LinearLegions: A Linear Size Cardinality Estimator

ABSTRACT

Counting the number of unique items in a data set is of interest in many applications. For example, the owner of a web property, e.g., a video sharing website, a social media website, a search engine, etc., benefits from knowing the number of unique visitors to their site, the number of unique people that a certain advertisement was shown to, etc. This disclosure describes a practical cardinality estimator that uses $O(m + \log\log N)$ space and has a conjectured $O(1/\sqrt{m})$ relative error, where m is an accuracy parameter and N is the maximum cardinality that is to be reported. The cardinality estimator improves upon the best-known space bounds of prior cardinality estimators and matches on relative error.

KEYWORDS

- Cardinality estimation
- Liquid legions
- Hyperloglog
- Maximum likelihood estimation

BACKGROUND

Counting the number of unique items in a data set is of interest in many applications. For example, the owner of a web property, e.g., a video sharing website, a social media website, a search engine, etc., benefits from knowing the number of unique visitors to their site, the number of unique people that a certain advertisement was shown to, etc. Estimating the number of items in a set is known as cardinality estimation, and is of fundamental importance in big data applications. It is widely used in network routers as well.

In 1985, Flajolet and Martin gave a method for estimating the number of unique items in a data set. This method was suitable for use in applications where the size of the data is too large to fit into the memory of a single machine. Flajolet and Martin proposed a sketch data structure that summarized the items that had been encountered so far in a data stream. The sketch could then be used to answer the query, "How many unique items were in the set?" As a byproduct, the sketch also supported a union operation, allowing the cardinality of the union of two sets to be computed from the sketches of the constituent sets.

The size of the data structure was $O(m \log N)$, where m is the number of registers allocated to the sketch, and N is the maximum cardinality that is to be reported. The Flajolet-Martin sketch traded off size for accuracy. The accuracy of a sketch algorithm is described by the relative error of the estimates that it provides. If n is the true size of the set and $\hat{n}$ is the estimate given, then the relative error is defined to be the standard deviation of $\hat{n}/n$. The Flajolet-Martin sketch had a relative error of $O(1/\sqrt{m})$.

While the Flajolet-Martin sketch represented a great advance, it also created an open question. For a given level of accuracy, what is the smallest sketch that will provide estimates for that level of accuracy? Durand and Flajolet (2003) introduced the LogLog sketch, which reduced the size of the data structure to $O(m \log\log N)$ while maintaining an accuracy of $O(1/\sqrt{m})$. This algorithm was subsequently further refined with the HyperLogLog (Flajolet et. al. 2007). On the other hand, Indyk and Woodruff (2003) gave a lower bound, showing that any sketch providing accuracy $O(1/\sqrt{m})$ must use at least $O(m)$ space in a certain computational model. Kane et. al. (2010) then gave an algorithm requiring $O(m + \log N)$ space and giving $O(1/\sqrt{m})$ accuracy.

However, this algorithm is not believed to be practical, and the current state of the art is the HyperLogLog++ of Heule et. al. (2013).

DESCRIPTION

This disclosure describes a practical cardinality estimator that uses $O(m + \log\log N)$ space and has a conjectured $O(1/\sqrt{m})$ relative error. The described cardinality estimator improves upon the best-known space bounds of previous cardinality estimators and matches on relative error. Empirical evidence is provided showing that the performance of the algorithm compares favorably with the current state of the art.

This described cardinality estimator can be implemented as a software library or as a software system. As mentioned earlier, the described cardinality estimator is more space efficient than previous solutions. This has two immediate impacts. First, in applications where it is useful to store data structures for estimating the cardinalities of data sets, the estimator can enable reduction in the overall amount of space that is used. Second, in applications where it is necessary to transmit such data structures over networks, it can enable reduction in the amount of network bandwidth consumed.

The algorithm presented here extends the ideas behind the Liquid Legions estimator presented in Wright et. al. (2020). This algorithm, as with Liquid Legions, makes use of an exponentially decreasing schedule of probabilities of register activations. That is to say, if p_i is the probability that a value will hash to the i^{th} register, then there is a positive constant α such that $p_{i+1}/p_i = e^{-\alpha}$. The number of registers maintained by the algorithm is proven to be proportional to $1/\alpha$ with high probability, and empirical evidence is given that the relative error is proportional to $\sqrt{\alpha}$. Thus, taking $m = 1/\alpha$, this gives an algorithm with $O(m)$ registers and

$O(1/\sqrt{m})$ standard error. In addition, a new estimation method is given that improves over the method used in Liquid Legions. As with HyperLogLog and Liquid Legions, the sketch representing the union of two sets is easily computed from their respective sketches.

Overview

The input consists of a sequence $x_1, x_2, ..., x_s$ of identifiers, of which an unknown number n are unique. Also, as input, a small positive real number α is given that indirectly determines both the space used by the algorithm and the accuracy of the results produced. As output, the algorithm produces an estimate $\hat{n}$ of n.

