Data Streaming Methods for Linear Regression Inference Testing

Colin Smith

Abstract

We investigate the problem of performing linear regression inference tests on large data streams. An improvement upon traditional methods is required due to the volume of data involved. We explore three methods to perform a linear regression inference test, in ways that mitigate the problems that arise from working with Big Data. Our first method treats the data stream as any other data set. This naïve method calculates exact values for the $\hat{\beta}$ coefficients in the linear model, and gives the most accurate results in the inference test. Our second method uses sampling to conduct an inference test that achieves an effect size provided by the user. Our third method computes inference tests and R^2 values in a sliding window, in case the slope coefficient changes over time. With these three solutions in hand, we give a user numerous options to effectively perform linear regression inference on large data streams. Our code is publicly available, and we include numerous experiments to demonstrate its application. We also include a word on inference for multivariable linear regression models.

1 Introduction

Over the course of the past century the volume of data produced every day has increased exponentially. With this increase it has become infeasible to store all of it at once. In the face of this problem, new techniques for analyzing these data must be created. One such technique is the data streaming model. Streaming algorithms are used for a number of applications from large scale data analysis to database management and optimization [1][2][3][4]. Over the past decade, much research has been conducted in this field of study, but statistical testing over data streams has been only a very small subsection of this field.

Statistical inference is one of the pillars of statistics. It is one of the major methods by which we analyze data to verify and improve our models about the real world. Linear regression inference tests are among the most common statistical inference tests. A linear regression inference test asks if the slope, β_1, of our regression line $y = \beta_1 \cdot x + \beta_0$ is equal or not equal to zero. Given a linear model, whether this slope is equal to zero or not is equivalent to the question of whether or not the variables x and y are correlated. Because we do not know the true population we must use estimators $\hat{\beta}_1$ and $\hat{\beta}_0$ to figure out if the true β's within some reasonable error.

In the past, statisticians have had trouble analyzing data sets of extreme size by traditional methods [5]. This paper consists predominantly of methods for performing a linear regression inference tests on very large data streams. A data stream is a series of data points that, in the most interesting applications, is too long to store on the computer processing the data [7]. Either a data center holds all of the data or the data is being created in real time. This means that either only a substantially smaller subset of the data stream must be looked at, or the analysis must occur in real time and no data is stored. For all methods in this paper, the stream is a sequence of (x,y) data pairs.

Data streams are the setting for algorithms in the field of Big Data. As Big Data becomes more important in every day life we recognize that it is important to discover efficient methods to process

and analyze these streams, in order to make use of all of the data we gather. As with any field of technological development, there are many problems that arise in the setting of Big Data, which must be overcome in order to use the advancements Big Data offers. In this paper, we find several effective algorithms to run linear regression inference tests over data streams, which mitigate such problems. We then examine the pros and cons of each method for various purposes in the analysis of data streams.

In computer science, it is standard to analyze algorithms in terms of time and space complexity. As this pertains to streaming algorithms, we note that linear space is impossible because the entirety of the stream cannot be stored, so if space were to increase in proportion to the size of the stream, our algorithm would not be viable. Likewise, time complexity must be near linear, as looking at all of the data is a very expensive operation. Therefore, we find algorithms that run a number of operations proportional to only the size of the stream.

In Section 3.1, we analyze a simple, effective approach to the problem of running linear regression inference tests over data streams. This method is able to find the exact values of $\hat{\beta}_0$, $\hat{\beta}_1$, and $S_{\hat{\beta}_1}$, which allow us to perform a linear regression inference test, with only one pass over all of the data. This solution runs in linear time and requires logarithmic memory with respect to the length of the data stream. We examine the problems with this approach in Section 3.2 and offer a solution to that problem using uniform random sampling in Section 3.3. In Section 4, we examine a different solution using a sliding window, which has a looser set of required assumptions. We discuss the potential problems with this solution in Section 4.2, and how we counteract these problems in Section 4.3. Section 5 documents a synopsis of the testing we did to show our algorithm's effectiveness. Code for all algorithms detailed in this paper can be found on GitHub at `https://github.com/cmsmitty441/LinReg_Stream`.

2 Related Works

Linear regression is a powerful and well-studied tool for statistical analysis. While inference testing of linear regressions has not been explored before in streaming context, linear regression on its own has been. The general case for linear regression involves not just a pair of variables, dependent and independent, but has an arbitrary number of variables. A streaming algorithm to fit an ordinary least squares linear regression model has been done in [8]. While this solution is effective for finding linear regression coefficients $\hat{\beta}_i$ in a stream, it does not suit our purposes, because the sampling method used in [8] inflates the standard error in the $\hat{\beta}$'s, and hence leads to incorrect confidence intervals and p-values. Briefly, the algorithm in [8] creates a sketch (i.e. a sample from the stream) chosen in a way to bias in favor of large leverage points, and hence increases variability. Mathematical details on leverage and its connection to variance may be found in [9]. Streaming algorithms for other important statistical tools such as the χ^2 test [10] and the Kolmogorov-Smirnov test [11] have also been discovered.

