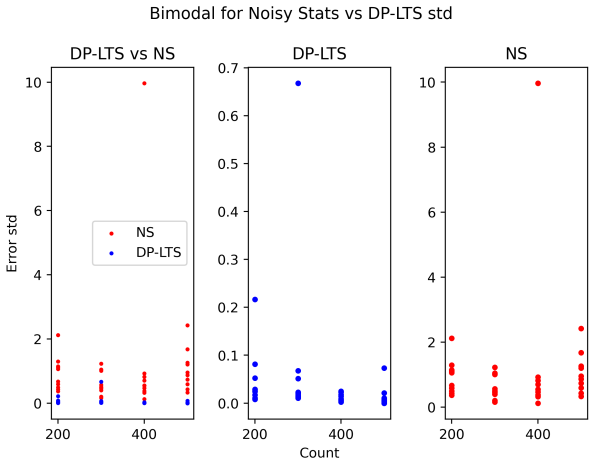

Fig. 14. Bimodal Distribution std, 100 - 1000

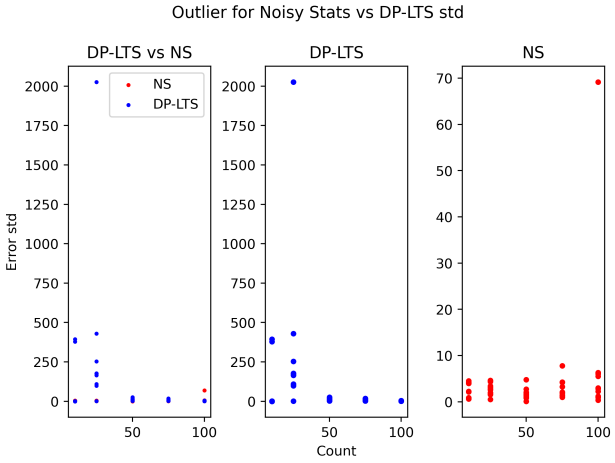

Fig. 15. Outlier Distribution std, 10 - 100

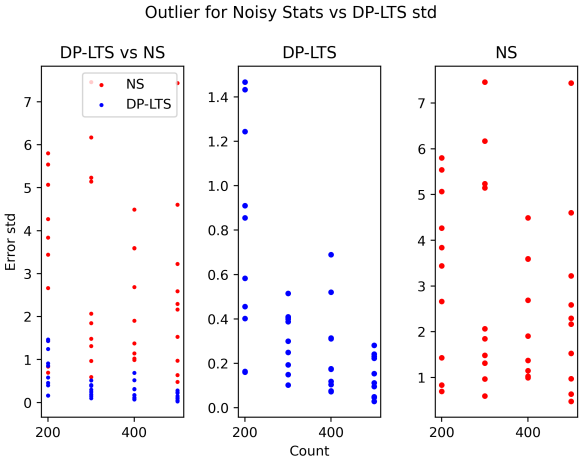

Fig. 16. Outlier Distribution std, 100 - 1000

5.3 Output of a Run of the Code

DataType	i	points	runs	trueBeta	DPLTSBetaAvg	DPLTSBetaAvgError	DPLTSBetaStd	NSBetaAvg	NSBetaAvgError	NSStd
Gaussian	0	10	10	0.8013	0.7071	0.0942	0.0877	0.8376	0.0363	10.6483
Bimodal	0	10	10	0.6343	1.0753	0.4411	0.2198	0.027	0.6073	1.6601
Outlier	0	10	10	0.1479	0.2731	0.1252	0.1071	2.2299	2.0821	4.0673
Gaussian	1	10	10	0.5676	0.5452	0.0224	0.0627	9.219	8.6514	21.6352
Bimodal	1	10	10	-0.2751	-1.2	0.9249	0.3816	24.7862	25.0613	80.0294
Outlier	1	10	10	-0.19	-0.0435	0.1465	0.1146	-1.2951	1.1051	4.1224
Gaussian	2	10	10	0.8173	1.0156	0.1983	0.1158	-5.1434	5.9608	16.2444
Bimodal	2	10	10	1.0669	1.3143	0.2474	0.1747	10.7821	9.7152	17.1815
Outlier	2	10	10	-0.3713	-0.1488	0.2225	0.1646	-2.5522	2.1809	3.975
Gaussian	3	10	10	0.9987	0.9037	0.095	0.2314	2.2737	1.275	6.2599
Bimodal	3	10	10	0.973	0.93	0.043	0.1105	0.4362	0.5368	3.9857
Outlier	3	10	10	0.1174	0.4524	0.335	0.1165	-0.5099	0.6273	0.8574
Gaussian	4	10	10	0.8471	0.623	0.2241	0.0602	5.2193	4.3722	8.8743
Bimodal	4	10	10	0.0377	-0.7252	0.7629	0.3097	-1.1041	1.1417	5.9362
Outlier	4	10	10	-0.0692	-0.031	0.0383	0.0743	0.9639	1.0332	2.2356
Gaussian	5	10	10	0.8394	4.7397	3.9003	1.6035	5.7857	4.9463	11.6557
Bimodal	5	10	10	0.2497	-0.0798	0.3295	0.2921	-0.019	0.2687	1.143
Outlier	5	10	10	-0.1055	-0.0115	0.094	0.1835	0.2698	0.3753	3.9777
Gaussian	6	10	10	0.778	1.1415	0.3635	0.1018	2.1888	1.4108	3.8041
Bimodal	6	10	10	-0.2192	0.3475	0.5666	0.3303	3.5811	3.8003	10.2611
Outlier	6	10	10	43.5832	-813.336	856.9192	393.3953	-1.1674	44.7506	2.1544
Gaussian	7	10	10	0.8589	1.511	0.652	0.3697	-0.4416	1.3005	1.3171
Bimodal	7	10	10	0.6094	1.5127	0.9034	0.1968	-1.9652	2.5745	3.8355
Outlier	7	10	10	0.0475	0.2672	0.2197	0.2424	-0.0513	0.0988	0.6705
Gaussian	8	10	10	0.7138	2.2221	1.5083	0.6349	-0.7161	1.4299	1.7598
Bimodal	8	10	10	0.7408	0.2581	0.4828	0.2621	3.2664	2.5256	6.8081
Outlier	8	10	10	0.1489	0.1522	0.0033	0.0688	0.1135	0.0354	0.5837
Gaussian	9	10	10	0.977	1.401	0.424	0.2613	8.4281	7.451	22.343
Bimodal	9	10	10	0.3174	0.3454	0.028	0.3666	2.4547	2.1373	10.2251
Outlier	9	10	10	99.3674	-701.4776	800.845	378.3644	-3.4092	102.7766	4.4981