The probability mass function $f(i),\ i = 0, 1, 2, ...$ is defined as $f(i) = (1 - e^{-\alpha})\, e^{-\alpha i}$, and $F(i)$ is the cumulative mass function associated to f, so $F(i) = 1 - e^{-\alpha(i+1)}$.

Imagine that the algorithm maintains an infinite bit array $B[0],\ B[1],\ B[2],\ ...,$ which is initialized to 0. For each input value x_j, an index into this bit array is determined and the corresponding bit is set to 1. It is assumed that a hash function h is available that hashes each x_i uniformly at random to values in the unit interval. The location $L(x_j)$ assigned to x_j is computed as:

$$L(x_j) = \lfloor F^{-1}(h(x_j)) \rfloor = \left\lfloor \frac{\log(1 - h(x_j))}{\alpha} \right\rfloor.$$

Consequently, the probability that x_j is assigned to location i is given by $f(i)$.

After inserting n items, the bitmap B will have a characteristic structure, depicted below in Figure 1. The head of the bitmap will consist of a sequence of 1's. Then, there will be a zero, possibly followed by a sequence of 0's and 1's. The portion from the first 0 to the final 1 is called

the fringe. In the diagram, these are the positions from i to j. This will be followed by the tail, which consists of an infinite sequence of 0's.

The central thesis of this disclosure is that for sufficiently large n, the size of the fringe depends only on α and not on n. In fact, with high probability, the size of the fringe is $O(1/\alpha)$. Thus, a data structure that stores only the fringe will achieve the objectives.

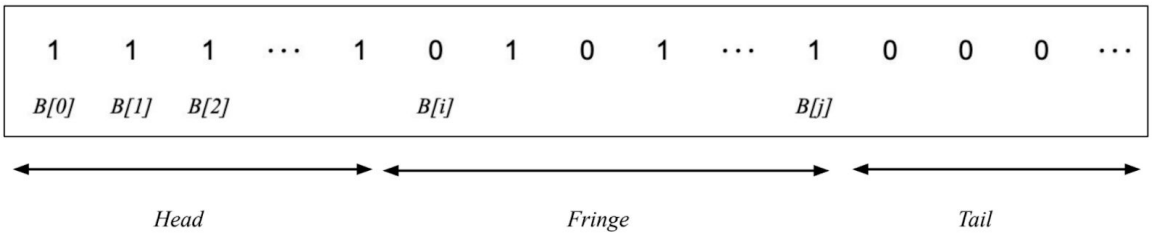

Fig. 1: Bitmap after *n* insertions have been made.

Analysis

In the following, n will be considered to be fixed but unknown. Let K_{max} be the highest index in B that is one, and similarly, let K_{min} be the lowest index in B that is zero. Thus, the size of the fringe is $K_{max} - K_{min} + 1$. Our goal is to bound this quantity. In particular, we will show that for any $\varepsilon > 0$, there is a value $K(\alpha, \varepsilon)$ such that $Pr(K_{max} - K_{min} > K(\alpha, \varepsilon)) < \varepsilon$.

Our proof strategy is as follows. Let m_n be the largest positive integer that is less than or equal to $(\log n) / \alpha$. In some sense, m_n can be thought of as the midpoint of the fringe. We will show that $K_{max} - m_n$ is bounded with high probability, and we will also show that $m_n - K_{min}$ is bounded with high probability. Consequently, $K_{max} - K_{min} = (K_{max} - m_n) - (m_n - K_{min})$ will be bounded with high probability. We state a number of theorems below, proofs for which are provided in the appendix.

- **Theorem 1**. For all $\varepsilon > 0,$ there is a value $U(\alpha, \varepsilon)$ such that $Pr(K_{max} - m_n > U(\alpha, \varepsilon)) < \varepsilon.$
- **Theorem 2**. For all $\varepsilon > 0,$ there is a value $L(\alpha, \varepsilon)$ such that $Pr(m_n - K_{min} > L(\alpha, \varepsilon)) < \varepsilon.$
- **Theorem 3.** If $0 < \alpha < 1,$ then $U(\alpha, \varepsilon) + L(\alpha, \varepsilon) = O(-\log(\varepsilon)/\alpha).$
- **Theorem 4.** If $0 < \alpha < 1,$ then with probability at least $1 - \varepsilon,$ $K_{max} = O(\log(n/\varepsilon)/\alpha).$ This follows from 1 and 3.

