Minner: Improved Similarity Estimation and Recall on MinHashed Databases

ANONYMOUS AUTHOR(S)

Quantization is the state of the art approach to efficiently storing and searching large high-dimensional datasets. Broder'97 [4] introduced the idea of Minwise Hashing (MinHash) for quantizing or sketching large sets or binary strings into a small number of values and provided a way to reconstruct the overlap or Jaccard Similarity between two sets sketched this way.

In this paper, we propose a new estimator for MinHash in the case where the database is quantized, but the query is not. By computing the similarity between a set and a MinHash sketch directly, rather than first also sketching the query, we increase precision and improve recall.

We take a principled approach based on maximum likelihood (MLE) with strong theoretical guarantees. Experimental results show an improved recall@10 corresponding to 10-30% extra MinHash values. Finally, we suggest a third very simple estimator, which is as fast as the classical MinHash estimator while often more precise than the MLE.

Our methods concern only the query side of search and can be used with any existing MinHashed database without changes to the data.

Additional Key Words and Phrases: datasets, quantization, minhash, estimation, sketching, similarity join

ACM Reference Format:
Anonymous Author(s). 2021. Minner: Improved Similarity Estimation and Recall on MinHashed Databases. 1, 1 (February 2021), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Set data (or sparse binary or categorical data) is a staple of data science. Efficient search and joins of such data is used in document deduplication [5], association rule learning [25], and for searching genomes and metagenomes in Bioinformatics [19].

Quantization is the act of representing data from a large or continuous space by a smaller set of discrete finite values. Also known as sketching or hashing, this often allows storing very large datasets on a single computer, or on fewer servers than otherwise needed. At the same time, because the compression and increases data locality, it has become a key component to processing such data efficiently.

MinHash sketches are randomized quantizations of sets (or equivalently 0/1 vectors). The idea is to pick K random functions $h_i : U \to [0, 1]$ and define the sketch of $X \subseteq U$ to be

$$q(x) = (\arg\min_{x \in X} h_1(x), \dots, \arg\min_{x \in X} h_K(x)).$$

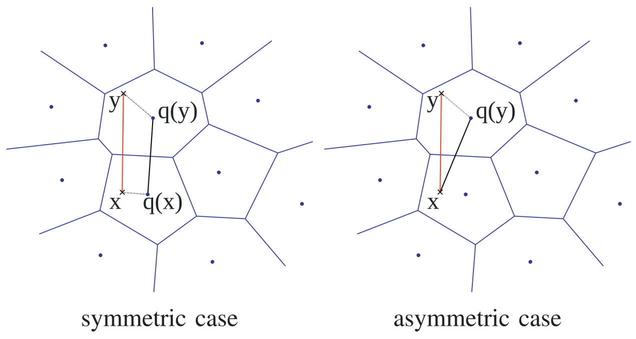

Fig. 1. Figure by Jegou et al. [14] illustrating the difference between symmetric and asymmetric estimation, as used in Euclidean Nearest Neighbour Search. The distance $q(y) - x$ is a better approximation of $y - x$ than $q(y) - q(x)$. In the set setting, when q is MinHash, it is not clear what it would even mean to compute $q(y) - x$?

After early uses in statistics [3] and correlation estimation [12], the term was coined by Broder [4, 5] in the context of detecting near-duplicate web pages. The sketch is known to be near-optimal [20] for estimating set similarity on a given space budget.

A typical scenario is that we want to store some sets $Y_1, \dots,$ so we compute the MinHash for each of them, with perhaps $K = 30$. (See fig. 2 for an example quantized database.) Now, given a new set, X, we quantize it and estimate the similarity with each Y_i by $\|q(X) - q(Y_i)\|_1/K$, which has expectation $\frac{|X \cap Y_i|}{|X \cup Y_i|}$, known as the Jaccard similarity between X and Y_i. Since $q(X)$ only has to be computed once, and each subsequent estimate takes time $O(K)$, rather than $|X \cap Y|$ if we were to compute the Jaccard similarity directly. Thus, the quantized database can be searched substantially faster than the direct approach. Sketching can also be combined with space partitions to produce state of the art performing set similarity search [7], but in this paper we focus on quantization only.

Quantization is also central in state of the art Euclidean nearest neighbour search and Maximum Inner Product Search (MIPS) [13]. In their seminal paper, Jegou et al. [14] argued for the application of *asymmetric distance estimation* as a way to improve search accuracy and speed in this context. Instead of sketching the query and computing the similarity "in sketch-space", $\|q(x) - q(y)\|_2$, one can use the full information of the query and compute $\|x - q(y)\|_2$ reducing the space requirement for equal recall by up to a factor of four. See fig. 1 for a visualization.

While asymmetric estimation is a natural idea for Euclidean distance, it is less clear how it might apply in the context of set search and MinHash quantization. Somehow we have to compute a similarity value, given X and $q(Y)$ better than $\|q(X) - q(Y)\|_1/K$. An indication that this may be possible is the case where the MinHash value of Y is not equal to the MinHash of X, but is perhaps equal to the second or third smallest value—that ought to count as evidence towards the sets being similar, even if the classical MinHash sketch would count it as evidence to the opposite.

1.1 Results

In this paper we take a principled approach to this problem.

1. We derive an estimator based on maximum likelihood. We analyse its variance, which we find to be 28% lower than the classical sketch-space estimator. (See figures 4 and 6.) For small similarities the we reduce the variance by a factor $\frac{|Y|}{|X|+|Y|}$ showing particularly good results when the (known) set X is much larger than the unknown set Y.
2. We investigate relaxations and numerical methods, trading precision for speed of estimation. (See tables 1 and 2.) A particular good choice is dubbed the "Minner Estimator" since it is based on counting the number of elements in X that hash to a value *smaller than the minimum* hash value of Y.
3. We run experiments on several large set datasets from [17], such as the Netflix dataset originally from KDD-Cup 2007. We show a reduction in the MinHash values needed of up to 30% for a given recall@10.

While our focus is mainly on applications characterized as "one-many", such as search, many applications characterized as "many-many" are trivially reduced to n times one-many. We thus also obtain better performance for important tasks such as duplicate detection and nearest neighbour graph construction.

A non-goal of the paper is to make the fastest possible implementation of Set Similarity Search. For this reason we don't include experiments measuring the runtime of our estimators. To be competitive in raw running time requires a serious engineering task with papers like [13] including 33,812 lines of optimized C and assembly code by many authors. In section 4 we discuss hardware level optimizations of this kind.