Because linear regression is such a well-studied topic, there are a number of well-known and very sophisticated ways to calculate a number of important statistical measures. The most important of these is Welford's method. Welford's method allows us to calculate the variance and covariance without error in a cumulative fashion. Typically, this calculation would have to be done in two passes, as it normally requires a pre-calculated mean. But, by the formulas given in [12], we can update the variance and covariance in only one pass over the data stream, which allows linear regression to be exactly calculated in a very efficient manner.

In order to do our statistical inference test, we include a fast algorithm to compute p-values for t-distributions and normal distributions. Luckily, this is a well-researched topic, and our algorithm makes use of the relationship between the gamma function and these distributions [13]. In order to use the normal distribution for our p-value computation, in Section 3, we assume the classic linear regression assumptions [14]:

1. the data represents a simple random sample (of size n) from some unknown population (in particular, pairs of data points (x_i, y_i) and (x_j, y_j) are independent),

2. the data exhibits a linear relationship $y = \beta_0 + \beta_1 x + \epsilon$, for fixed constants β_0, β_1, and an error term ϵ,

3. homoskedasticity, i.e. the standard deviation of the error term ϵ does not depend on x, and

4. the error terms are distributed according to a normal distribution with mean 0 and standard deviation σ_ϵ.

We note that the normality assumption is not strictly necessary for our results, since the Central Limit Theorem guarantees our slope estimate $\hat{\beta}_1$ will be approximately normally distributed for large n [14].

In Section 4.1, we relax the assumption that β_0 and β_1 are constant over the entire stream, and we allow for the possibility that these quantities are changing over time. This means that for certain windows of time, we can have a statistically significant relationship between x and y, while at other times the relationship can fail to be statistically significant (i.e. $\beta_1 = 0$).

3 Testing Over an Entire Data Stream

In this section we provide an algorithm which utilizes all of the data in the data stream to calculate the linear regression coefficients, $\hat{\beta}_0$ and $\hat{\beta}_1$, as well as the standard error in $\hat{\beta}_1$, $S_{\hat{\beta}_1}$. We then use these values to calculate a p-value with which we can reject or fail to reject the null hypothesis, H_0. The data stream represents the totality of our simple random sample for the purposes of statistics. This has a few drawbacks related to the volume of data and how it affects our interpretation of the statistics. The first of these solutions has been dubbed the basic approach, wherein each data point in the data stream updates our relevant variables. The second method is to take a random sample as an estimate of the data. This solves a number of the problems from the basic solution.

3.1 The Basic Approach

The most basic solution is simply to look at each data point as it streams in, and use an update algorithm to update all of the values that we need to keep track of in order to compute the $\hat{\beta}_1$ and its standard error. Because our stream has no limit to its size, an algorithm for these values that has to look at stream elements multiple times is not particularly useful.

The values we are concerned with for calculating the beta coefficients are S_x, the variance in the dependent variable x, and S_{xy}, the covariance in the dependent and independent variables x and y. In order to calculate these values we will need running variables for the mean values of x and y as well as a count of the number of items which have passed in the data stream. In order to calculate the standard error in the beta coefficients we also have to keep track of the mean of the values x $\cdot$ y and y^2. The following recursive formula keeps a running mean for the sequence $\{q_1, q_2, ..., q_n\}$; to begin the iteration $\bar{q}_0 = 0$. We represent the mean of all values of q up to the n^{th} value as $\bar{q}_n$.

$$\bar{q}_n = \bar{q}_{n-1} + \frac{1}{n}(q_n - \bar{q}_{n-1}) \tag{1}$$

We use this to keep mean values for x, y, $x \cdot y$, and y^2, by setting q_n equal to the values x_n, y_n, $x_n \cdot y_n$, and y_n^2 respectively for all such values.

The following two equations are used to keep a running value for the variance in the dataset $\{x_1, ..., x_n\}$, $S_{x,n}$, and the covariance in the dataset $\{(x_1, y_1), ..., (x_n, y_n)\}$, which we dentote $S_{xy,n}$

[6][12].

$$S_{x,n} = \frac{1}{n}((x - \bar{x}_{n-1}) \cdot (x - \bar{x}_n) - S_{x,n-1}) \tag{2}$$

$$S_{xy,n} = \frac{1}{n}((x - \bar{x}_{n-1}) \cdot (y - \bar{y}_n) - S_{xy,n-1}) \tag{3}$$

The algorithm to update these values is shown in Algorithm 1 (below).

Algorithm 1: The Update Algorithm

```
INPUT: the data stream
set count, meanX, meanY, meanXY, MeanY^2, variance, and covariance to 0;
for each (x,y) in stream do
    count++;
    M_{x,k-1} = meanX;
    meanX += (x - meanX)/count;
    M_{y,k-1} = meanY;
    meanY += (y - meanY)/count;
    variance += ((x - M_{x,k-1})·(x - meanX) - variance)/count;
    covariance += ((x - M_{x,k-1})·(y - meanY) - covariance)/count;
    meanXY += (x·y - meanXY)/count;
    meanY^2 += (y^2 - meanY^2)/count;
end
OUTPUT: count, meanX, meanY, meanXY, MeanY^2, variance, and covariance
```