Theorems 1 and 2 together prove that the size of the fringe is bounded with high probability. Theorem 3 establishes that the size of the fringe is $O(1/\alpha).$ Theorem 4 is used in the runtime analysis. A more precise bound than the one stated above is

$$U(\alpha, \varepsilon) + L(\alpha, \varepsilon) \leq -\frac{2 \log \varepsilon}{\alpha} + \frac{3 \log 2}{\alpha} + 1.$$

Algorithm

Python code for the cardinality estimator is given below in Figure 2. The data structure maintains three values: the fringe F, at a cost of $O(1/\alpha),$ the index f_{start} of the first element of the fringe, at a cost of $O(\log \log n),$ and $\alpha,$ at a cost of $O(1).$

If B_k is the state of the hypothetical bitmap after k items have been inserted and F_k is the state of the fringe, then the algorithm maintains the invariant that $F_k[i] = B_k[i + f_{start}].$

Insertions

The insertion algorithm is as follows. When a new item x is to be inserted, if $L(x)$ is less than $f_{start},$ it is discarded. Otherwise, $F[L(x) - f_{start}]$ is set to 1. If $F[0]$ is now 1, then the first element of F is deleted and f_{start} is incremented. This is repeated until either the fringe is empty or $F[0] = 0.$

The insert function contains two operations that are not $O(1)$, the append operation at line 21 and the loop in lines 23-25. In view of Theorem 4, with high probability, the append operation will not be performed more than $O(\log n)$ times (treating ε and α as constants for this analysis). Consequently, the amortized cost of the code in line 21 is $O(1)$. The code in lines 23-25 deletes items that were previously inserted into F in line 21. Consequently, the total cost of these operations cannot be more than $O(\log^2 n)$, which on an amortized basis, is again $O(1)$.

Cardinality Estimation

To compute the estimated cardinality of the data structure, a maximum likelihood method is used. The probability that $B[i] = 0$ after n insertions is $(1 - f(i))^n$, while the probability that $B[i] = 1$ is $(1 - (1 - f(i))^n)$. Consequently, the likelihood of n given B is given by the formula:

$$\mathcal{L}(n|B) = \prod_{i=0}^{K_{max}-1} B[i](1 - (1 - f(i))^n) + (1 - B[i]) * (1 - f(i))^n$$

The maximum of this function can easily be found via a univariate optimization method such as Brent's algorithm. The estimates produced by this method appear to be highly accurate.

Maximum likelihood estimators have a couple useful properties. They are consistent, e.g., they converge in probability to the quantity being estimated. And they are efficient, meaning that no other unbiased estimator can asymptotically achieve a lower mean squared error.

```
import numpy as np
from scipy.optimize import minimize_scalar

class LinearLegionsSketch:
 def __init__(self, alpha):
   self.f_start = 1  # Starting index of fringe.
   self.fringe = np.array([])
   self.alpha = alpha

 def pmf(self, i):
   """Probability that B[i] will be chosen for an insertion."""
   return (1 - np.exp(-self.alpha)) * np.exp(-self.alpha*i)

 def insert(self, x):
   """Inserts x, where 0 < x < 1."""
   i = int(-np.log(1-x)/self.alpha + 0.00001) - self.f_start
   if i < 0:
     return
   m = i - len(self.fringe) + 1
   if m > 0:
     self.fringe = np.append(self.fringe, [0] * m)
   self.fringe[i] = 1
   while len(self.fringe) > 0 and self.fringe[0] == 1:
     self.fringe = self.fringe[1:]
     self.f_start += 1

 def _negative_log_likelihood(self, n):
  if n < 2:
      return 1.0e99
   B = np.concatenate([np.ones(self.f_start), self.fringe])
   q = (1.0 - self.pmf(np.arange(len(B))))**n
   p = 1-q
   return -np.sum(np.log(p*B + q * (1.0-B)))

 def cardinality(self):
   """Estimates cardinality of sketch."""
   return minimize_scalar(lambda x: self._negative_log_likelihood(x),
     method='brent')['x']
```

Fig. 2: LinearLegions cardinality sketch

Empirical Results

Three central theorems have been stated in this disclosure: (1) the size of the fringe is a constant that depends on α but not on n, (2) the size of the fringe is proportional to $1/\alpha$, and (3) the relative error is proportional to $\sqrt{\alpha}$. Empirical evidence will now be given to support these theorems. In addition, we give empirical data on the bias and relative error in comparison to the Liquid Legions estimator.

Size of Fringe in Relation to n

Figure 3 shows the size of the fringe in relation to n for various values of α. Five different values of α were considered, from $\alpha = 0.005$ to $\alpha = 0.1$. Along the x axis, ten different values of n are plotted, from $n = 10,000$ to $n = 100,000$. For each value of n, 100 runs were performed, and the average size of the maximum fringe for each cardinality estimator was plotted. 95% error bars are also shown. As can be seen, for each value of α, the sizes of the fringe appear to be constant.