1.1.1 *Lower K for a given Variance and Recall.*

Technically, the most difficult part is in the precise variance analysis of the maximum likelihood estimator. Since our estimators are unbiased, the variance corresponds to the mean squared error when estimating similarity for pairs of sets. A factor two loss in variance would need to be counterbalanced by a factor two increase in the number of MinHash values, significantly reducing practical performance. One may ask other questions, such as the necessary K to obtain high probability confidence bounds, but since we work with independent values, this is mostly trivial.

An important "downstream" measure is the recall on real datasets. This determines how large a K is needed in order to return the true nearest neighbour when scanning a database with estimated similarities. The exact value depends heavily on the "difficulty" of the dataset, and varies between 20 and more than 500 for a 90% recall@10 in our experiments. In every case our new method is able to obtain a significantly higher recall when given the same information as is available to the classical MinHash estimator.

Other papers have focused on compressing the MinHash sketch itself, a famous algorithm being the [11]. In general $O(\log\log u + K \log K)$ bits suffice for similarity estimation [8], improving over storing the ranks directly with $\lceil K \log_2 u \rceil$ bits. Our paper differs by focusing on *reducing* K, which can then be combined with those results for even greater compression. Reducing K also has the benefit of increasing processing speed, something not gained by simply storing the same sketch in less space.

MinHashed Database						
id	Size, n_y	r_1	r_2	r_3	…	r_K
0	254	4594	4439	9295	…	658
1	107	66	3675	457	…	6805
2	3322	342	1173	11	…	409
3	501	9928	226	603	…	2784

Fig. 2. In this example MinHased Database four sets have been quantized into K values each. Instead of storing the value of $h : [u] \to [0,1]$ as a real number we have used the equivalent representation of ranks, in which h is a random bijection $h : [u] \to [u]$. This allows more efficient compression of the MinHash values.

2 BACKGROUND AND RELATED WORK

Estimators for MinHash, other than the "classic" one referred to in this paper, have been derived for various purposes. Cohen [8] made improved estimators for graph algorithms, and Ertl [9] derived better estimators for Flajolet's HyperLogLog [12] variation of MinHash. The extremely successful Mash [19] software in Bioinformatics works by a Bayesian estimator of sketch similarity that takes into account the process of 'mer'-tokenization that created the data. However for the simple task of estimating similarities in the many-one setting, there appears to have been no prior work.

2.1 Quantization and Search

Since Jegou et al. 2010 [14] quantization has been a critical part of fast search data structures. In particular the approach of Product Quantization, which sketches vectors in $\mathbb{R}^d$ in a fast, data-sensitive way. Recently Guo et al. [13] broke all records [2] for Maximum Inner Product Search (MIPS) and Cosine Similarity, based on a new Product Quantization technique sensitive to those measures. The secret to these amazing results is the use of Single instruction, multiple data (SIMD) instructions on modern CPUs, which can consume large batches of quantized data per instruction.

In comparison the landscape for Set Similarity Search is less developed. Recent theoretical work [1, 6] has discovered the optimal randomized space partitions to use and replaced MinHash after 20 years as the best known approach. However the state of the art implementations [7] still use MinHash sketching for faster similarity estimation among points in the space region. In contrast to the Euclidean case, they have thus far had to the classical symmetric estimator.

2.2 Alternative sketches

There are a number of other sketches for estimating set similarity that we do not study in this paper. In general any sketch that allows cardinality estimation and taking the union of two sketches can be used to estimate the set overlap and similarity. Most estimators for these sketches are of the symmetric type, so it would be interesting to study whether some of them have good asymmetric estimators as well.

HyperLogLog [11], HyperMinHash [24], MaxLogHash [23], SetSketch [10] and b-Bit MinHash [15] focus on compressing coordinated samples, however they don't try to get extra information *per sample* as we do in this paper.

MinHash itself has variations, such as bottom-k MinHash and k-partition MinHash. Cohen [8] gives a nice survey of those as well as many more variations and applications. Thorup [22] analyses bottom-k in a setting superficially similar to ours: For two sets $Y \subseteq X$ and S a bottom-k sketch of X, he bounds the deviation of $|S \cap Y$ from its expectation. That is, he compares a sketched set with an unsketched set. However, since Y is a subset of X it turns out, that for bottom-k the intersection $S \cap Y$ is exactly the same as $S \cap S(Y)$, so he is still in the classical "symmetric" setting.

3 THE ESTIMATORS

In this section we develop a maximum likelihood estimator (MLE) for MinHash, analyse the variance and compare it to the classical MinHash estimator. Such an estimator has the lowest possible variance asymptotically as $K \longrightarrow \infty$, but can be slow to evaluate. We thus proceed to develop a third estimator that is as fast as the classical estimator while experimentally nearly on par with the MLE in variance. Using numerical methods we finally show how to get the full power of the MLE in time linear in the number of MinHash values.

A quick note on notation: If $u \in \mathbb{N}$, we use $[u] = \{0, \dots, u-1\}$ to indicate the set of numbers up to u. If P is a proposition, we use $[P]$ to indicate the variable that is 1 if P and 0 if not P.

3.1 Maximum Likelihood Estimator

In the classical analysis of MinHash (and consistent hashing) we imagine two sets $X, Y \subseteq [u]$ being given, and then compute the probability that a random hash function $h : [u] \longrightarrow [0,1]$ will pick its smallest value in the intersection $X \cap Y$. This turns out to depend only on the size of $X \cap Y$, and so it doesn't matter that we don't actually know the sets X and Y while making the estimates.

To improve upon the classical estimator we have to use the fact that we in fact know h at the time of making the estimate. However if we just take h as a given in the analysis, there won't be anything random left, and the analysis will end up depending on the values of Y. In case of maximum likelihood estimation the analysis is closely tied to the actual algorithm, so depending on Y, which we don't know, is not a good idea.

Our compromise is to assume the values of h on X are known, but not the values of h outside of X. This results in an analysis (and an estimator) that uses the information given to the algorithm about X, but doesn't use any information about Y, just as in the original analysis. Note that this model of analysis is only used for deriving the MLE. When, in section 3.2, we analyse the variance, we will assume h is all unknown and get an expression only in terms of $|X|$, $|Y|$ and $|X \cap Y|$.