Gaussian	0	25	10	0.8646	1.1389	0.2744	0.1665	-1.6996	2.5641	4.5645
Bimodal	0	25	10	0.3488	0.9142	0.5654	0.3904	4.9846	4.6358	16.0491
Outlier	0	25	10	37.3576	278.1285	240.7709	177.7251	-0.0611	37.4186	1.5657
Gaussian	1	25	10	0.8917	0.9693	0.0777	0.0477	0.3246	0.5671	0.724
Bimodal	1	25	10	0.2072	-1.4997	1.7069	1.074	-1.4437	1.6509	4.132
Outlier	1	25	10	0.2648	0.2542	0.0106	0.1425	-1.35	1.6148	4.6155
Gaussian	2	25	10	0.6704	0.8444	0.1739	0.0562	-3.9466	4.617	10.0339
Bimodal	2	25	10	0.9109	1.5728	0.6619	0.301	-0.0044	0.9153	1.3331
Outlier	2	25	10	35.2179	265.2228	230.0049	174.3618	-1.4388	36.6567	2.592
Gaussian	3	25	10	0.7542	0.9419	0.1877	0.0574	1.453	0.6988	5.9862
Bimodal	3	25	10	0.6868	4.3029	3.6161	1.3142	0.4855	0.2013	1.4642

DataType	i	points	runs	trueBeta	DPLTSBetaAvg	DPLTSBetaAvgError	DPLTSBetaStd	NSBetaAvg	NSBetaAvgError	NSStd
Outlier	3	25	10	45.6312	508.8296	463.1984	164.8855	-0.7656	46.3968	2.0046
Gaussian	4	25	10	0.7994	0.7177	0.0817	0.0444	6.0967	5.2973	13.34
Bimodal	4	25	10	0.8166	0.861	0.0444	0.058	1.6467	0.8301	2.6478
Outlier	4	25	10	49.4953	1134.1297	1084.6344	429.2846	1.4175	48.0778	4.3405
Gaussian	5	25	10	0.7146	1.2602	0.5456	0.3646	0.7535	0.039	4.802
Bimodal	5	25	10	-0.0306	-0.5664	0.5359	0.1258	-6.4638	6.4333	21.5882
Outlier	5	25	10	32.9346	257.661	224.7264	97.781	-0.1683	33.1028	3.0221
Gaussian	6	25	10	0.5326	0.1929	0.3396	0.0651	-1.217	1.7496	3.8544
Bimodal	6	25	10	0.1326	-0.1493	0.2819	0.0661	-1.5304	1.6629	2.9924
Outlier	6	25	10	55.9579	427.3105	371.3526	109.341	0.1419	55.816	0.5066
Gaussian	7	25	10	0.7067	1.008	0.3012	0.163	0.2787	0.428	1.4166
Bimodal	7	25	10	0.2245	-0.3739	0.5984	0.3384	0.7817	0.5573	1.7462
Outlier	7	25	10	46.3046	299.7928	253.4882	175.0474	-0.7022	47.0068	3.4065
Gaussian	8	25	10	0.7472	1.1193	0.3721	0.1875	0.3343	0.4129	0.7905
Bimodal	8	25	10	0.1036	-0.3761	0.4796	0.0654	-0.0677	0.1712	2.476
Outlier	8	25	10	46.5807	6509.5529	6462.9721	2025.1164	-0.2449	46.8256	1.7363
Gaussian	9	25	10	0.7388	0.7758	0.037	0.0488	0.0458	0.693	0.9838
Bimodal	9	25	10	0.7625	1.2021	0.4396	0.469	2.137	1.3745	3.0649
Outlier	9	25	10	37.8455	444.2864	406.4408	253.2854	1.2623	36.5832	2.6555
Gaussian	0	50	10	0.7511	1.8808	1.1297	0.4864	1	0.2489	2.5724
Bimodal	0	50	10	0.0407	-0.2944	0.3351	0.0326	1.1824	1.1417	2.6546
Outlier	0	50	10	28.5021	67.8855	39.3833	16.8894	-0.4145	28.9166	1.2801