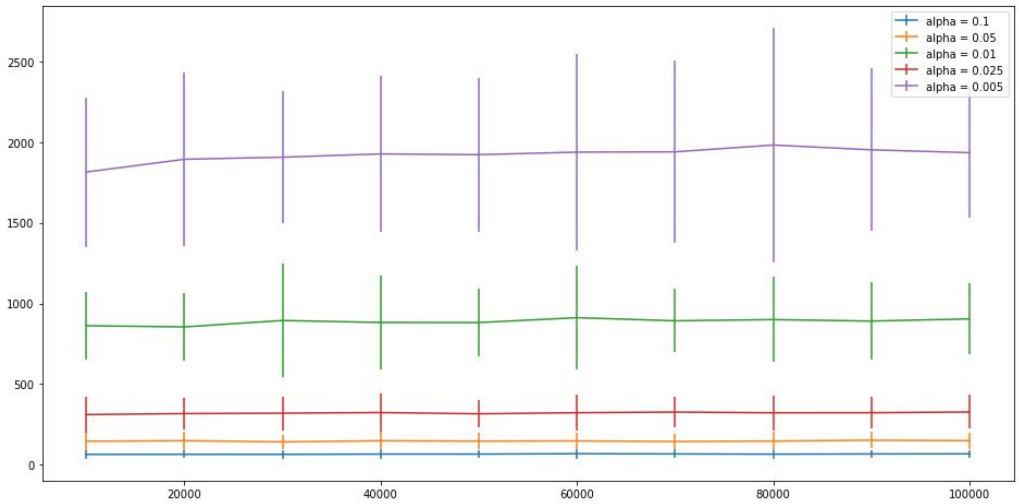

Fig. 3: Size of Maximum Fringe in Relation to n

Size of Fringe in Relation to 1/alpha

In the next experiment, the size of the fringe was plotted in relation to $1/\alpha$. Twenty values of α were chosen, ranging from 0.001 to 0.02. For each α, a cardinality estimator was constructed and 10,000 random values were inserted into it. The maximum size of the fringe was then recorded. This was repeated 4,000 times. The values were then plotted. The plot is given below in Figure 4.

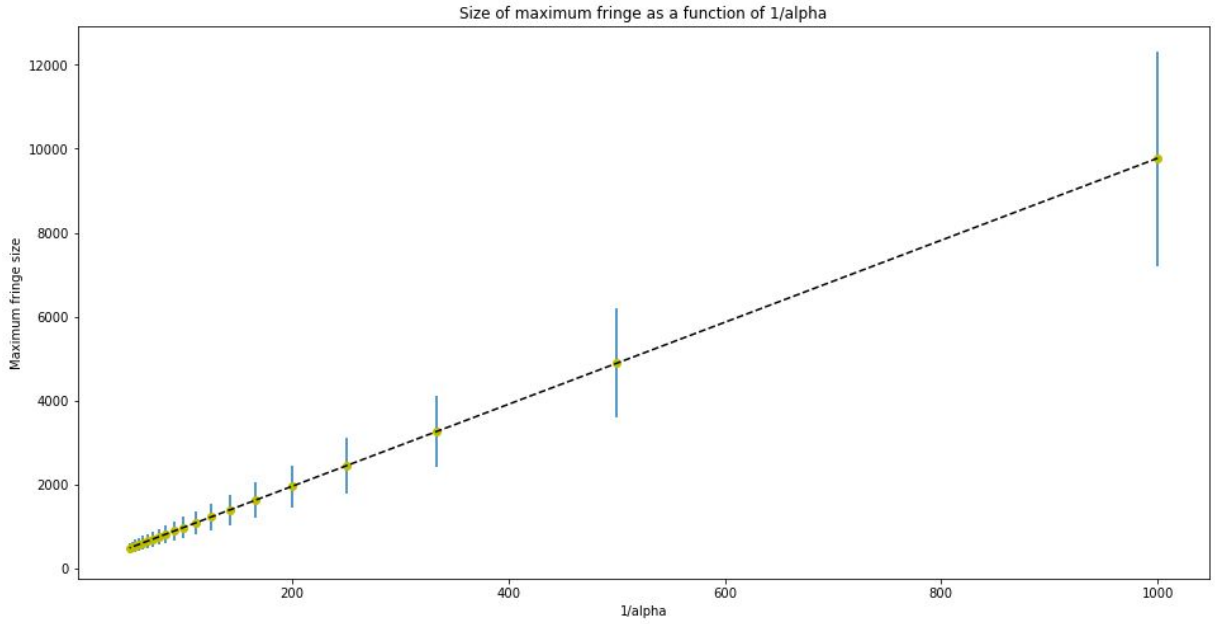

Fig. 4: Size of Maximum Fringe in Relation to $1/\alpha$.

If f_{max} is the maximum fringe size, then a linear regression of f_{max} on $1/\alpha$ gives the following fit:

$$f_{max} = 9.77 / \alpha + 1.65$$

The R^2 value of this fit exceeds 0.999.

Relative Error

The following graph shows the relative error in relation to $\sqrt{\alpha}$. The data used to generate this graph is the same as for the previous graph.