We discuss one final model. Recall that we can equivalently assume $h : [u] \longrightarrow [u]$. We call the output of this h the "rank", since it describes the placement of the hash value in the order of hash values of $[u]$. Let $\mathcal{X} = \{h(x) : x \in X\}$ and $\mathcal{Y} = \{h(y) : y \in Y\}$. Then knowing h on X corresponds to knowing $\mathcal{X}$, but sampling $\mathcal{Y}$ uniformly at random. This is a useful model combinatorially, since it completely removes h from the analysis. [1]

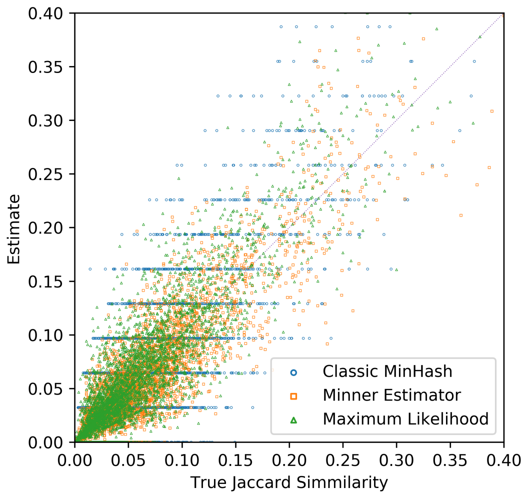

Fig. 3. Estimated vs. True Jaccard similarity on a random query in the Netflix dataset with 5,000 pairs, using $K = 31$. The classical estimator can only take $K + 1$ different values which leads to some visual banding. Our new estimators are more free and overall more concentrated around the diagonal.

Given a random variable and a statistical model, "estimation" is the task of inferring the parameters to the model on the basis of the observation. A maximum likelihood estimator chooses the parameters that maximizes the probability that the model generates the particular observed value. That is, if the observation is equal to ρ with probability $p(\rho;\ \theta)$, θ is unknown, once we observe ρ, we estimate θ as the $\arg\max p(\rho;\ \theta)$.

In our case we get the following model discussed above: Given a set $\mathcal{X}$ of size n_x and values n_y and v, we sample a set $\mathcal{Y}$ with $|\mathcal{Y}| = n_y$ and $v = |\mathcal{X} \cap \mathcal{Y}|$. We let r be some MinHash value the estimator may observe. The log-likelihood of the observation is then:

$$\ell(r; v) = \log \Pr_{\mathcal{Y}}[\min \mathcal{Y} = r],$$

in other words, the probability that a set $\mathcal{Y}$ sampled with overlap v with $\mathcal{X}$ had this particular value as its smallest.

We note that if we do not consider n_y to be known, we can let the model have two parameters (rather than just v) and estimate both of them. This could be done in a Bayesian way by for example counting the frequency of set sizes in the database and build this into the model. However in the database model not much space is saved, since the set size is only one value out of K.

We define $n_x = |\mathcal{X}|$ and the observed rank $R = \min \mathcal{Y}$. We also define $m = m(r) = \sum_{x \in \mathcal{X}} [x < r]$ to be the number of values in $\mathcal{X}$ smaller than r.

[1]It also suggests a future Bayesian estimator in which Y is not assumed to be uniformly distributed, but follow some prior distribution, such as the "skewed data" model of [18].

Proposition 1.

$$\Pr_Y[R = r] = \begin{cases} \frac{\binom{n_x-m-1}{v-1}\binom{u-r-(n_x-m)}{n_y-v}}{\binom{n_x}{v}\binom{u-n_x}{n_y-v}} & if\, r \in \mathcal{X} \\ \frac{\binom{n_x-m}{v}\binom{u-r-1-(n_x-m)}{n_y-v-1}}{\binom{n_x}{v}\binom{u-n_x}{n_y-v}} & if\, r \notin \mathcal{X} \end{cases}$$

Note we take $\binom{n}{k} = 0$ for $n < k$. In particular this may happen if $n_x - m < v$. The probability of $R = r$ in this case is 0, since $n_x - m$ is the number of x-ranks at least r, and all of $\mathcal{X} \cap \mathcal{Y}$ must have rank at least r.

Proof. Not considering R there are $\binom{n_x}{v}\binom{u-n_x}{n_y-v}$ ways to choose $\mathcal{Y}$ such that $|\mathcal{Y}| = n_y$ and $|\mathcal{X} \cap \mathcal{Y}| = v$. We proceed by cases..

First consider the case $r \in \mathcal{X}$. Then the remaining $v - 1$ overlapping elements have to be chosen from $\{x \in \mathcal{X} : x > r\}$. by definition of m there are $n_x - m - 1$ such values. The remaining $n_y - v$ non-overlapping elements have to be chosen from $\{x \notin X : x > r\}$. There are $u - r$ elements in $[u]$ greater than r, and of those $n_x - m$ are in $\mathcal{X}$. Thus the number of ways to choose $\mathcal{Y}$ with $r \in \mathcal{X}$ is $\binom{n_x-m-1}{v-1}\binom{u-r-(n_x-m)}{n_y-v}$.

The case $r \notin \mathcal{X}$ follows by similar arguments. □

Using proposition 1 we can write the log-likelihood in the following concise manner:

$$\ell(r; v) = \log \frac{\binom{n_x-m-[r\in\mathcal{X}]}{v-[r\in\mathcal{X}]}\binom{u-r-[r\notin\mathcal{X}]-(n_x-m)}{n_y-v-[r\notin\mathcal{X}]}}{\binom{n_x}{v}\binom{u-n_x}{n_y-v}}. \quad (1)$$

If we observe $K > 1$ values $r_1, r_2, \ldots, r_K$ we get, by independence of the MinHash functions, a log likelihood of

$$\ell(r_1; v) + \ell(r_2; v) + \cdots + \ell(r_K; v).$$

It is trivial (if not efficient) to enumerate all $v \in [\min\{n_x, n_y\} + 1]$ and compute which one has the highest log-likelihood.

We finally define our estimators for intersection (T_v) and Jaccard similarity (T_j).

Definition 1 (Maximum Likelihood Estimator (MLE)). *The maximum likelihood estimators for respectively set overlap and Jaccard similarity are*

$$T_v(r_1, \ldots, r_K) = \arg\max_{v\in[\min\{n_x,n_y\}+1]} \ell(r_1; v) + \ell(r_2; v) + \cdots + \ell(r_K; v).$$

$$T_j(r_1, \ldots, r_K) = \frac{T_v(r_1, \ldots, r_K)}{n_x + n_y - T_v(r_1, \ldots, r_K)}.$$

The re-parametrizing the estimator to Jaccard follows from the two quantities being monotone in each other. Hence if T_v maximizes the likelihood of v, T_j will maximize the likelihood of the Jaccard similarity.