Gaussian	1	50	10	0.6743	0.6906	0.0163	0.0241	2.0886	1.4144	2.8538
Bimodal	1	50	10	0.2923	5.7936	5.5013	2.4771	-1.7182	2.0105	4.4222
Outlier	1	50	10	26.198	23.0325	3.1655	2.9828	-0.0597	26.2577	0.1039
Gaussian	2	50	10	0.7823	0.7991	0.0169	0.0262	0.367	0.4152	0.6138
Bimodal	2	50	10	-0.0998	-0.4899	0.3901	0.2187	0.2953	0.3951	0.7715
Outlier	2	50	10	32.4088	39.0402	6.6313	2.7805	2.6438	29.7651	4.7842
Gaussian	3	50	10	0.8679	0.9344	0.0665	0.0199	-0.1553	1.0232	0.4769
Bimodal	3	50	10	0.5936	1.71	1.1164	0.8368	1.5448	0.9512	3.5095
Outlier	3	50	10	32.7575	93.8837	61.1262	25.3458	0.7109	32.0466	1.6267
Gaussian	4	50	10	0.7687	0.7671	0.0016	0.0127	0.992	0.2233	0.9686
Bimodal	4	50	10	0.5204	0.7036	0.1832	0.0261	0.2603	0.2601	1.7092
Outlier	4	50	10	34.376	55.233	20.857	11.0112	0.0669	34.3091	2.0317
Gaussian	5	50	10	0.7159	0.625	0.0909	0.0335	0.62	0.0959	1.7349
Bimodal	5	50	10	0.446	0.5645	0.1185	0.0187	5.9642	5.5181	16.7972
Outlier	5	50	10	32.3624	48.9495	16.587	10.878	0.7015	31.6609	0.9738
Gaussian	6	50	10	0.6017	0.4179	0.1838	0.1099	0.5336	0.0681	2.7775
Bimodal	6	50	10	0.5283	1.6961	1.1678	0.7472	-1.2174	1.7457	3.2092
Outlier	6	50	10	35.4469	22.4709	12.976	5.6814	0.2933	35.1536	1.65
Gaussian	7	50	10	0.9784	0.9716	0.0068	0.0104	0.1658	0.8126	2.1386
Bimodal	7	50	10	0.8642	1.5914	0.7273	0.3427	1.1868	0.3226	2.0481
Outlier	7	50	10	26.0326	32.3074	6.2747	5.0333	-0.1357	26.1683	0.8001
Gaussian	8	50	10	0.612	0.6387	0.0267	0.0331	0.855	0.243	2.4063
Bimodal	8	50	10	0.3368	1.9232	1.5864	0.674	-2.8386	3.1753	9.8197
Outlier	8	50	10	37.7477	37.6285	0.1192	3.3682	1.1944	36.5533	2.0169
Gaussian	9	50	10	0.501	0.4286	0.0724	0.0334	-0.4645	0.9655	1.4469
Bimodal	9	50	10	0.4411	4.0313	3.5903	2.2847	1.0316	0.5906	4.3308
Outlier	9	50	10	34.9346	54.5606	19.626	14.877	0.8799	34.0548	2.7195
Gaussian	0	75	10	0.6927	0.7986	0.1059	0.0963	-0.1885	0.8813	2.2486
Bimodal	0	75	10	0.462	-0.1398	0.6018	0.3076	7.6416	7.1796	19.8079
Outlier	0	75	10	31.0157	34.011	2.9952	2.0853	-0.1516	31.1674	1.4277
Gaussian	1	75	10	0.7969	0.6782	0.1187	0.0694	2.9182	2.1213	4.1245
Bimodal	1	75	10	0.3508	-0.881	1.2318	0.4827	0.3425	0.0083	1.974
Outlier	1	75	10	33.9342	66.6447	32.7105	17.2686	1.6883	32.2459	7.7639
Gaussian	2	75	10	0.6482	0.6593	0.0111	0.0179	1.1066	0.4585	1.1832
Bimodal	2	75	10	0.413	0.8801	0.4671	0.1917	-0.2009	0.6139	0.76