Once we have run this algorithm over the data stream we determine our $\hat{\beta}_1$ by the typical formula: $\hat{\beta}_1 = \frac{covariance(x,y)}{variance(x)}$ and $\hat{\beta}_0$ with $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$ [15]. We can then find the standard error of $\hat{\beta}_1$, $S_{\hat{\beta}_1}$, using the following formulas [15]:

$$S_\epsilon^2 = \frac{1}{n-2}\left[\sum_{i=1}^{n} y_i^2 - \hat{\beta}_0 \sum_{i=1}^{n} y_i - \hat{\beta}_1 \sum_{i=1}^{n} x_i y_i\right] \tag{4}$$

$$S_{\hat{\beta}_1} = S_\epsilon \sqrt{\frac{n}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}} \tag{5}$$

Through a short bit of manipulation we can convert the unwieldy equations above into more useful forms. First we distribute $\frac{n}{n}$ through our formula for S_ϵ^2.

$$S_\epsilon^2 = \frac{n}{n-2}\left[\overline{y^2} - \hat{\beta}_0\overline{y} - \hat{\beta}_1\overline{xy}\right] \tag{6}$$

Then, by the definition of variance, σ^2, we can see the following [15]:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n} \tag{7}$$

$$= \frac{\sum_{i=1}^{n} x_i^2}{n} - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n^2} \tag{8}$$

$$= \frac{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}{n^2} \tag{9}$$

Therefore we can rewrite the formula for $S_{\hat{\beta}_1}$:

$$S_{\hat{\beta}_1} = S_\epsilon \sqrt{\frac{1}{n\sigma^2}} \tag{10}$$

Because we have our $S_{\hat{\beta}_1}$ in terms of variables that we have kept running values of, we know we can calculate it after our update algorithm has been run on all items in the data stream. We use the quantities $\hat{\beta}_1$ and $S_{\hat{\beta}_1}$ to conduct our hypothesis test. Recall that the null hypothesis, H_0, assumes $\beta_1 = 0$, while the alternative hypothesis is that $\beta_1 \neq 0$. Because we are dealing with a huge number of data points, the Central Limit Theorem implies that the statistic $\hat{\beta}_1$ is approximately normally distributed with mean β_1 and standard deviation $S_{\hat{\beta}_1}$. Thus, we can use a normal distribution to determine our z-score and p-value. The z-score is $\hat{\beta}_1 / S_{\hat{\beta}_1}$ and the p-value is calculated using a p-value calculator. This is the main output of our algorithm: using the p-value we can reject or fail to reject the null hypothesis H_0. The output is 1 if we rejected and 0 if we failed to reject.

This algorithm suffers from having to look at every item in the stream, which for extremely large data streams is a problem. Even though the update algorithm can run extremely quickly, with so many data points, it will still take an unfeasible amount of time to run through the largest of data sets. The algorithm does benefit, however, from having constant memory for almost all of it's variables.

It is standard to report the asymptotic time complexity (resp. space complexity) of an algorithm using Big O notation. We remind the reader that this notation represents the asymptotic number of operations (resp. bits)[16]. This means that as we increase the size of the input, the behavior of the algorithm approximates the form given in the notation. For example, counting may require $O(log_2(n))$ bits because we can use n bits to represent 2^n unique numbers. A more formal definition for this notation can be found on page 36 of [16].

Because of our assumption that the population has a true β and σ^2, which do not grow with the size of the stream, the only variable that is dependent on the size of the data stream is the count. This value requires $O(log(n))$ bits where n is the size of the stream. We can use fewer bits by using approximate counting, but this will introduce an error into the variables as the count will not be 100% accurate. An example of approximate counting would be to use Morris counting, which can keep an approximate count using only $O(log(log(n)))$ bits [17].

3.2 The Effect Size Problem

The aforementioned solution has another major flaw. As the size of the data stream grows ever larger, the effect size of the inference test grows ever smaller. We can see this from the equation for effect size:

$$E = z_\alpha \cdot \frac{S}{\sqrt{n}} \tag{11}$$

where z_α is the z value for a given confidence level (often 1.96 in statistics), S is the standard deviation of the sample (in this case, $S = S_{\hat{\beta}_1}$), and n is the sample size.

This means that given enough data we will reject the null hypothesis under almost any circumstances. So, with these astronomical volumes of data we need to make a determination between what is statistically significant and what is practically significant. Imagine the case that you are testing the correlation between weight and salary in a large population. In your model, you may expect that the correlation coefficient is such that a person makes \$100 less for each pound heavier they are, so $\hat{\beta} = -100$. But, perhaps the true value of β is -\$100.001. This is a difference of less than a penny, so your estimate of \$100 is certainly good enough; however, given a large enough sample size we expect the sample estimate to be extremely close to the true value, -\$100.001. For

large n then, the effect size becomes extremely small as z_α is constant and S does not increase with the sample size. So, while the user believes that being less than a penny off is not a significant deviation from the model, with a large enough sample size, we will reject a null hypothesis on the basis of less than a single penny!

3.3 Uniform Random Sampling

To mitigate the effect size problem, we can take a random sample of the data stream instead of looking at the whole thing. In order to find the sample size, we ask the user for the desired effect size. This allows the user to determine the cutoff for what they find to be practically significant. Our test then rejects the null hypothesis *if and only if* the effect is practically significant, according to the user.