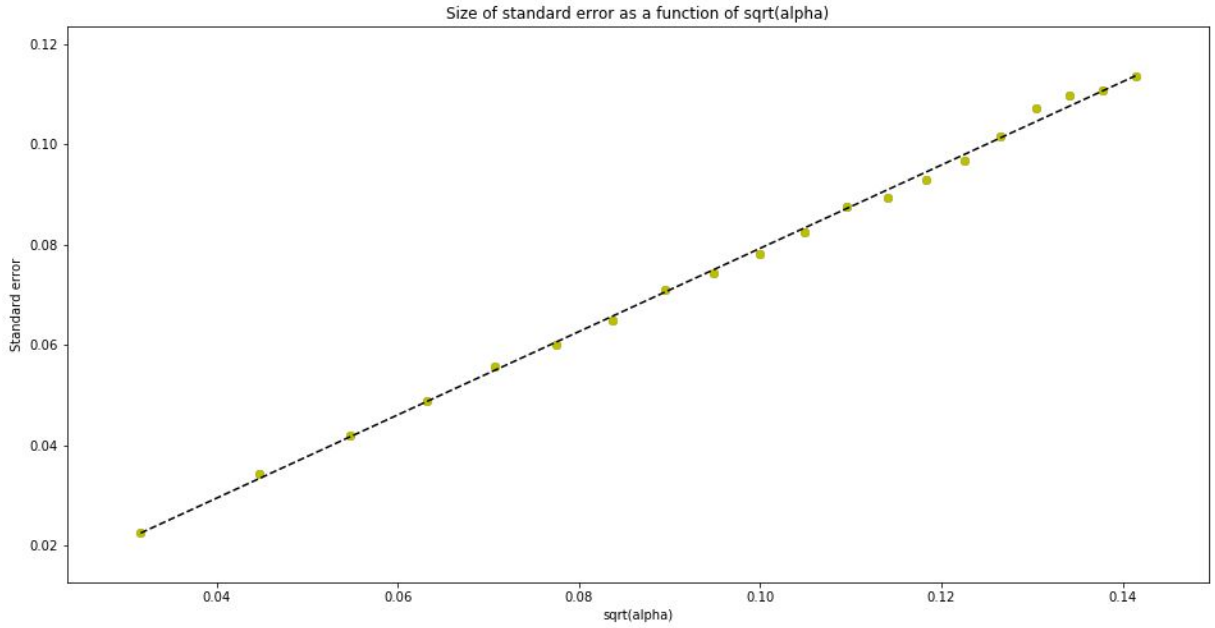

Fig. 5: Relative error in relation to $\sqrt{\alpha}$.

In $\hat{n}_{se}$ is the standard error of $\hat{n}/n$, then a linear regression generated the following fit:

$$\hat{n}_{se} = .83\sqrt{\alpha} - 0.0038$$

The R^2 value in this case was slightly worse, coming in at 0.99.

The following table summarizes the estimated sizes of the data structure and the relative error for a few values of α :

α	**Size of Sketch (in bits)**	**Relative Error**
0.00082	11,884	0.02
0.00028	35,344	0.01
0.00011	86,915	0.005

As a point of comparison, to obtain a relative error of 0.005, the HyperLogLog would require a precision parameter of $p = 15$ bits, thus requiring $2^{15} = 32,768$ registers, storing 6 bits per register, for a total storage cost of 196,608 bits. Thus, this represents a potential improvement of a factor of over 50% relative to the HyperLogLog.

Relative Bias

In this section and the next section, comparisons are given to the estimator used by the Liquid Legions sketch of Wright et. al. (2020). Liquid Legions also uses an exponentially decreasing schedule of register activation probabilities. Once the sketch has been computed, though, a different estimator is used to compute the cardinality of the set. Let S be the total number of bits set in a bitmap B of size M. Then, the expected value of S is given by

$$E[S] = \sum_{i=0}^{M} \Pr(B[i] = 1) = \sum_{i=0}^{M} (1 - (1 - f(i))^n)$$

Given an observed value of S, a univariate optimization algorithm can be used to determine the value of n that most closely matches S. This value of n is then output as the cardinality estimate. This estimator will be called the Liquid Legions estimator.

The relative bias of an estimator X is defined to be the $E_X[\hat{n}/n] - 1$. Measurements were made of the bias of the Linear Legions estimator and Liquid Legions estimator. The value of M was taken to be K_{max}. The dataset used was the same as for the previous two experiments.