3.2 Analysis

We want to analyse the MLE estimator in the model where h is unknown. In this setting we have for the MinHash estimator $E[T] = \frac{1}{K}\sum_i \Pr[q_i(X) = q_i(Y)] = j$ and

$$V[T] = \frac{E[(T-j)^2]}{K} = \frac{E[T^2] - j^2}{K} = \frac{\Pr[T = 1] - j^2}{K} = \frac{j(1-j)}{K}. \quad (2)$$

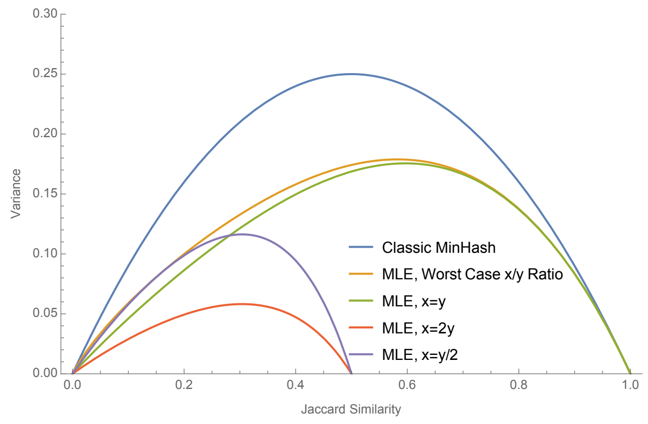

Fig. 4. Variance of maximum likelihood estimator based on Fischer Information bound. For j close to 0 or 1 the worst case MLE bound is asymptotically equal to the Classical bound, whereas for $j \approx 0.21$ has only $\approx 62\%$ of the variance. See fig. 5 for a corresponding experimental test.

The expectation and variance thus depends only on the similarity and not on the specifics of X and Y.

The main work of this section will be proving the following proportion:

Proposition 2. *As $K \to \infty$, the variance of the MLE estimator converges to*

$$\frac{j(1+j)^3 n_y(n_y - jn_x)(n_x - jn_y)}{(n_x + n_y)\left((1+j)^2 n_x n_y - j^2(n_x + n_y)^2\right) K} \quad (3)$$

over the randomness of the random hash function.

The bound is a little complicated by the fact that it includes the sizes of the sets n_x and n_y. We note that the bound is convex in the ratio n_y/n_x, giving lower variances than eq. (2) for $n_x << n_y$ or $n_x >> n_y$. Figure 4 shows eq. (3) when the ratio is taken to be *worst possible* as a function of j, as well as when it is taken to be 1. In the symmetric case $n_x = n_y$ the asymptotic variance reduces to

$$\frac{j(1-j)}{K}\frac{(1+j)^3}{2(1+3j)},$$

which is easy to compare with eq. (2) since $\frac{(1+j)^3}{2(1+3j)} \in [\frac{1}{2}, 1]$ for all $j \in [0, 1]$. For $n_x/n_y \neq 1$ the Jaccard similarity is bounded above by $\frac{\min\{n_x,n_y\}}{\max\{n_x,n_y\}}$ which the MLE exploits and discards those higher values from consideration, resulting in 0 variance in that range. For small j eq. (3) is $\frac{n_y j}{n_x+n_y} - O(j^2)$ compared with $j - O(j^2)$ for eq. (2). It makes sense that the variance is lower when $|X|$ is big compared to $|Y|$, since we are given X, but don't know Y.

The global risk of an estimator is defined as the worst possible variance over the parameter space. In the case of the classical MinHash estimator the global risk is $1/(4K)$ at $j = 1/2$. We can also compute the global risk of the MLE estimator, which is $0.1788/K$, 28.5% less than the classical estimator.

Recall the model of the analysis is the same as for classical MinHash: Given X and Y we sample a random hash function $h : [u] \to [u]$. We compute $r = \min_{y \in Y} h(y)$ and $m = \sum_{x \in X} [h(x) < r]$.

Proof of proposition 2. We first find the variance for the MLE estimator for v and then show how to re-parametrize it to use the Jaccard similarity.

Using Stirling's approximation $\log n! = n \log n - n + O(\log n)$, we rewrite the log-likelihood eq. (1) as

$$\begin{aligned}\ell(r;v) &= (n_x - m - [r \in \mathcal{X}] + 1)H(\tfrac{v-[r\in\mathcal{X}]}{n_x-m-[r\in\mathcal{X}]+1}) - (n_x+1)H(\tfrac{v}{n_x+1})\\ &\quad + (u - r - [r \notin \mathcal{X}] - n_x + m + 1)H(\tfrac{n_y-v-[r\notin\mathcal{X}]}{u-r-[r\notin\mathcal{X}]-n_x+m+1})\\ &\quad - (u - n_x + 1)H(\tfrac{n_y-v}{u-n_x+1}) + O(\log u),\end{aligned}$$

where $H(p) = p \log \frac{1}{p} + (1-p)\log\frac{1}{1-p}$ is entropy function.

Standard results [21] on maximum likelihood estimators say that the variance converges to $1/I(v)$ where

$$I(v) = E\left[-\frac{d^2}{dv^2}\ell(r;v)\right]$$

is known as the Fischer information. [2]

We can now evaluate the first two derivatives:

$$\frac{d}{dv}\ell(r;v) = \log\left(\frac{(1-\frac{1}{n_y-v+1})^{[r\notin\mathcal{X}]}(1-\frac{m}{n_x-v+1})}{(1-\frac{1}{v+1})^{[r\in\mathcal{X}]}(1-\frac{r-m}{u-n_x-n_y+v+1})}\right) + O(1/u) \quad (4)$$

$$\begin{aligned}\frac{d^2}{dv^2}\ell(r;v) &= [r \notin \mathcal{X}](\tfrac{1}{n_y-v+1} - \tfrac{1}{n_y-v}) + (\tfrac{1}{n_x-v+1} - \tfrac{1}{n_x-v-m+1})\\ &\quad + [r \in \mathcal{X}](\tfrac{1}{v+1} - \tfrac{1}{v}) + (\tfrac{1}{u-n_x-n_y+v+1} - \tfrac{1}{u-n_x-n_y+v-r+m+1})\\ &\quad + O(1/u^2). \qquad (5)\end{aligned}$$

We now have three terms to bound: $E[r \in \mathcal{X}]$, $\frac{1}{n_x-v-m+1}$ and $\frac{1}{u-n_x-n_y+v-r+m+1}$. Since any element of Y has even chance of becoming the smallest under h, we have

$$E[r \in \mathcal{X}] = \Pr[r \in \mathcal{X}] = \frac{v}{n_y}.$$

When considering the distribution of r and m, we will assume the values of h have exponential distribution, $Exp(1)$, instead of uniform over $[0,1]$. This corresponds to using $\log 1/h(x)$ instead of $h(x)$ which is a strictly monotone transformation and so equivalent in terms of comparing hash values. Let $y^* = \arg\min_{y\in Y} h(y)$ and $h^* = h(y^*)$. (Note this is different from r, which is the rank.) We then have that Then $h^* \sim \mathrm{Exp}(y)$ by stability of minimums of exponentially distributed random variables.