The z value, z_α, is calculated from the significance level, α, using an inverse normal distribution calculator. We estimate $S_{\hat{\beta}_1}$ from the first 50 data points in the stream. This value will be recalculated after our sample, but due to our assumption of homoskedasticity, it is appropriate to use the first 50 data points for an estimation of $S_{\hat{\beta}_1}$ as we expect the value of $S_{\hat{\beta}_1}$ not to change greatly over time. We calculate the desired sample size using this formula:

$$n = max\left\{\left(\frac{z_\alpha \cdot S_{\hat{\beta}_1}}{E}\right)^2, 30\right\} \tag{12}$$

This gives us the sample size such that the test will reject the null hypothesis if and only if the detected effect size is larger than E. We force the value n to be ≥ 30 because the Central Limit Theorem's guarantee of normality may not hold below that value, which means that we could not make our assumption of normality. We anticipate that such a situation would almost never arise in practice.

In order to get an unbiased estimator of $S_{\hat{\beta}_1}$, we use reservoir sampling [18]. This method of sampling works for uniformly randomly sampling streams of unknown size, which is what we need as we often will not have an exact count of items in the stream. We do note that reservoir sampling does take $O(nm - m^2)$ time for a sample size of m. Once the sample is done our algorithm from Section 3.1 can run through the sample in $O(m)$ time.

It is not often that statisticians have too much data. Generally in statistics we have a sample of some population. But, in this case because of the sheer volume of data we are forced to take a sample of a sample. While our method is effective, this down-sampling loses a lot of information. The amount of information discarded is exactly the difference between a result that is practically significant and one that is statistically significant (but not practically significant). As long as the value of β_1 is constant over the whole stream, our method (of uniform sampling and then testing the hypothesis) will yield a correct result, with the usual caveat that Type 1 and Type 2 errors are possible [14].

If the value of β_1 does change over time, for instance, we would see changes in the value of β_1 as it streams in. This would be a major problem for random sampling as every collection time would be equally likely to be sampled and different times may be mixed together. This sampling also must be used on a data set that is not generating data in real time. The way that this form of the algorithm works is that it takes the sample while the stream comes in and then, once it has the sample, it runs the linear regression inference test on that sample. The time to run a linear regression on this sample is negligible as the sample size is much less than the stream size. In the next section we look at a way to deal with more dynamic data sets and avoid losing any valuable information.

4 Using a Sliding Window

A solution to the effect size problem and to the dynamic data set problem is to use a sliding window. A sliding window looks at a small continuous piece of the data stream, and as new data streams in, old data is dropped from the window. Instead of looking at the whole data stream and outputting a single boolean response, we now output a stream of boolean responses. This stream tells us at any given time, based on the current window, if we reject or fail to reject the null hypothesis.

While this could be done by running traditional linear regression algorithms over the window, we found significantly more efficient method of calculation. We keep certain values of running sums in the window and modify our update equations from the previous section to work in the context of the sliding window. This running sum is dealt with by keeping a list of each item in the window and each time a new point streams in we subtract out the oldest item, in a sense we shift the window along the data stream. This "shift" method takes the form:

$$Shift \equiv q_f \leftarrow q_f - f(s_0) + f(s) \tag{13}$$

where q_f is a value getting updated, f is a function on a group of data, s is new data, s_0 is old data. For example, if $f((x, y)) = xy$, q_{xy} would be the sum of the values of xy for all x and y in the window.

We use this to update our values for all relevant functions f, keeping previous values in a queue so that we can always have s_0. The queue uses two functions, push() and pop(). The push() function adds a value at the end of the queue. The pop() function removes an item from the front of the queue and returns that value. When we calculate the p-value, we then divide by the window size to get our means, variance, etc. Because we only keep a fixed number of data points and variables, the space complexity of this algorithm is $O(k)$ memory for an instance with a window

size of k.

Algorithm 2: Sliding Window Algorithm

INPUT: α = desired significance value
run Algorithm 1 until count equals your desired window size, keeping a queue Q of each (x, y, $M_{x,k-1}$, meanX, and meanY) tuple;
(sumX, sumY, sumXY, sumY^2, sumV, sumC) = (meanX, meanY, meanXY, MeanY^2, variance, and covariance) $\cdot$ count;
windowSize = count;
for *each remaining (x,y) in stream* **do**
  (x_0, y_0, lmeanX_0, meanX_0, meanY_0) = Q.pop();
  lmeanX = sumX/windowSize;
  Run *Shift* on the values sumX, sumY, sumXY, and sumY^2 with f = x, y, x$\cdot$y, and y^2 respectively.
  (meanX, meanY) = (sumX, sumY)/windowSize;
  Run Shift on sumV and sumC with f = (x - lmeanX)$\cdot$(x - meanX) and (x - lmeanX)$\cdot$(y - meanY) respectively.
  Q.push((x, y, lmeanX, meanX, meanY));
  $\hat{\beta}_1$ = sumC/sumV;
  $\hat{\beta}_0$ = meanY - $\hat{\beta}_1\cdot$ meanX;
  $S_{\hat{\beta}_1} = \left(\text{windowSize}\cdot(\text{sumY}^2 - \hat{\beta}_0\cdot \text{sumY} - \hat{\beta}_1\cdot \text{sumXY})\cdot\frac{1}{sumV}\right)^{\frac{1}{2}}$;
  calculate p-value with $z = \hat{\beta}_1/S_{\hat{\beta}_1}$;
  OUTPUT True if p-value $< \alpha$ else False;
  each time a unique window is reached, we use the current $S_{\hat{\beta}_1}$ to update our window size using the formula expressed in section 3.3, if $n < 30$ then set n to 30;
end