Outlier	2	75	10	31.8395	40.4097	8.5702	6.129	-0.4944	32.3339	1.9442
Gaussian	3	75	10	0.5843	0.6436	0.0592	0.04	0.6121	0.0278	1.0745
Bimodal	3	75	10	0.4505	1.1886	0.738	0.2797	1.5551	1.1045	1.6248
Outlier	3	75	10	30.8371	49.9054	19.0684	10.3129	0.6782	30.1589	4.2654
Gaussian	4	75	10	0.6111	0.4245	0.1866	0.0251	-1.2013	1.8124	3.3554
Bimodal	4	75	10	0.4147	1.0915	0.6767	0.297	-0.7609	1.1756	2.632
Outlier	4	75	10	26.9204	28.5867	1.6662	2.4373	1.7177	25.2027	3.265
Gaussian	5	75	10	0.6785	0.657	0.0215	0.0198	0.3501	0.3284	3.1205
Bimodal	5	75	10	0.6353	1.7679	1.1326	0.4916	16.5332	15.8979	50.5966

DataType	i	points	runs	trueBeta	DPLTSBetaAvg	DPLTSBetaAvgError	DPLTSBetaStd	NSBetaAvg	NSBetaAvgError	NSStd
Outlier	5	75	10	28.5645	37.7686	9.2041	7.2312	-0.4707	29.0353	0.9532
Gaussian	6	75	10	0.629	0.5932	0.0358	0.0364	-0.5058	1.1349	1.7261
Bimodal	6	75	10	0.7586	1.5891	0.8305	0.4037	3.2849	2.5263	6.4391
Outlier	6	75	10	26.7544	20.6759	6.0785	2.2301	-0.2888	27.0432	1.2983
Gaussian	7	75	10	0.7888	0.7622	0.0266	0.0198	1.9296	1.1407	2.6553
Bimodal	7	75	10	0.4267	0.1938	0.2329	0.0762	0.1497	0.277	0.2072
Outlier	7	75	10	32.3684	26.5993	5.7692	4.0697	-0.2375	32.606	2.0728
Gaussian	8	75	10	0.707	0.7441	0.0371	0.0229	0.4822	0.2248	1.7713
Bimodal	8	75	10	0.6598	2.7488	2.089	1.3682	-1.401	2.0608	4.2556
Outlier	8	75	10	40.5462	42.4461	1.8999	2.8415	0.5524	39.9938	1.9491
Gaussian	9	75	10	0.6171	0.6214	0.0043	0.0189	0.1736	0.4435	0.1933
Bimodal	9	75	10	0.3175	0.1245	0.193	0.0272	1.0328	0.7154	1.9958
Outlier	9	75	10	33.2728	48.8731	15.6004	6.6008	0.7276	32.5451	4.2002
Gaussian	0	100	10	0.6278	0.6165	0.0112	0.0159	0.4486	0.1791	0.2615
Bimodal	0	100	10	0.4586	0.2495	0.2091	0.0273	2.3506	1.892	7.5187
Outlier	0	100	10	34.6113	38.528	3.9168	3.3314	1.9406	32.6707	5.5051
Gaussian	1	100	10	0.7911	0.8361	0.045	0.0215	0.4139	0.3773	2.5802
Bimodal	1	100	10	0.5948	0.883	0.2883	0.0819	1.7777	1.183	4.2391
Outlier	1	100	10	31.5584	32.2415	0.6831	1.9451	-1.0053	32.5637	2.8226
Gaussian	2	100	10	0.8533	0.8675	0.0142	0.0165	1.8515	0.9982	3.3394
Bimodal	2	100	10	0.3013	0.1319	0.1694	0.0193	0.1492	0.1521	1.4369
Outlier	2	100	10	28.0408	26.9085	1.1322	0.9957	-1.8946	29.9353	6.2824
Gaussian	3	100	10	0.6551	0.6738	0.0187	0.0169	2.0149	1.3598	2.6058
Bimodal	3	100	10	0.1823	-0.172	0.3542	0.1478	2.6524	2.4701	11.9301
Outlier	3	100	10	33.2039	34.1104	0.9065	0.7279	-0.1304	33.3344	0.9032