We can now see $m = \sum_{x \in X\setminus Y}[h(x) < h^*]$ as having binomial distribution $B(n_x - v, p)$, conditioning on h^*, where $p = \Pr[h(x) \le h^*] = 1 - \exp(-h^*)$ by the CDF for the exponential distribution. (We only sum over $X \setminus Y$ rather than all of X since no value in $X \cap Y$ can be smaller than h^* by definition.) Because of the binomial revision $\frac{1}{n-i+1}\binom{n}{i} = \frac{1}{n+1}\binom{n+1}{i}$ we can evaluate

$$\begin{aligned}E_m\left[\frac{1}{n_x-v-m+1}\right] &= E_{h^*}\left[\frac{1-p^{n_x-v+1}}{(1-p)(n_x-v+1)}\right]\\ &= \frac{1}{n_y-1}\left(\frac{n_y}{n_x-v+1} - \frac{1}{\binom{n_x+n_y-v}{n_y}}\right),\end{aligned}$$

where the second equality follows by properties of the exponential distribution. Note that the expectation is defined for all $n_y \ge 0$ by limits.[3]

We can similar note that $r - m$ is the number of values in the complement of $X \cup Y$, and so has binomial distribution $B(u - n_x - n_y + v, p)$. By the same arguments as above, we get that

$$E\left[\frac{1}{u-n_x-n_y-v-(r-m)+1}\right] = \frac{1}{n_y-1}\left(\frac{n_y}{u-n_x-n_y+v+1} - \frac{1}{\binom{u-n_x-n_y+v}{y}}\right).$$

Combining all the terms of eq. (5), and assuming n_x and n_y sufficiently large we get the simple result

$$I(v) = \frac{1}{n_y(n_x - v)} + \frac{1}{v(n_y - v)} + O\left(\frac{1}{\min\{n_x, n_y\}}\right).$$

We can now use the re-parametrization formula for Fischer Information to compute

$$I_j(j) = v'(j)^2 I_v(v(j)),$$

where $v(j) = \frac{j}{1+j}(x+y)$. By the previously stated facts on maximum likelihood estimators, this proves the proposition. □

We have succeeded in analysing the variance of the maximum likelihood estimator. There are more questions to ask, such as how large K must be to start seeing convergence to the stated bound. We give some experimental evidence for these questions in section 5.2.

3.3 Minner Estimator

In the previous sections we derived and analysed a Jaccard similarity estimator based on maximum likelihood. The estimator has to evaluate the log-likelihood function, ℓ, for all $v \in [\min\{n_x, n_y\} + 1]$, which means it takes time at least $\Omega(K \min\{n_x, n_y\})$ per database point.

In this section we investigate numerical methods for speeding up the MLE and suggests a new, very fast, estimator we call the Minner Estimator, which can be computed as fast as the classical MinHash estimator.

The starting point of this derivation is the continuous derivative log-likelihood eq. (4), which we would like to solve $= 0$ for v. If we apply the approximation $\log(1-\varepsilon) \approx -\varepsilon$, we we get

$$\frac{d}{dv}\ell(r;v) \approx -\frac{[y^* \notin X]}{n_y - v} - \frac{m}{n_x - v} + \frac{[y^* \in X]}{v} + \frac{r - m}{u - n_x - n_y + v}.$$

This is a convenient form, since it is linear in the variables, $[y^* \in X]$, m and r. As we observe multiply r_i values, we can define $R = \sum_i r_i$, $M = \sum_i m_i$ and $C = \sum_i [y_i^* \in X]$. This gives us a single equation to solve

$$\sum_i \frac{d}{dv}\ell(r_i;v) \approx -\frac{K-C}{n_y - v} - \frac{M}{n_x - v} + \frac{C}{v} + \frac{R - M}{u - n_x - n_y + v} = 0. \quad (6)$$

[2]This is a bit more tricky than it seems, since the standard proof of this fact [21] uses that the expectation is taken over the same probability space as ℓ is defined. However one can check that the key step in which that is used is to show $E[f''/f] = \int (f''(x)/f(x))f(x)dx = \int f''(dx) = (\int f(x)dx)'' = 0$, where $f = \exp(\ell)$ is the probability distribution. Since $E_h[f''/f] = E_h[E_{h_{|\overline{X}}} f''/f] = E_h[0] = 0$ we can run the entire proof using E_h rather than $E_{h_{|\overline{X}}}$ and get the same result. Thus it suffices to thus focus on bounding $E_h[-\frac{d^2}{dv^2}\ell(r;v)]$.

[3]In particular at $y = 1$ it equals $H_{n_x-v+1}/(n_x - v + 1)$, where H_n is the harmonic number.

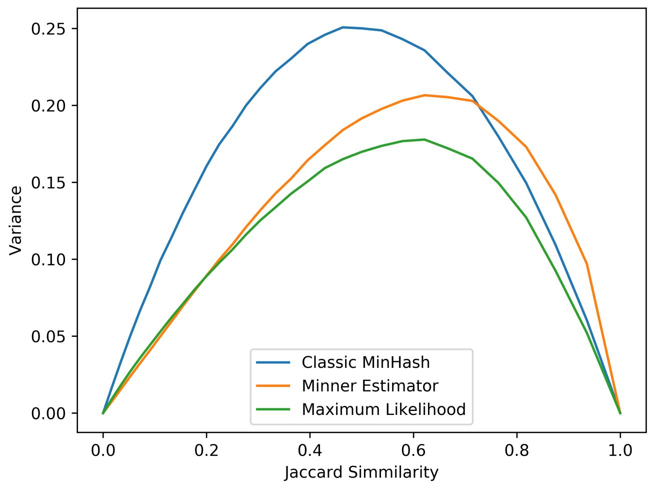

Fig. 5. Measured variance of estimators, over 100,000 repetitions at $|X| = |Y| = K = 30$ and $u = 500$. The MLE has already almost converged to fig. 4. The Minner Estimator is seen to be particular good for low Jaccard similarities, which may be why it works so well on the practical datasets tested in section 5 which tend to have similarities concentrated in the $[0, 0.2]$ range, as seen in fig. 3.

This equation can be rewritten as a degree three polynomial and solved by standard methods. The time complexity has thus been decreased from $\Omega(K \min\{n_x, n_y\})$ to $O(K)$ plus the time it takes to find the polynomial roots.

However, solving a polynomial for every point in the database is hardly as fast as the classical MinHash estimator.