4.1 Finding Window Size

In order to find the window size, we ask the user for the desired effect size and use the methods described in Section 3.3. Even though we are not as stringent with our assumptions as before, we still use the method of calculating $S_{\hat{\beta}_1}$ with the first 50 data points, as we still assume the data to be locally homoskedastic, i.e. the data will have large regions that are homoskedastic, but may change over long periods of time. We also re-calculate the window periodically, to make sure that our effect size stays where the user wants it, as we can no longer assume that $S_{\hat{\beta}_1}$ will remain constant over the whole stream. As such, we also must warn the user if the window size would fall below 30. We cannot let n fall below 30, however, as it would prevent us from using our assumptions about normality of $\hat{\beta}_1$. While we have loosened our restraints with some of the assumptions we made in the previous sections, normality must remain if we are to have a useful linear regression inference test. We note, however, that it would be simple to appeal to a t-test, with $n-1$ degrees of freedom, to find p-values, even if $n < 30$ did occur.

4.2 The Type I Error Problem

A problem arises with this method concerning type I error. In the case of many independent tests on data where there is truly no correlation, we would expect to see false positives proportional to our significance level (often 5% or $\alpha = 0.05$ in statistics). This is due to the nature of statistical inference. Because we are testing to see if the probability of the data occurring naturally falls below our significance value, the chance of a false positive is equal to the significance value by definition of significance value. This is a problem for us, because over a large stream we are conducting many different statistical tests, and because in the worst case, we can consider non-overlapping windows

to be independent tests. This means that the number of false positives is proportional to the size of our data stream, which is unbounded. There is a solution for taking multiple independent inference tests called the Bonferroni correction [15]. This correction, however, is to divide the significance value by the number of independent tests. Because our number of tests is unbounded, this correction would yield arbitrarily small p-values, so we cannot use the Bonferroni correction.

4.3 Output Stream Analysis

In order to remedy the type I error problem, we have designed a solution wherein we can analyze the output stream and determine if it is likely that the positives we return are true positives. We know that the frequency of type I errors we expect is equal to our significance level. For this example say α is 0.05. We would expect about 5% of our results to be false positives. Therefore we keep a percentage of the positive values that are output by the stream. This way, if the percentage of positives is near the significance level we can warn the user of the fact that we are returning positives with about the frequency of expected false positives. But, if the percentage of positives is much greater than 5% we would tell the user that the number of positives far exceeds the expected number of false positives. This does require an addition to our space complexity. The best way in which we can keep this running percentage is by using equation (1) to keep a running average of the bits. This average is the percentage of 1's that have appeared in the output stream. So, if this running percentage is too close to the significance level, we warn the user that the times the output stream rejects H_0 may be false positives. This requires us to have a count of our outputs which, as we discussed earlier, requires $O(log(n))$ bits using a counter.

We also are able to calculate the R^2 value in this sliding window format by just adding a few variables to be stored in the queue. The definition of R^2 is as follows [15]:

$$R^2 = \frac{SSR}{SST} \tag{14}$$

where SSR is the variability in y that is explained by the model, and SST is the total variability in y. Another value which is related to these two is the value SSE which is the difference between SST and SSR . These have the following definitions [15]:

$$SSR = \sum(\hat{y}_i - \bar{y})^2 \tag{15}$$

$$SSE = \sum(y_i - \hat{y}_i) \tag{16}$$

$$SST = SSR + SSE \tag{17}$$

$$= \sum(y_i - \bar{y})^2 \tag{18}$$

Our linear regression model says $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 \cdot x_i$, so once we make that replacement in [16] we can keep track of SSR and SST in sliding window fashion. Because both SSR and SST are sums, we use our *Shift* operation - equation (13) - with the functions as

$$f = (\hat{y}_i - \bar{y})^2 \ for \ SSR \tag{19}$$

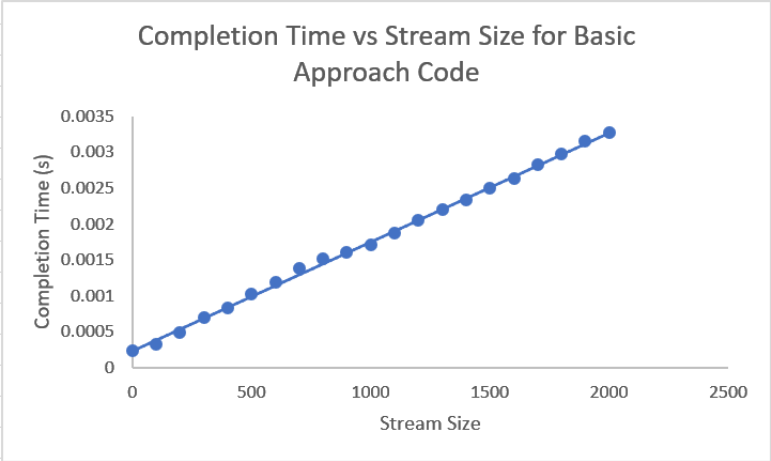
$$f = (y_i - \bar{y})^2 \ for \ SST \tag{20}$$