Gaussian	4	100	10	0.5249	0.4246	0.1003	0.0103	4.9828	4.4579	15.1682
Bimodal	4	100	10	0.1816	-0.093	0.2746	0.019	-0.0608	0.2424	0.7164
Outlier	4	100	10	23.0133	26.1664	3.153	2.1126	1.021	21.9923	6.0596
Gaussian	5	100	10	0.5885	0.6037	0.0152	0.0261	0.9624	0.3739	2.4744
Bimodal	5	100	10	0.5089	0.1638	0.3451	0.0715	-0.4645	0.9734	1.4834
Outlier	5	100	10	32.0367	39.3822	7.3455	3.3377	0.7415	31.2952	1.1754
Gaussian	6	100	10	0.7064	1.1095	0.4031	0.186	-0.3218	1.0282	1.7814
Bimodal	6	100	10	0.4385	0.1944	0.2441	0.0081	-0.5049	0.9434	1.8101
Outlier	6	100	10	26.5	26.5788	0.0788	0.5555	1.5736	24.9264	2.9914
Gaussian	7	100	10	0.8309	0.8668	0.0358	0.0247	-0.8424	1.6733	2.8916
Bimodal	7	100	10	0.3974	0.4304	0.033	0.0242	0.3398	0.0575	0.7613
Outlier	7	100	10	28.6632	30.1741	1.5109	1.1524	0.0478	28.6154	0.35
Gaussian	8	100	10	0.5768	0.7389	0.1622	0.0334	1.4448	0.8681	1.7483
Bimodal	8	100	10	0.3459	-1.2869	1.6329	0.8964	1.8799	1.534	4.8422
Outlier	8	100	10	28.8014	30.8234	2.0219	1.5507	0.5672	28.2343	2.2375
Gaussian	9	100	10	0.9177	0.9004	0.0174	0.0103	-0.8393	1.7571	3.34
Bimodal	9	100	10	0.7223	3.747	3.0247	1.9478	-0.0323	0.7546	3.6361
Outlier	9	100	10	35.3153	33.302	2.0133	1.205	-21.5047	56.82	69.1449
Gaussian	0	200	10	0.8605	0.9078	0.0473	0.0173	0.4569	0.4036	0.3343
Bimodal	0	200	10	0.5446	0.452	0.0926	0.0104	0.2899	0.2547	0.3683
Outlier	0	200	10	24.7562	26.578	1.8217	1.466	1.8962	22.86	5.8016
Gaussian	1	200	10	0.8353	0.8454	0.0101	0	0.8016	0.0337	1.296
Bimodal	1	200	10	0.4307	0.6885	0.2578	0.0814	0.4642	0.0335	0.4284
Outlier	1	200	10	21.8034	22.3573	0.5538	0.5833	0.1817	21.6218	1.4339
Gaussian	2	200	10	0.6589	0.6893	0.0304	0.0213	0.2019	0.457	0.362
Bimodal	2	200	10	0.5499	0.7302	0.1804	0.025	1.3091	0.7592	1.0618
Outlier	2	200	10	28.8347	31.4247	2.5899	0.164	-0.4484	29.2832	3.4434
Gaussian	3	200	10	0.8583	0.8673	0.0091	0.0097	0.1583	0.6999	2.1913
Bimodal	3	200	10	0.4028	0.6746	0.2718	0.0244	-0.1862	0.589	1.1079
Outlier	3	200	10	24.8517	25.1513	0.2996	0.4018	-1.743	26.5947	5.0681
Gaussian	4	200	10	0.6378	0.6562	0.0185	0.0158	0.7513	0.1136	0.7802
Bimodal	4	200	10	0.5107	0.5567	0.046	0.0078	1.3403	0.8296	2.1198
Outlier	4	200	10	28.2819	28.7652	0.4834	0.9106	0.0524	28.2295	0.8354
Gaussian	5	200	10	0.7462	0.7628	0.0166	0.0146	1.2184	0.4721	1.0478
Bimodal	5	200	10	0.3009	0.2173	0.0836	0.0098	-0.0216	0.3224	0.4939