However, we would like a simpler estimator still. In practice u is very large (otherwise it wouldn't be set data), so we'll approximate $\frac{R-M}{u-n_x-n_y+v} \approx 0$. If we assume $n_y >> v$ we may approximate $\frac{K-C}{n_y-v} \approx 0$ we get the simple solution to eq. (6), $v = \frac{Cn_x}{C+M}$. Alternatively, if $n_x >> v$ we approximate $\frac{M}{n_x-v} \approx 0$, we get $v = \frac{Cn_y}{K}$. We then combine the two approximations into the following estimator:

Definition 2 (Minner Estimator).

$$T_v(r) = \min\left\{\frac{Cn_x}{C+M}, \frac{Cn_y}{K}\right\}.$$

The resulting value is clearly always in the acceptable range $[0, \min\{n_x, n_y\}]$, since $C \le C + M$ and $C \le K$. To estimate Jaccard we take $T_j(r) = v/(n_x + n_y - v)$. As before we can compute C and M in $O(K)$ time per database point, and now we have replaced the finalization of finding the roots of a polynomial with a simple division.

While $E[\frac{Cn_y}{K}] = v$ is nice and unbiased (for estimating v), the combined estimator is not necessary so. Using the arguments from the previous analysis section, we find that for a single observation,[4]

$$\begin{aligned} E[\frac{cn_x}{c+m}] &= \frac{vn_x}{y} E[\frac{1}{1+m}] = \frac{vn_x}{n_x - v + 1}(H_{n_x+n_y-v} - H_{n_y-1}) \\ &\approx \frac{vn_x}{n_x - v + 1} \log \frac{n_x + n_y - v}{n_y - 1}, \end{aligned} \tag{7}$$

where H_n is the nth Harmonic Number. If we let $n_x = n_y = n \to \infty$ we get

$$(7) = \frac{2j}{1-j} \log \frac{2}{1+j},$$

a quantity sandwiched between the Jaccard similarity, j, and the Sørensen–Dice coefficient, $\frac{2j}{1+j}$. While not unbiased, this is at least monotone in j (respectively v).

Experimentally, the actual Minner estimator seems to converge to j for larger K and j not too large. That is, the estimator which uses the sums C and M, rather than taking the mean of $\frac{c_i n_x}{c_i+m_i}$, and which makes the minimum with Cn_y/K. For larger Jaccard similarities Minner seems to slightly underestimate Jaccard, just as we see on the variance get worse as $j \to 1$ in fig. 5.

We finally present a numerical way to combine the speed of the Minner Estimator with the consistency of the MLE. The idea is a common one in MLE design, which is to apply Newton's method to the problem of solving $\frac{d}{dv}\ell(r; v) = 0$.[5] To maintain $O(K)$ complexity per database point we apply Newton's method to the approximate derivative equation eq. (6). This is nice, because the second derivative is still linear in C, R and M:

$$\sum_i \frac{d^2}{dv^2}\ell(r_i; v) \approx \frac{K-C}{(n_y-v)^2} + \frac{M}{(n_x-v)^2} + \frac{C}{v^2} + \frac{R-M}{(u-n_x-n_y+v)^2}.$$

Newton's method now proceeds with iterations $v_{i+1} = v_i - \frac{\ell'(r;v_i)}{\ell''(r;v_i)}$.

This concludes the derivation of the Minner Estimator with Newton refinement. In the next section we give pseudo-code matching what was used to perform our experiments on real world data.

4 IMPLEMENTATION DETAILS

While the algorithm is simple conceptually, there are a few tricks to making it run as fast as the classical MinHash sketch.[6]

In algorithm 1 we show how one may use our estimator to do a fast scan through a database. We assume $D_{i,j} \in [0, 1]$ stores the minimum hash value of Y_i under hash function h_j. It is perhaps more typical to have $D_{i,j}$ store the $\arg\min_{y \in Y_i} h(y)$, but in that case one can simply compute $h(D_{i,j})$ at runtime, which is usually a very cheap function. The MinHash survey of Cohen [8] discusses many very efficient ways of storing MinHash values.

In algorithm 1 we start by hashing each element of X under the K hash functions. We sort the resulting values, to be able to perform line 8 and line 9 more efficiently. These lines respectively check if a given Y hash-value is also in X, and counts how many hash-values from X are smaller than the Y hash-value.

[4]We were not able to analyse the case where C and M are the sum of multiple c_i and m_i values, nor the effect of combining the two estimators using the minimum.
[5]People sometimes use Newton's method with the expected Hessian instead of the actual second derivative [16], however in our case we'll be able to efficiently compute it exact.
[6]We don't strive to be faster than the classical estimator for a fixed K, but for a given recall we can be faster, since our K can be smaller.

Algorithm 1 Given a query $X \subseteq [u]$ and a database $D \in [0,1]^{n \times K}$ of sketches of sets $Y_1, \dots \subseteq [u]$, the algorithm estimates the similarity with each Y_i.

```
1: for $j = 1$ to $K$ do
2:     $H_j \leftarrow \text{sorted}(\{h_j(x) : x \in X\})$
3: Initialize a max-heap.
4: for $i = 1$ to $n$ do
5:     $C, M, R \leftarrow 0, 0, 0$
6:     for $i = 1$ to $K$ do
7:         $R \leftarrow R + D_{i,j}$
8:         $C \leftarrow C + [D_{i,j} \in H_j]$    ▷ Done by single table lookup
9:         $M \leftarrow M + \sum_{r \in H_j} [r < D_{i,j}]$
10:    $y \leftarrow |Y_i|$    ▷ Assuming the set sizes are stored
11:    $v \leftarrow \min\{\frac{C|X|}{C+M}, \frac{Cy}{K}\}$    ▷ Minner estimator
12:    for $i = 1$ to Newtons do    ▷ Optional refinement
13:        $\nu \leftarrow \frac{R/u + C/(v+1) - (K-C)/(y-v+1) - M/(|X|-v+1)}{C/(v+1)^2 + (K-C)/(y-v+1)^2 + M/(|X|-v+1)^2}$
14:        $v \leftarrow \min\{\max\{0, v\}, |X|, y\}$
15:    $j \leftarrow v/(|X| + y - v)$    ▷ If Jaccard similarity is required.
16:    Push $(j, i)$ to the max-heap if big enough.
```

There are many ways to perform these tasks. If the range of h is small, say under a million, we can precompute tables. For large ranges, we can use that the H_j's are sorted and binary search, which works for both tasks. In practice, if $|Y|$ is not too much bigger than $|X|$, a linear scan of H_j, will yield the right position in constant time. One can also use one of a plethora of fast prefix sum data structures, which are particularly simple since the values of H_j are uniformly random.

Ideally we would like each value $D_{i,j}$ to take up just 4 bits. If so, one can use the "shuffle" SIMD instruction to perform 16 or more table lookups in a single instruction. This method, common in Product Quantization implementations [?], has the potential to make our estimators as fast as the classical one, even per MinHash value, since the shuffle table fits in registers.