R^2 is a value which we can output in order to give the user information that is not affected by the type I error problem. Specifically, R^2 represents the percentage of the variability in y that is

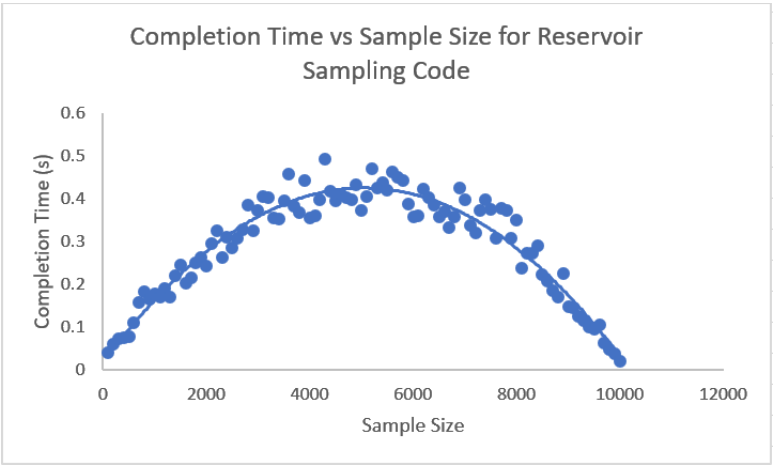
explained by x, and is another way to measure the association between x and y [15]. Users can use this value to discover false positives or false negatives yielded by the inference test. When there is a very low R^2 value, but the inference test rejects H_0, we can expect that this is a false positive. A similar analysis is made for high R^2 values being used to find false negatives.

5 Experiments

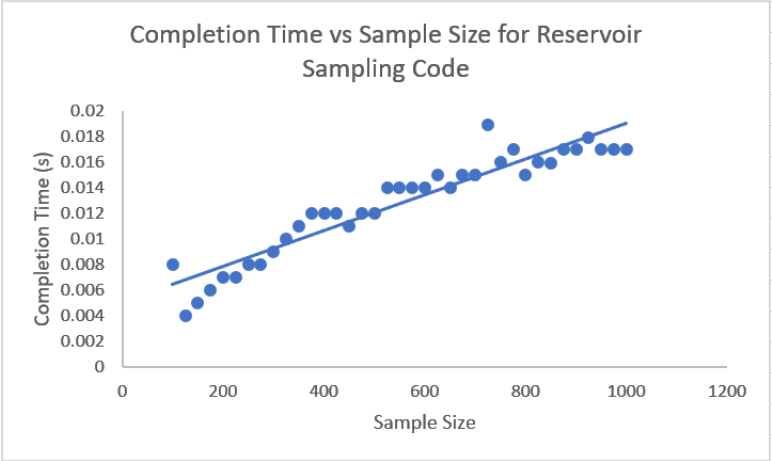
In order to see how well our algorithms work, we ran a number of tests on the various methods discussed in Sections 3.1, 3.3, and 4. For these tests we used data generated by a linear model with normally distributed residuals. In the first graph, we show that the basic approach discussed in Section 3.1, runs in linear time with respect to the stream size.



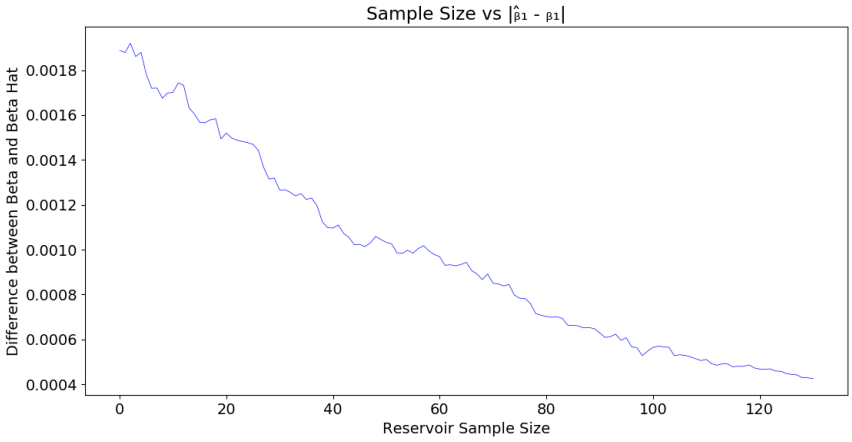
Likewise, we show that for the reservoir sampling method, the time varies as $O(nm - m^2)$ with the sample size m. For this the steam size was held constant ($n = 10000$ data points) and the sample size was varied between $m = 100$ and $m = 10000$.