Outlier	5	200	10	27.9283	28.013	0.0847	0.1604	-0.3517	28.2799	3.8384
Gaussian	6	200	10	0.7691	0.754	0.0151	0.0131	0.1389	0.6302	2.0637
Bimodal	6	200	10	0.441	0.2366	0.2044	0.0521	0.0703	0.3707	0.6719
Outlier	6	200	10	25.966	24.3415	1.6245	1.2441	1.1665	24.7994	2.6628
Gaussian	7	200	10	0.6199	0.6197	0.0002	0.0156	2.57	1.9501	2.239
Bimodal	7	200	10	0.5538	0.2771	0.2767	0.0287	0.5719	0.0181	1.2965

DataType	i	points	runs	trueBeta	DPLTSBetaAvg	DPLTSBetaAvgError	DPLTSBetaStd	NSBetaAvg	NSBetaAvgError	NSStd
Outlier	7	200	10	27.8873	29.3966	1.5093	0.8549	2.864	25.0233	5.5412
Gaussian	8	200	10	0.6455	0.6211	0.0244	0.014	0.6137	0.0318	1.8103
Bimodal	8	200	10	0.3793	0.2679	0.1114	0.0173	0.4749	0.0956	1.1428
Outlier	8	200	10	31.8795	32.8846	1.0051	0.4553	0.2597	31.6198	0.696
Gaussian	9	200	10	0.7147	0.7285	0.0137	0.011	0.4817	0.233	0.553
Bimodal	9	200	10	0.4851	0.9964	0.5113	0.2164	-0.0413	0.5264	0.5883
Outlier	9	200	10	27.4483	30.9276	3.4793	1.4327	1.9832	25.4651	4.2689
Gaussian	0	300	10	0.7169	0.7354	0.0185	0.0154	1.0384	0.3215	1.445
Bimodal	0	300	10	0.3326	0.1393	0.1934	0.0105	1.2828	0.9501	1.0444
Outlier	0	300	10	24.8752	25.9253	1.0501	0.5154	1.0963	23.7789	5.2346
Gaussian	1	300	10	0.6214	0.5742	0.0471	0.0022	0.3972	0.2242	0.3419
Bimodal	1	300	10	0.4683	0.3794	0.0889	0.0138	0.3621	0.1061	0.2039
Outlier	1	300	10	24.0543	25.0874	1.0331	0.2496	0.4808	23.5734	1.4826
Gaussian	2	300	10	0.7786	0.7879	0.0093	0.0061	1.1217	0.3431	1.7865
Bimodal	2	300	10	0.3059	0.1438	0.162	0.0118	0.0215	0.2843	0.5682
Outlier	2	300	10	22.4353	21.6457	0.7895	0.1493	-0.2344	22.6697	1.3162
Gaussian	3	300	10	0.7193	0.7422	0.0229	0.0036	1.5086	0.7892	1.5123
Bimodal	3	300	10	0.3691	0.2148	0.1543	0.019	0.3088	0.0603	0.4995
Outlier	3	300	10	24.2266	24.8686	0.6419	0.3881	-2.9842	27.2108	7.4567
Gaussian	4	300	10	0.7284	0.7158	0.0126	0.006	5.0848	4.3565	14.3227
Bimodal	4	300	10	0.6858	1.041	0.3552	0.0674	1.0305	0.3447	1.0113
Outlier	4	300	10	28.2871	28.3948	0.1077	0.1019	-0.5376	28.8247	2.0664
Gaussian	5	300	10	0.7798	0.7447	0.035	0.0079	-0.2886	1.0684	1.0382
Bimodal	5	300	10	0.5044	0.2407	0.2637	0.0157	0.5178	0.0134	0.3983
Outlier	5	300	10	28.5316	29.1114	0.5798	0.3004	0.3883	28.1433	0.5953
Gaussian	6	300	10	0.7466	0.733	0.0136	0.0059	0.7886	0.042	0.4217
Bimodal	6	300	10	0.3718	0.2247	0.1471	0.0229	0.6402	0.2684	1.2259
Outlier	6	300	10	22.3292	22.4378	0.1087	0.1936	-0.6549	22.9841	1.8473