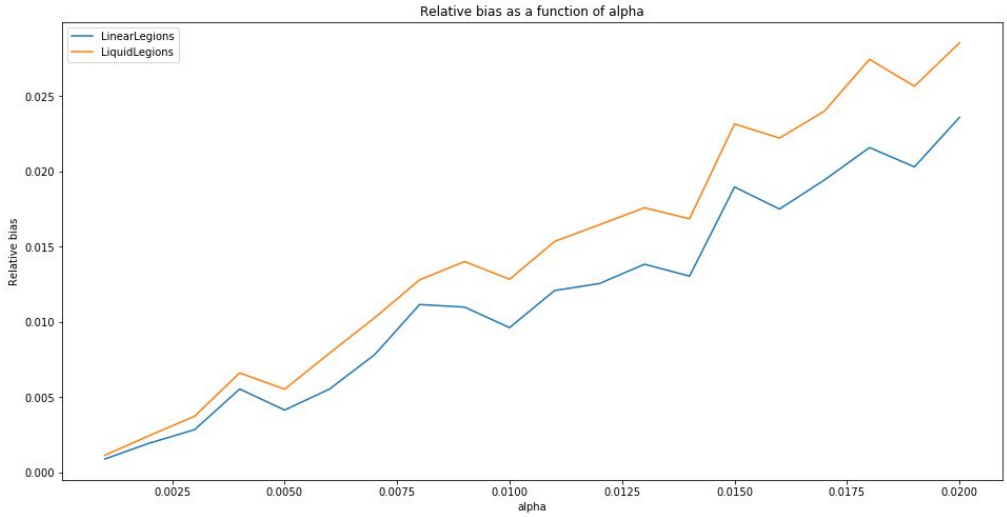

Fig. 6

As can be seen, both methods exhibit bias that decreases as $\alpha \rightarrow 0$, although the bias exhibited by Linear Legions is lower.

Relative Error Compared to Liquid Legions

The final plot compares the relative error of Linear Legions to that of Liquid Legions. As can be seen, Linear Legions shows a consistent improvement of about 10% relative to Liquid Legions.

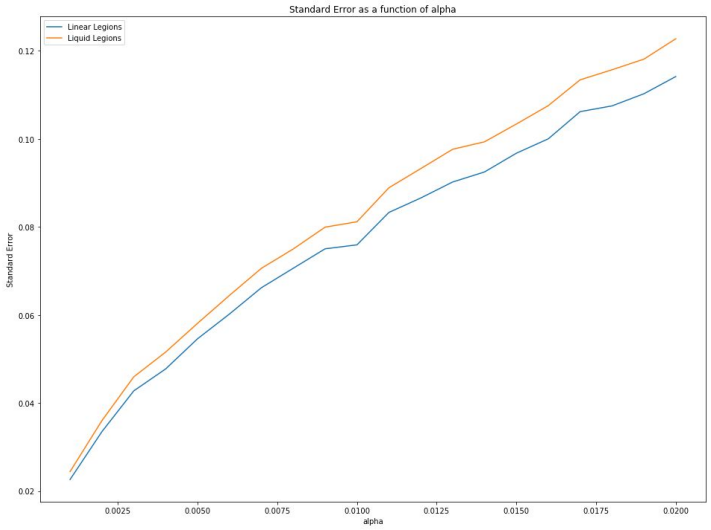

Fig. 7

CONCLUSION

This disclosure describes a new technique for cardinality estimation, named Linear Legions. It has better space than previously published estimators, while matching earlier techniques in standard error. Given Linear Legions sketches for two sets A and B, the sketch of the union is easily computed. Another issue is how to compute cardinalities by frequency. If pre-processing is allowed, then one approach would be to compute one sketch for each frequency. If f frequencies are measured, then the space requirement of this approach would be $O(fm)$. For example, if the maximum frequency is 10, then the space would be approximately $10m$. Alternatively, it is possible to maintain fingerprints and counts of items that are inserted in the sketch. If a 64-bit fingerprint and count is maintained, then the space requirement becomes $64m$, which is significantly larger. Another downside of the fingerprint-based approach is that elements with $f = 1$ will be overrepresented in the sketch. Consequently, the standard error for larger frequencies will be higher. Thus, the preprocessing approach is to be recommended.

NOTE

A mathematical proof that the standard error is $O(\sqrt{\alpha})$ has not been provided herein. While the empirical evidence is compelling, having a mathematical proof that the relative error is $O(\sqrt{\alpha})$ would tie up an important loose end.

REFERENCES

[1] Durand, Marianne, and Philippe Flajolet. "Loglog counting of large cardinalities." *European Symposium on Algorithms*. Springer, Berlin, Heidelberg, 2003.

[2] Flajolet, Philippe, and G. Nigel Martin. "Probabilistic counting algorithms for data base applications." *Journal of computer and system sciences* 31.2 (1985): 182-209.

[3] Flajolet, Philippe, et al. "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm." 2007.

[4] Heule, Stefan, Marc Nunkesser, and Alexander Hall. "HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm." *Proceedings of the 16th International Conference on Extending Database Technology*. 2013.

[5] Indyk, Piotr, and David Woodruff. "Tight lower bounds for the distinct elements problem." *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*. IEEE, 2003.

[6] Kane, Daniel M., Jelani Nelson, and David P. Woodruff. "An optimal algorithm for the distinct elements problem." *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2010.

