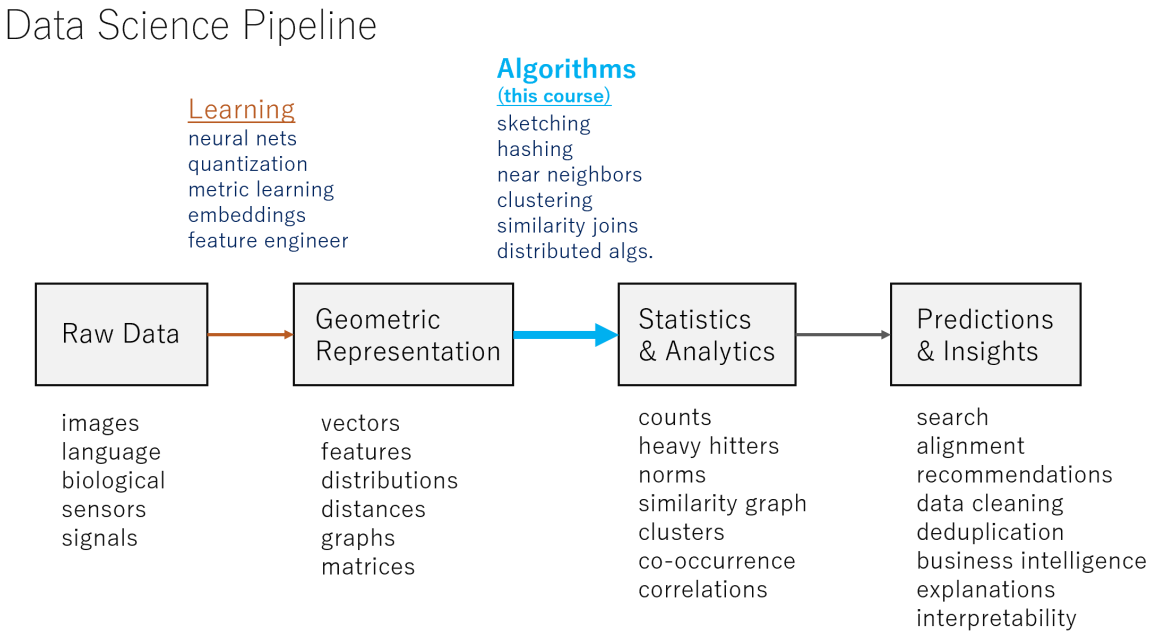
**Data Science Through a Geometric Lens** Fall 2019

# Lecture 01 — September 30, 2019

*Prof. Cyrus Rashtchian* *Topics: Overview; Sketching*




**Overview.** In this lecture, we overview the main topics in the course. We also review basic probability inequalities. And we will provide our first sketching result: Robert Morris's method of counting large numbers in a small register [Mor78].

## 1 Topics Covered in the Course

1. sketching / streaming
2. dimensionality reduction
3. nearest neighbor search
4. clustering
5. distributed algorithms for these problems

The above figure motivates this course by placing the topics in a typical ML or data analytics pipeline. The point is that geometric data arises everywhere, and the many advances in (deep) learning means that the best data representations tend to be geometric (e.g., vectors and distances). After this layer of learning, the task is to quickly process/analyze the resulting data. For huge datasets, this motivates the need for efficient algorithms. We will focus on popular techniques that come with *provable guarantees*. For example, if the pipeline has many steps, we will want some certainty that the intermediate steps will always work very well!

The unifying theme is that all of the course topics deal with algorithms for data with geometric structure. The focus is on approximation algorithms that scale to handle large datasets in modern systems (a.k.a., *big data*). In contrast to standard algorithms courses, we aim to to handle situations where a data set is so large or a computation is so complex that a naive approach would require an impractical amount of memory or time. We will often exploit geometric structure and/or use randomized algorithms to achieve the improved bounds.

## Topic 1: Sketching and Streaming

A central theme will be trading-off accuracy for a reduction in space. This can be thought of as an algorithmic analogue of lossy compression. We study *sketching*, which corresponds to a compression function $C$. This compresses a data set $x$ into a different representation $C(x)$. The aim is to make $C(x)$ as small as possible, while still enabling us to query $f(x)$ when given access to only $C(x)$, as opposed to all of $x$ (e.g., when $x$ is too large to store/process). We will see many examples: approximate counting/histograms, estimating the number of distinct elements, and norm estimation.

There are some things we might want when we are designing such a sketch/compression $C(x)$. Perhaps, we want $f$ to take on two arguments instead of one, as is the case when we want to compute $f(x, y)$ from $C(x)$ and $C(y)$. Often, we want $C(x)$ to be composable. In other words, if $x = x_1x_2x_3 \dots x_n$, we want to be able to compute $C(xx_{n+1})$ using just $C(x)$ and $x_{n+1}$.