The reduction to 4 bits is possible because the ranks are heavily concentrated around n_y/u, and so have much lower entropy than the direct $\log_2 u$. Using rounding to an exponential set of values, like $\{1, 2, 4, \dots\}$ corresponds to storing just the length of the rank, and provides a good approximation. Another approach is the b-bit Minwise Hashing [15] technique, which stores only the last b bits of the rank.

5 EXPERIMENTS

In this section, we show that our proposed estimators lead to improved performance on maximum Jaccard similarity search on real datasets. All code is available at [?], implemented in Python with critical parts written in Cython for near C level speed.

We run our experiments on the Flickr, DBLP and Netflix dataset from a recent set-similarity join survey by Mann et al. [17]. These datasets were also used by [7] and [?]. In the survey these datasets were identified as archetypical examples with and without Zipf-like behaviour. Mann et al. writes: "Most datasets, like FLICKR (cf. Figure 7), show a Zipf-like distribution and contain a large number of infrequent tokens (less than 10 occurrences), which favors the prefix filter. In contrast, NETFLIX has almost no tokens that occur less than 100 times," In practice this means that the Netflix dataset requires a much larger K, for reasonable recall, than the Flickr dataset, but we still get substantial improvements in recall on both.

5.1 Recall vs Memory usage

We are interested in how the number of hash values per stored set trades with the recall. The number of such values is directly proportional to the memory required to store the dataset.

We focus on the measure recall@10, which means how often the estimated top 10 most similar sets contain the true most similar set. If the searcher wants to obtain the true nearest set, they can use the estimation as a fast first scan over the data, and then compute the true similarities between the query and the 10 candidates. For optimal speed the engineer wants to trade-off K for the number of candidates that need to be "rescored", so recall@30 or 100 may also be interesting. We picked 10 for our experiments because of its standard use in benchmarks [2].

Measuring improvement in terms of increase in recall is a bit tricky, since a 98% to 99% improvement may be much harder to get than a 40% to 44%, even if the later is 4 times bigger in terms of percentage points gained. Instead we opt to measure improvement in terms of how many more MinHash values we would have needed with the classical estimator to obtain the recall of our best estimator. This has the advantage of corresponding directly to space saved.

To measure the improvement in terms of increased K values, we assume a simple model recall@10 $= 1 - \exp(K/a)$ for some constant a depending on the estimator and the dataset. Thus for recall r we need $K = a \log \frac{1}{1-r}$ MinHash Values. For a given recall r the improvement of model a_1 over a_2 in terms of K is thus a factor a_1/a_2.

We perform regressions (with L2 loss) based on this model and the data in tables 1, 2 and 3. For Netflix we get $a_{\text{classic}} = 300.7$ and $a_{\text{best}} = 233.9$ showing an improvement of 28.5% in terms of K values. For Flickr we get $a_{\text{classic}} = 10.73$ and $a_{\text{best}} = 9.644$, corresponding to an 11.3% improvement, and for DBLP we get $a_{\text{classic}} = 204.2$ and $a_{\text{best}} = 168.3$, a 21.4% improvement.

The following shows the effect on recall@10 with 10,000 queries:

These results use the minhash estimator as the initial guess. An alternative is to estimate v as by/K, since $Eb = v/y$. However that appears to be worse, giving recall 1@10 of 0.1718 at newton=1 and $K = 30$. With more newton steps, however, the recall "recovers" to 0.1889 at newton=8. ($n = 2$ and $n = 4$ gives recall 0.1754 and 0.1826 respectively.)

$M_i^T =$

$(M_i)^T$

or $(M^T)_i$

The first table was wrong, because it didn't account for points with equal similarity. The following table fixes that problem:

We can also test on the Flickr Dataset

5.2 Estimation: One way complexity of set similarity

TODO: Say something about fig. 5

Recall@10 on the Netflix dataset					
K	Classic	MLE	Minner	Minner 1N	Minner 8N
1	0.0033	0.0076	**0.0099**	0.0057	0.0063
10	0.0501	0.0396	**0.0623**	0.0506	0.0462
30	0.1474	0.1773	**0.1914**	0.1910	0.1862
100	0.3831	0.48(∗)	0.4640	0.4870	**0.4903**
400	0.7510	0.83(∗)	0.8054	0.8326	**0.8338**
500	0.7942	0.85(∗)	0.8440	0.8660	**0.8667**

Table 1. The Minner estimator is best at small K, but eventually the asymptotics kick in and the Maximum likelihood estimator overtakes. The MLE is very slow however, and one can get most of the benefits by applying a single iteration of Newton's method on top of the Minner Estimator. For the midrange K values 30 and 100 we get a near 30% improvement in recall over the classical estimator. (∗): MLE results for large K were stopped after 2000 queries and 48 hours.

Recall@10 on the Flickr dataset					
K	Classic	MLE	Minner	Minner 1N	Minner 8N
1	0.2379	0.3410	**0.3595**	0.2806	0.2969
5	0.6256	0.5457	**0.6688**	0.5913	0.6138
10	0.7770	0.7155	**0.8122**	0.7327	0.7469
20	0.8657	0.8540	**0.8963**	0.8217	0.8352
30	0.9108	0.9080	**0.9301**	0.8597	0.8714

Table 2. The Flickr dataset is much easier than the Netflix dataset and as such doesn't require as many MinHash values to obtain good recall. The Maximum likelihood estimator never overcomes its asymptotic disadvantage, but the Minner estimator improves 2-7% in recall over the classic estimator, and all of 51% at $K = 1$.

Recall@10 on the DBLP dataset					
K	Classic	MLE	Minner	Minner 1N	Minner 8N
1	0.0036	0.0097	**0.0105**	0.0071	0.0080
10	0.0987	0.0998	0.1057	0.0993	**0.1072**
30	0.2978	0.3438	0.3264	0.3445	**0.3524**
100	0.5736	0.6460	0.6145	0.6519	**0.6536**
400	0.8676	0.90(∗)	0.8932	0.9148	**0.9153**
500	0.9009	0.93(∗)	0.9214	0.9380	**0.9385**

Table 3. The DBLP dataset appears somewhere in between Netflix and Flickr in terms of difficulty. On DBLP the MLE estimator (or Minner with Newton iterations) generally does better than pure Minner. (∗): MLE results for large K were stopped after 2000 queries and 48 hours.

We also perform some experiments on estimation. While this is not the main focus of the paper, we note that the results are consistent with the variance computed for the MLE estimator.

6 CONCLUSION

We have shown that it is possible to substantially improve upon the traditional MinHash estimator in the one-way setting.