[7] Kreuter, Benjamin, Craig William Wright, Evgeny Sergeevich Skvortsov, Raimundo Mirisola, and Yao Wang. "Privacy-Preserving Secure Cardinality and Frequency Estimation." (2020).

APPENDIX: Proofs of theorems

Theorem 1. For all $\varepsilon > 0$, there is a value $U(\alpha, \varepsilon)$ such that $Pr(K_{max} - m_n > U(\alpha, \varepsilon)) < \varepsilon$.

Proof. In fact, we can take $U(\alpha, \varepsilon) = 1 - (\log \varepsilon)/\alpha$. Let $r = m_n + U(\alpha, \varepsilon)$. As a preliminary, we note that $e^{-\alpha r} \leq \varepsilon/n$. This can be seen as follows:

$$\begin{aligned} e^{-\alpha r} &= e^{-\alpha(m_n + U(\alpha,\epsilon))} \\ &= e^{-\alpha(\lfloor \log n/\alpha \rfloor + 1 - \log \epsilon/\alpha)} \\ &\leq e^{-\alpha(\log n/\alpha - \log \epsilon/\alpha)} \\ &= e^{-(\log n - \log \epsilon)} \\ &= \epsilon/n. \end{aligned}$$

Recall that the cumulative mass function for assigning elements to B is $F(i) = 1 - e^{-\alpha(i+1)}$. This represents the probability that a randomly chosen identifier will be assigned to an index at location i or lower. Let $F_{max}(i)$ be the probability that all n identifiers are assigned to locations i or lower. Then, $F_{max}(i) = F(i)^n$. Thus,

$$\begin{aligned} \Pr(K_{max} - m_n > U(\alpha, \epsilon)) &= \Pr(K_{max} > m_n + U(\alpha, \epsilon)) \\ &= 1 - Pr(K_{max} \leq m_n + U(\alpha, \epsilon)) \\ &= 1 - F_{max}(m_n + U(\alpha, \epsilon)) \\ &= 1 - (1 - e^{-\alpha(m_n + U(\alpha,\epsilon)+1)})^n \\ &= 1 - (1 - e^{-\alpha(r+1)})^n \\ &\leq 1 - (1 - ne^{-\alpha(r+1)}) \\ &= ne^{-\alpha(r+1)} \\ &\leq n(\epsilon/n) \\ &= \epsilon. \end{aligned}$$

The first inequality is justified by the fact that $(1 - x)^n \geq 1 - nx$ if $0 < x < 1$. QED.

Corollary. Let B_{max} the largest index i for which $B[i] = 1$. For all $\varepsilon > 0$, the probability that $B_{max} < (\log(n/\varepsilon) + 1)/\alpha$ is greater than $1 - \varepsilon$.

Proof: $(\log(n/\varepsilon) + 1)/\alpha = (\log n + 1 - \log \varepsilon)/\alpha = m_n + U(\alpha, \varepsilon)$.

The upshot of this corollary is that the maximum index $B_{max} = O(\log(n)/\alpha)$ with arbitrarily high probability.

The second theorem is a bit harder to prove, and so we proceed through a series of lemmas.

Lemma 1. Let $k < m_n$. Then, $\Pr(B[m_n - k] = 0) \leq (e^{(1-e^{-\alpha})})^k$.

Proof:

$$
\begin{aligned}
\Pr(B[m_n - k] = 0) &= (1 - f(m_n - k))^n \\
&= (1 - (1 - e^{-\alpha})e^{-\alpha(m_n - k)})^n \\
&\leq \exp(-(1 - e^{-\alpha})e^{-\alpha(m_n - k)})^n \qquad (1) \\
&\leq \exp(-(1 - e^{-\alpha})e^{-\alpha(\log n/\alpha - k)})^n \qquad (2) \\
&= \exp(-(1 - e^{-\alpha})e^{-\log n}e^{\alpha k})^n \\
&= \exp(-(1 - e^{-\alpha})(1/n)e^{\alpha k})^n \\
&= \exp(-(1 - e^{-\alpha})e^{\alpha k}) \\
&= e^{-(1-e^{-\alpha})e^{\alpha k}} \\
&\leq e^{-(1-e^{-\alpha})ke^{\alpha}} \qquad (3) \\
&= (e^{-(1-e^{-\alpha})e^{\alpha}})^k \\
&= (e^{1-e^{\alpha}})^k
\end{aligned}
$$

The inequality at step (1) is justified because $1 - x < e^{-x}$. The inequality at step (2) is justified because $m_n \leq \log n/\alpha$. The inequality at line (3) is justified because $-e^{\alpha k} <- ke^{\alpha}$ if $k \geq 1$ and $\alpha \geq 0$. QED.

Lemma 2. $\Pr(K_{min} \leq m_n - k) \leq \frac{u^k}{1-u}$, where $u = e^{1-e^{\alpha}}$.