A main use of sketching techniques is for *streaming* algorithms (think sensors or daily data like tweets/posts). A stream is a sequence of data elements that comes in bit by bit (or number by number, etc), like items on a conveyor belt. Streaming is the act of processing these data elements *on the fly* as they arrive (as opposed to computing something in batch after loading the whole dataset). The goal of streaming is to answers queries within the constraints of sublinear memory. We can (and will) use sketches to design streaming algorithms.

## Topic 2: Dimensionality Reduction

There are many instances where data sets have high dimensionality. For example, consider spam filtering. A simple approach to spam detection uses a bag-of-words model, where each email is represented as a high-dimensional vector. The indices come from a dictionary of words, and the value at each index is the number of occurrences of that word. In situations like these where we have a high-dimensional computational geometry problem, we may want to reduce the number of dimensions in pre-processing while preserving the approximate geometric structure. This speeds up the eventual algorithm because we can process data using the lower dimensional data.

## Topic 3: Nearest Neighbor Search and Locality Sensitive Hashing

We consider two algorithmic problems for geometric data: nearest neighbor search and clustering. For the first problem, given a query, the goal is to find the nearest point(s) in a dataset to the query. A simple way to do this is a linear scan. If there are $n$ elements in $d$-dimensional space, we can store all of them using $O(nd)$ space. Then, on a query $q$, we can compare $q$ to all $n$ vectors in the dataset. This takes time $O(nd)$. Can we do better?

Yes, if we are okay allowing an approximation. Then, we can improve this to sublinear time by increasing the space a little bit. The main technique will be locality sensitive hashing (LSH). The idea to develop noisy hash functions where the query will be more likely to hash to the same buckets as similar input vectors (and less likely for far/dissimilar inputs). Beyond nearest neighbor search,

LSH-related ideas are prevalent and hugely important in many many practical applications (image search, AR/VR, NLP, bioinformatics, etc).

### Topic 4: Clustering

Clustering is another fundamental geometric problem. The goal is to find a small number of clusters that summarize or group the data into meaningful subsets. This is often hard to define precisely, and a good clustering will often depend on the data or the domain. We will consider multiple ways of doing this, with different objectives and guarantees. For example $k$-means, $k$-median, and $k$-center all will find $k$ center points to minimize some cost function. This cost will change, depending on how we measure the distance.

The bad part about clustering is that most objectives are NP-Hard. The good (and mysterious) part is that people use heuristics all the time (in all areas of science) and things mostly work out.

### Topic 5: Distributed Algorithms

If a data set is really large, it may not be possible to store or process all the data at the same time using only one processor. One reason is that algorithms are very slow if the data is not in main memory (RAM/Cache). So it is often better to use a large number of processors/cores, either with a cluster of CPUs or GPUs. But this makes the algorithm design process more complicated — and more interesting! We consider distributed algorithms for geometric problems, such as similarity search and clustering. A nice aspect of algorithms in this area is that there is a common framework to design/analyze algorithms. It is based on MapReduce. Even though the specifics of systems matter, there are also basic metrics that determine the quality/speed of an algorithm. For example, we will consider the total communication among processors, and we will trade this off with the amount of work that each processor must do.

## 2 Approximate Counting

We will now discuss our first detailed example of a sketching algorithm. In the following, we discuss a problem first studied by Robert Morris [Mor78]. This is essentially the first streaming paper, from 1978, way before big data was a thing. The motivation for him was to understand space-bounded devices (back when space/memory was expensive).

**Problem.** Design an algorithm that monitors a sequence of events and upon request can output (an estimate of) the number of events thus far. More formally, create a data structure that maintains a single integer $n$ and supports the following operations.

- `init();` sets $n \leftarrow 0$
- `update();` increments $n \leftarrow n + 1$
- `query();` outputs $n$ or an estimate of $n$

A trivial algorithm stores $n$ as a sequence of $\lceil \log n \rceil = O(\log n)$ bits (a counter).

**Question.** Can you solve this problem using fewer than $\log n$ bits??

**Nope.** If we want `query();` to return the exact value of $n$, this is the best we can do. Suppose for some such algorithm, we use $f(n)$ bits to store the integer $n$. There are $2^{f(n)}$ configurations for these bits. In order for the algorithm to be able to store the exact value of all integers up to $n$, the number of configurations must be greater than or equal to the number $n$. Hence,

$$2^{f(n)} \geq n \Rightarrow f(n) \geq \log n.$$

**Approximate Solution.** The goal is to use much less space than $O(\log n)$, and so we must instead answer `query();` with some estimate $\tilde{n}$ of $n$. We would like this $\tilde{n}$ to satisfy

$$\Pr(|\tilde{n} - n| > \varepsilon n) < \delta, \tag{1}$$

for some $0 < \varepsilon, \delta < 1$ that are given to the algorithm up front. For example, get the answer to a factor of $\varepsilon = 1/10$ with probability 90%.