Gaussian	7	300	10	0.7222	0.7211	0.0011	0.0046	0.7743	0.0522	1.0044
Bimodal	7	300	10	0.231	-0.0167	0.2477	0.0511	0.0615	0.1695	0.162
Outlier	7	300	10	24.0836	24.3911	0.3075	0.4077	-1.0098	25.0934	5.1457
Gaussian	8	300	10	0.7539	0.7426	0.0113	0.0026	0.3628	0.3911	0.5768
Bimodal	8	300	10	0.3744	1.7532	1.3789	0.6674	0.2299	0.1445	0.4971
Outlier	8	300	10	22.3944	21.7204	0.674	0.4093	0.7387	21.6557	0.9665
Gaussian	9	300	10	0.7071	0.6962	0.0109	0.0071	0.3885	0.3186	0.652
Bimodal	9	300	10	0.338	0.1552	0.1828	0.0216	0.3709	0.0329	0.4557
Outlier	9	300	10	27.8793	28.7998	0.9205	0.4002	-0.7135	28.5928	6.1701
Gaussian	0	400	10	0.864	0.8995	0.0355	0.0078	1.8043	0.9402	1.8069
Bimodal	0	400	10	0.4984	0.6221	0.1237	0.023	0.8755	0.3771	0.5451
Outlier	0	400	10	26.6928	27.6059	0.9131	0.6892	1.0813	25.6115	3.5976
Gaussian	1	400	10	0.7712	0.7646	0.0067	0.0039	0.8401	0.0689	1.8102
Bimodal	1	400	10	0.3075	0.1715	0.136	0.0248	0.1472	0.1603	0.5518
Outlier	1	400	10	24.4567	23.9633	0.4934	0.3105	2.7366	21.7202	4.4892
Gaussian	2	400	10	0.6345	0.674	0.0396	0.0083	0.6011	0.0334	0.652
Bimodal	2	400	10	0.3543	0.1822	0.1721	0.0085	0.5925	0.2382	0.7016
Outlier	2	400	10	31.4429	30.7398	0.7031	0.5208	1.1358	30.3071	3.5892
Gaussian	3	400	10	0.5676	0.5714	0.0038	0.0032	-0.227	0.7947	0.7935
Bimodal	3	400	10	0.4739	0.3095	0.1644	0.0178	0.1015	0.3724	0.3764
Outlier	3	400	10	26.1117	26.1403	0.0285	0.1048	-0.2865	26.3982	3.5958
Gaussian	4	400	10	0.6759	0.6589	0.0169	0.0082	0.818	0.1421	0.5125
Bimodal	4	400	10	0.3372	0.2291	0.1081	0.0064	0.2166	0.1206	0.3255
Outlier	4	400	10	22.7548	22.6919	0.0629	0.1188	0.1206	22.6342	1.1472
Gaussian	5	400	10	0.7112	0.7306	0.0194	0.0015	0.3849	0.3263	0.2921
Bimodal	5	400	10	0.4255	0.1535	0.272	0.0143	0.2255	0.2001	0.8181
Outlier	5	400	10	24.3651	24.9046	0.5396	0.0727	0.2196	24.1454	1.0261
Gaussian	6	400	10	0.6929	0.7411	0.0482	0.0073	0.6103	0.0826	0.8007
Bimodal	6	400	10	0.4054	0.2448	0.1606	0.005	0.2219	0.1835	0.9239
Outlier	6	400	10	22.5281	22.049	0.4791	0.1757	-1.4547	23.9828	2.6871
Gaussian	7	400	10	0.785	0.7763	0.0088	0.0058	0.4271	0.3579	0.25
Bimodal	7	400	10	0.3722	0.2407	0.1315	0.0025	0.2674	0.1048	0.1249
Outlier	7	400	10	26.1779	26.1939	0.016	0.0756	-0.8992	27.0771	1.9071
Gaussian	8	400	10	0.659	0.6803	0.0213	0.0111	0.5193	0.1397	0.3531
Bimodal	8	400	10	0.5097	0.3521	0.1576	0.0085	0.7256	0.2159	0.4691