Proof:

$$
\begin{aligned}
\Pr(K_{min} \leq m_n - k) &\leq \sum_{i=0}^{m_n-k} \Pr(B[i] = 0) \\
&= \sum_{j=m_n-k}^{m_n} \Pr(B[m_n - j] = 0) \\
&\leq \sum_{i=0}^{m_n-k} u^{i+k} \qquad (4) \\
&= u^k \sum_{i=0}^{m_n-k} u^i \\
&= u^k \frac{1 - u^{m_n-k+1}}{1-u} \\
&\leq \frac{u^k}{1-u} \qquad (5)
\end{aligned}
$$

The inequality at line (4) is justified by Lemma 1, and the inequality at line (5) is justified because $u < 1$ and therefore $1 - u^{m_n-k+1} < 1$. QED.

Lemma 3. If $0 < \alpha < 1$, then $\frac{1}{1-u} < \frac{2}{\alpha}$, where $u = e^{1-e^{\alpha}}$.

Proof: We start by noting that

$$
\begin{aligned}
u &= e^{1-e^{\alpha}} \\
&\leq e^{1-(1+\alpha)} \qquad (6) \\
&= e^{-\alpha} \\
&\leq 1 - \alpha + \alpha^2/2 \qquad (7) \\
&\leq 1 - \alpha/2. \qquad (8)
\end{aligned}
$$

The inequality at line (6) is justified because $1 + x < e^x$. The inequality at line (7) is obtained by truncating the Taylor series for e^x. The inequality at line (8) is obtained by observing that if $\alpha < 1$, then $\alpha^2 < \alpha$, so we may replace $\alpha^2/2$ by $\alpha/2$. We now note that

$$
\begin{aligned}
u < 1 - \alpha/2 &\iff \alpha/2 < 1 - u \\
&\iff \frac{1}{\alpha/2} > \frac{1}{1-u} \\
&\iff \frac{1}{1-u} < \frac{2}{\alpha}.
\end{aligned}
$$

QED.

Lemma 4. $u^{1/\alpha} < 1/e,$ where $u = e^{1-e^{\alpha}}$.

Proof: As noted in the previous lemma, $u \leq \exp(-\alpha)$. Therefore, $u^{1/\alpha} \leq \exp(-\alpha)^{1/\alpha} = e^{-1}$.

QED.

Lemma 5. If $0 < \alpha < 1,\ 0 < \varepsilon < 1$ and $k > \frac{-\log \varepsilon/2}{\alpha}$, then $\frac{u^k}{1-u} \leq \varepsilon,$ where $u = e^{1-e^{\alpha}}$.

Proof: Here

$$
\begin{aligned}
\frac{u^k}{1-u} &\leq \frac{u^{(-\log \epsilon/2)/\alpha}}{1-u} && (9) \\
&= \frac{(u^{1/\alpha})^{-\log \epsilon/2}}{1-u} \\
&\leq \frac{(e^{-1})^{(-\log \epsilon/2)}}{1-u} && (10) \\
&= \frac{\epsilon/2}{1-u} \\
&\leq \left(\frac{\epsilon}{2}\right)\left(\frac{2}{\alpha}\right) && (11) \\
&= \epsilon.
\end{aligned}
$$

The inequality at line (9) is justified because $0 < u < 1$ and $k > \frac{\log \varepsilon/2}{\alpha}$. The inequality at line (10) comes from applying Lemma 4. The inequality at line (11) comes from applying Lemma 3.

QED.

Theorem 2. If $0 < \alpha < 1,$ there is a value $L(\alpha, \varepsilon)$ such that $Pr(m_n - K_{min} > L(\alpha, \varepsilon)) < \varepsilon.$

Proof: We take $L(\alpha, \varepsilon) = -\log(\varepsilon/2)/\alpha,$ where $u = e^{1-e^{\alpha}}$. We then have

$$\begin{aligned}
\Pr(m_n - K_{min} > L(\alpha,\epsilon) &= \Pr(K_{min} < m_n - L(\alpha,\epsilon)) \\
&\leq \frac{u^{L(\alpha,\epsilon)}}{1-u} && (12) \\
&= \frac{u^{-(\log \epsilon/2)/\alpha}}{1-u} \\
&\leq \epsilon. && (13)
\end{aligned}$$

Line (12) is justified by Lemma 2, and line (13) is justified by Lemma 5. QED.

Theorem 3. $U(\alpha, \varepsilon) + L(\alpha, \varepsilon) = O(-(\log \varepsilon)/\alpha)$.

Proof: Recall that $U(\alpha, \varepsilon) = 1 - (\log \varepsilon)/\alpha$ and $L(\alpha, \varepsilon) = -(\log \varepsilon/2)/\alpha$, where $u = e^{1-e^{\alpha}}$. Both of these are $O(-\log \varepsilon/\alpha)$. QED.