We note that for the case of $m << n$, $O(nm - m^2)$ is approximately $O(nm)$, and for our purposes in streaming $m << n$ will always be the case. For the case where we look at $100 \leq m \leq 1000$ we get the following graph:

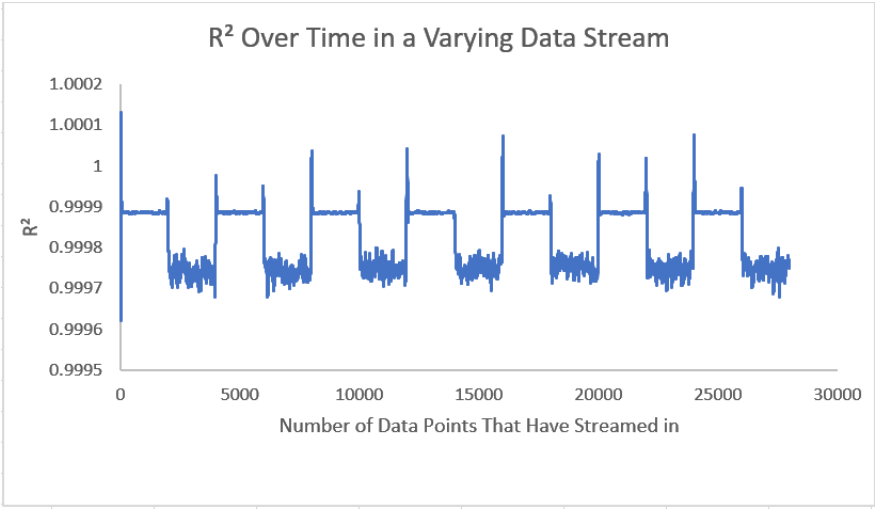


which show a near linear relation between sample size and completion time. Next, we tested our uniform random sampling method and found the difference between the true value of β for our generated data and the $\hat{\beta}$ found by the method.



It seems natural that the difference between $\hat{\beta}_1$ and the true β_1 decreases as the sample size goes up.

In the following graph, we look at data where the relation between x and y changes over time.



These data are simulated such that every 2000 data points it switches between there being a strong correlation between x and y and no correlation between x and y. As you can see the R^2 value is close to 1 while the data are correlated, and further from 1 while they are not.

6 Multivariable Linear Regression

We conclude with a brief discussion of how the problem of linear regression inference testing can be extended to more than two variables, the case of multivariable regression. In general, a linear regression inference test can be performed on a model involving any number of independent variables. In this case the linear model is not $y = \beta_0 + \beta_1 \cdot x$, but rather $y = \beta_0 + \beta_1 \cdot x_1 + ... + \beta_k \cdot x_k + \epsilon$. Because the equations for solving for the $\hat{\beta}$ coefficients become increasingly more complex as k grows, matrix computation is employed to find $\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_k$.

In the context of multivariable linear regression, there are two types of statistical inference tests, that are discussed in [15]. One tests the null hypothesis $H_0 : \beta_1 = \beta_2 = \cdots = \beta_k = 0$, with the alternative hypothesis, H_a, that some β_i is not equal to zero. This test determines if the entire model is useful. The test statistic for this test is F which is defined as follows:

$$F = \frac{\frac{SST}{k-1}}{\frac{SSE}{n-k}} \tag{21}$$

Assuming the errors ϵ are normally distributed (which we can assume for sufficiently large sample size), this test statistic is distributed according to an F-distribution with degrees of freedom $k - 1, n - k$. The other inference test for multivariable linear regression tests the null hypothesis $H_0 : \beta_i = 0$, with an alternative hypothesis $H_a : \beta_i \neq 0$. This test determines if x_i is correlated with y. The test statistic is z, defined as follows.

$$z = \hat{\beta}_i / S_{\hat{\beta}_i} \tag{22}$$

The z-statistic is normally distributed so long as the sample size is sufficiently large. Once these test statistics are computed, a p-value calculator [13] can be used to determine p-values associated with the test statistics. These p-values can be used to reject or fail to reject H_0.

A method mentioned in [8] can be used to estimate the $\hat{\beta}$ coefficients for the multivariable linear regression model, and requires $O(n \cdot k \cdot log(n))$ time for a stream of size n. While this method accurately estimates the $\hat{\beta}$ coefficients, it is not usable for our purposes as the sampling involved has a bias toward points in the data with more leverage. This increases the variability in the $\hat{\beta}$ coefficients, and therefore increases the value of $S_{\hat{\beta}_1}$. Because of how important $S_{\hat{\beta}_1}$ is to the inference test, we cannot use this sampling method.

Instead we look to an algorithm to based on the Givens Rotation [19], that can be used to solve for the $\hat{\beta}$ coefficients in $O(n^2 k)$, where n is the size of the data stream [19]. So, we can use uniform random sampling to get an unbiased estimate of $S_{\hat{\beta}_1}$ and then run the Givens Rotation based algorithm on that sample. For a sample of size m, this gives us a time complexity of $O(m^2 k)$, which works well as the sample size is defined by the user, not the stream.

The Givens Rotation based algorithm is what is known as an online algorithm. This type of algorithm takes in data one tuple, $(y, x_0, x_1, ..., x_k)$, at a time and updates its value accordingly. Because of this property, we can use the Givens Rotation method in the context of a sliding window by both adding in data and removing data. We would then have a sliding window method, that runs in $O(m^2 k)$ time per element of the stream, for a window size of m, giving a time complexity of $O(nm^2 k)$ time overall. Because m and k are constant, this is linear with respect to the stream size, meaning that it is effective for our purposes.