There is no clear answer as to which of the new estimators are best. The Minner estimator with newton=8 can sometimes give

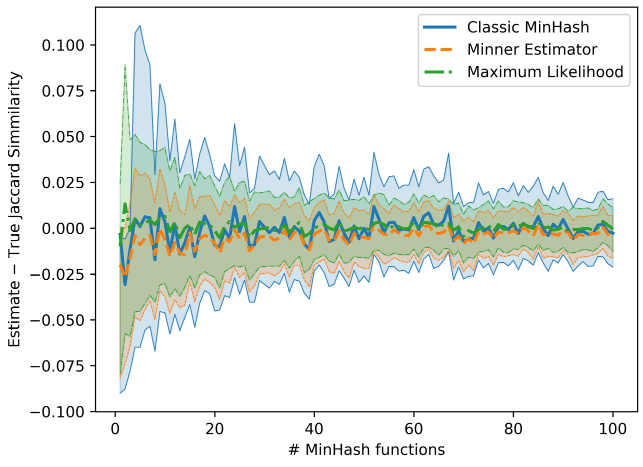

Fig. 6. Mean values with 1 standard deviation bounds (15.9% percentile) on the estimation error for 5 million pairs from the Netflix dataset.

results quite different from MLE, because of the approximation into making it linear.

It's not always clear which

If used as a part of a Machine Learning pipeline, it may be beneficial to supply the minner count in the input.

TODO:

While our estimators only take time $O(k)$ per data point, they still need to do at least two table lookups per value, where the classical estimator just needs to do a single equality comparison. Ideally our algorithm runtime should be dominated by loading data from memory into cache/registers.

One might

Depending on

It particularly makes sense to try and

We don't do any direct speed comparisons.

The fact that our variance is $\frac{n_y}{n_x+n_y}$ of MinHash's variance suggests that our method might be particularly useful for containment queries, in which the

7 OPEN PROBLEMS

Can we also better estimate other types of MinHash?

1. Weighted MinHash
2. b-bit MinHash
3. Bottom-k MinHash
4. Find the best sketch for sets in general. Like how Supermajorities is best for queries.
5. Use a prior better than uniform for Y. One might for example take into account skew distributions, where some tokens are more likely than other.
6. Can we show the Minner Estimator is unbiased? Is seems unbiased, but is it? Can we prove anything else about it, or maybe improve it?

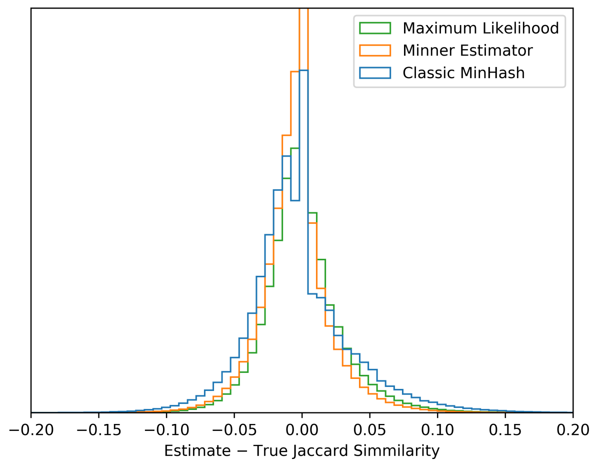

Fig. 7. Density of estimation errors on Netflix dataset at $K = 31$.

REFERENCES

[1] Thomas D. Ahle and Jakob Bæk Tejs Knudsen. 2020. Subsets and Supermajorities: Optimal Hashing-based Set Similarity Search. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 728–739. https://doi.org/10.1109/FOCS46700.2020.00073

[2] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International Conference on Similarity Search and Applications*. Springer, 34–49.

[3] Kenneth RW Brewer, LJ Early, and SF Joyce. 1972. Selecting several samples from a single population. *Australian Journal of Statistics* 14, 3 (1972), 231–239.

[4] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[5] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. 1997. Syntactic clustering of the web. *Computer networks and ISDN systems* 29, 8-13 (1997), 1157–1166.

[6] Tobias Christiani and Rasmus Pagh. 2017. Set similarity search beyond minhash. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 1094–1107.

[7] Tobias Christiani, Rasmus Pagh, and Johan Sivertsen. 2018. Scalable and robust set similarity join. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1240–1243.

[8] Edith Cohen. 2016. Min-Hash Sketches. In *Encyclopedia of Algorithms*. 1282–1287. https://doi.org/10.1007/978-1-4939-2864-4_573

[9] Otmar Ertl. 2017. New cardinality estimation algorithms for HyperLogLog sketches. *CoRR* abs/1702.01284 (2017). arXiv:1702.01284 http://arxiv.org/abs/1702.01284

[10] Otmar Ertl. 2021. SetSketch: Filling the Gap between MinHash and HyperLogLog. *CoRR* abs/2101.00314 (2021). arXiv:2101.00314 https://arxiv.org/abs/2101.00314

[11] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*. Discrete Mathematics and Theoretical Computer Science, 137–156.

[12] Philippe Flajolet and G Nigel Martin. 1985. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences* 31, 2 (1985), 182–209.

[13] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.

[14] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[15] Ping Li and Christian König. 2010. b-Bit minwise hashing. In *Proceedings of the 19th international conference on World wide web*. 671–680.

[16] Nicholas T Longford. 1987. A fast scoring algorithm for maximum likelihood estimation in unbalanced mixed models with nested random effects. *Biometrika* 74, 4 (1987), 817–827.

[17] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. 2016. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment* 9, 9 (2016), 636–647.

[18] Samuel McCauley, Jesper W Mikkelsen, and Rasmus Pagh. 2018. Set similarity search for skewed data. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 63–74.

[19] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. 2016. Mash: fast genome and metagenome distance estimation using MinHash. *Genome biology* 17, 1 (2016), 1–14.

[20] Rasmus Pagh, Morten Stöckel, and David P Woodruff. 2014. Is min-wise hashing optimal for summarizing set intersection?. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 109–120.

[21] Dmitry Panchenko. 2006. Lecture Notes: Properties of MLE: consistency, asymptotic normality. Fisher information. https://ocw.mit.edu/courses/mathematics/18-443-statistics-for-applications-fall-2006/lecture-notes/lecture3.pdf.

[22] Mikkel Thorup. 2013. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 371–380.

[23] Pinghui Wang, Yiyan Qi, Yuanming Zhang, Qiaozhu Zhai, Chenxu Wang, John CS Lui, and Xiaohong Guan. 2019. A memory-efficient sketch method for estimating high similarities in streaming sets. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 25–33.

[24] Yun William Yu and Griffin M Weber. 2020. HyperMinHash: MinHash in LogLog space. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[25] Zijian Zheng, Ron Kohavi, and Llew Mason. 2001. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 401–406.