Outlier	8	400	10	22.6093	21.9028	0.7065	0.1741	-0.3412	22.9505	1.3742
Gaussian	9	400	10	0.6556	0.6539	0.0017	0.0093	1.083	0.4274	0.7172
Bimodal	9	400	10	0.3941	0.4562	0.0622	0.006	3.9006	3.5066	9.9625

DataType	i	points	runs	trueBeta	DPLTSBetaAvg	DPLTSBetaAvgError	DPLTSBetaStd	NSBetaAvg	NSBetaAvgError	NSStd
Outlier	9	400	10	23.9783	24.414	0.4357	0.314	0.6347	23.3436	0.9923
Gaussian	0	500	10	0.6379	0.6359	0.002	0.0013	1.7399	1.102	2.2402
Bimodal	0	500	10	0.4037	0.2617	0.142	0	0.2732	0.1305	1.2611
Outlier	0	500	10	23.1721	23.3258	0.1537	0.154	-2.4361	25.6082	7.4385
Gaussian	1	500	10	0.6342	0.6765	0.0424	0.0044	0.6675	0.0334	0.264
Bimodal	1	500	10	0.4123	0.1869	0.2254	0.0052	0.2247	0.1876	0.331
Outlier	1	500	10	20.7306	21.6486	0.918	0.0293	-0.0888	20.8193	0.6384
Gaussian	2	500	10	0.6015	0.5854	0.0161	0.0032	0.2976	0.3038	0.9446
Bimodal	2	500	10	0.4359	0.5324	0.0964	0.0057	-0.2575	0.6935	0.9583
Outlier	2	500	10	22.6088	24.1448	1.536	0.2811	0.1212	22.4876	0.4802
Gaussian	3	500	10	0.6567	0.6195	0.0373	0.0061	0.3121	0.3446	0.4189
Bimodal	3	500	10	0.4973	0.7525	0.2551	0.0734	0.0498	0.4476	0.7349
Outlier	3	500	10	24.7585	25.3199	0.5615	0.1126	0.4354	24.323	2.5907
Gaussian	4	500	10	0.6482	0.6168	0.0313	0.0145	0.5094	0.1387	0.6724
Bimodal	4	500	10	0.5217	0.6389	0.1172	0	0.7784	0.2567	0.592
Outlier	4	500	10	25.4433	25.9816	0.5382	0.2321	0.4948	24.9485	0.973
Gaussian	5	500	10	0.7605	0.7781	0.0176	0.0069	0.9526	0.192	1.0155
Bimodal	5	500	10	0.4965	0.3624	0.1341	0.0029	1.0812	0.5847	2.4238
Outlier	5	500	10	26.2621	27.5384	1.2763	0.2411	5.6961	20.566	3.2247
Gaussian	6	500	10	0.5883	0.6153	0.0271	0.007	0.802	0.2138	0.3473
Bimodal	6	500	10	0.399	0.223	0.176	0.0103	-0.5611	0.9601	1.6788
Outlier	6	500	10	26.3815	26.1473	0.2341	0.0954	0.3015	26.0799	2.168
Gaussian	7	500	10	0.9088	0.9056	0.0032	0.0017	0.3124	0.5964	0.881
Bimodal	7	500	10	0.3337	0.1837	0.1499	0	0.6934	0.3598	1.2072
Outlier	7	500	10	23.3006	23.549	0.2484	0.0493	-0.588	23.8885	2.2974
Gaussian	8	500	10	0.6653	0.6614	0.0039	0.0044	1.6522	0.9869	3.1387
Bimodal	8	500	10	0.484	0.5953	0.1113	0.0211	0.2643	0.2198	0.8689
Outlier	8	500	10	23.9752	22.4536	1.5216	0.0466	-0.3786	24.3538	1.53
Gaussian	9	500	10	0.6775	0.6884	0.0109	0.0035	0.6569	0.0205	0.6987
Bimodal	9	500	10	0.4003	0.2151	0.1852	0.007	-0.0101	0.4105	0.4278
Outlier	9	500	10	24.2684	24.8	0.5316	0.2234	-1.0899	25.3583	4.605