We also note that the space complexity for these algorithms is $O(mk)$, which is sublinear relative to the size of the stream. All methods mentioned use a type of sampling or a sliding window, and thus must store m data points for sample size m (in the sliding window case, m is the window size). The space complexity, however, is also dependent on the number of $\hat{\beta}$ coefficients, as each new $\hat{\beta}$ coefficient adds another column to the matrices used in these methods. Because of this, the space complexity for all of these methods is $O(mk)$. These general approaches work well for a linear regression inference test with any number of variables. But, in simple case (k = 1), the algorithms shown in Section 3 and 4 are the best approach, as they offer the same or better time and space complexity, without the computational expense and mathematical complexity of algorithms involving matrix manipulation.

7 Conclusion

Linear regression inference testing is an important piece of statistics, that presents a number of problems when scaled to the large sizes presented by data streams. Because the volume of data produced is ever growing, we must seek to bypass these problems using streaming techniques. We have developed three such solutions, that mitigate certain problems involved with running linear regression inference tests on large data sets. Each of these methods presents a solution for specific problems a statistician may face when performing linear regression inference. If an exact solution is of extreme importance, such that the effect size of the test can approach extremely small values, we present the basic approach (Section 3.1) as it gives an exact solution in a usable amount of time. In the case that the user is okay with using only a fraction of the available information in the interest of finding an approximate solution more quickly, the sampling method (Section 3.3) is the appropriate solution. Finally, in the case that the user cannot make the assumptions required of the first two solutions, the sliding window (Section 4) presents the opportunity to run a linear regression inference test which is more adaptable to data stream of pairs (x, y) where the linear regression slope β_1 varies over time.

A problem that presents itself as a natural extension upon this work is the discovery of an approximate solution for the case of multivariable linear regression can be done in a time similar to the $O(n \cdot k \cdot log(n))$ time found in [8], without negatively affecting $S_{\hat{\beta}_1}$. Such a solution is required to perform linear regression inference. We believe that this should be possible based on the methods used in [8], but as of this writing it has not yet been shown.

8 Acknowledgments

We thank the Anderson Scholarship fund for funding this research and Dr. Ashwin Lall of Denison University for his help in examining early drafts of the paper, as well as his insight and expertise in this field of research.

References

[1] A. Noga, Y. Matias, and M. Szegedy. "The Space Complexity of Approximating the Frequency Moments." *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* - STOC '96, 1996, doi:10.1145/237814.237823.

[2] Aggarwal, Charu C., and Chandan K. Reddy. Data Clustering: *Algorithms and Applications*. Chapman and Hall/CRC, 2014.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. *ACM PODS*, 2002, 1–16.

[4] A. Chakrabarti, K. D. Ba, and S. Muthukrishnan. "Estimating Entropy and Entropy Norm on Data Streams." *Internet Mathematics*, vol. 3, no. 1, 2006, pp. 63–78., doi:10.1080/15427951.2006.10129117.

[5] D. A. Nolan, and T. Speed. *Stat Labs: Mathematical Statistics through Applications*. Springer, 2001.

[6] D. E. Knuth, *The Art of Computer Programming Volume. 2: Seminumerical Algorithms*. Addison-Wesley, 1998.

[7] J. Leskovec, A. Rajaraman, and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[8] P. Drineas, M. Mahoney, and S. Muthukrishnan. "Sampling Algorithms for ℓ_2 Regression and Applications." Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm - SODA '06, 2006, doi:10.1145/1109557.1109682.

[9] J. Neter, M. Kutner, C. Nachtsheim, and W. Wasserman *Applied Linear Regression Models*. Third Edition, McGraw-Hill, 1996.

[10] E. Farrow, et al. "Accessible Streaming Algorithms for the Chi-Square Test." 2017. Denison University, student paper.

[11] A. Lall. "Data Streaming Algorithms for the Kolmogorov-Smirnov Test." 2015 IEEE International Conference on Big Data (Big Data), 2015, doi:10.1109/bigdata.2015.7363746.

[12] B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products." Technometrics, vol. 4, no. 3, 1962, pp. 419–420. JSTOR, www.jstor.org/stable/1266577.

[13] W. Press, and W. T. Vetterling. *Numerical Recipes*. Cambridge Univ. Press, 2007.

[14] R. V. Hogg and E. A. Tanis. *Probability and Statistical Inference*. Pearson, 2010.

[15] Akritas, Micheal G., *Probability & Statistics with R for Engineers and Scientists.* Pearson, 2016.

[16] J. Kleinberg, E. Tardos, *Algorithm Design*. Addison-Wesley, 2006.

[17] Morris, Robert. "Counting Large Numbers of Events in Small Registers." Communications of the ACM, vol. 21, no. 10, 1978, pp. 840–842., doi:10.1145/359619.359627.

[18] J. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software* 11:1, pp. 37–57, 1985.

[19] J. H. Maindonald, *Statistical Computation*. J. Wiley & Sons, 1984.