Morris's algorithm provides such an estimator for some $\varepsilon, \delta$ that we will analyze shortly. We assume the algorithm has a perfect source of randomness. Then, the algorithm works as follows:

- `init();` sets $X \leftarrow 0$
- `update();` increments $X$ with probability $2^{-X}$
- `query();` outputs $\tilde{n} = 2^X - 1$

Intuitively, the variable $X$ is attempting to store a value that is approximately $\log_2 n$. Before giving a rigorous analysis in Section 4, we first give a probability review.

# 3 Probability Review

We are mainly discussing discrete random variables. For this section we consider random variables as taking values in some set $S \subset \mathbb{R}$. Recall the expectation of $X$ is defined $\mathbb{E}X = \sum_{j \in S} j \cdot \Pr(X = j)$. We now state a few basic lemmas and facts without proof. For more details (and proofs and examples), there are many excellent resources on Wikipedia and Google about these inequalities.

**Lemma 1** (Linearity of expectation)**.** *For all reals $a$ and $b$ and random variables $X$ and $Y$,*

$$\mathbb{E}(aX + bY) = a \cdot \mathbb{E}X + b \cdot \mathbb{E}Y.$$

**Lemma 2** (Markov)**.** *If $X$ is a nonnegative random variable, then for all $\lambda > 0$,*

$$\Pr(X \geq \lambda) \leq \frac{\mathbb{E}X}{\lambda}$$

**Lemma 3** (Chebyshev)**.** *Under the same conditions as Markov,*

$$\Pr(|X - \mathbb{E}X| \geq \lambda) \leq \frac{\mathbb{E}(X - \mathbb{E}X)^2}{\lambda^2} = \frac{\mathrm{Var}[X]}{\lambda^2}$$

*Proof.* Observe that

$$\Pr(|X - \mathbb{E}X| \geq \lambda) = \Pr((X - \mathbb{E}X)^2 > \lambda^2).$$

The claim follows by Markov's inequality. □

Also note that

$$\Pr(|X - \mathbb{E}X| \geq \lambda) = \Pr(|X - \mathbb{E}X|^p > \lambda^p)$$

for all $p \geq 1$. If we apply Markov's inequality to this statement, we get a more general version of Chebyshev's inequality:

$$\Pr(|X - \mathbb{E}X| \geq \lambda) \leq \frac{\mathbb{E}|X - \mathbb{E}X|^p}{\lambda^p}.$$

**Lemma 4** (Chernoff)**.** *Suppose $X_1, X_2, \ldots, X_n$ are independent random variables with $X_i \in [0,1]$. Let $X = \sum_{i=1}^n X_i$ with $\mu = \mathbb{E}X$. If $0 < \epsilon < 1$, then*

$$\Pr(|X - \mathbb{E}X| > \varepsilon\mu) \leq 2 \cdot e^{-\varepsilon^2\mu/3}.$$

*Proof.* We will prove a weaker statement: we assume that the $X_i$ are independent Bernoulli. Let $X_i = 1$ with probability $p_i$ and $X_i = 0$ otherwise. Note then $\mu = \sum_{i=1}^n p_i$. Furthermore, we also do not attempt to achieve the constant $1/3$ in the exponent, but are happy to settle for any fixed constant. For the upper tail, we have for any $t > 0$ that

$$\begin{aligned}
\Pr(X > (1+\varepsilon)\mu) &= \Pr\left(e^{tX} > e^{t(1+\varepsilon)\mu}\right) \\
&\leq e^{-t(1+\varepsilon)\mu} \cdot \mathbb{E}e^{tX} \text{ (Markov)} \\
&= e^{-t(1+\varepsilon)\mu} \cdot \mathbb{E}e^{\sum_i tX_i} \\
&= e^{-t(1+\varepsilon)\mu} \cdot \mathbb{E}\prod_i e^{tX_i} \\
&= e^{-t(1+\varepsilon)\mu} \cdot \prod_i \mathbb{E}e^{tX_i} \text{ (independence)} \\
&= e^{-t(1+\varepsilon)\mu} \cdot \prod_i (1 - p_i + p_i e^t) \\
&= e^{-t(1+\varepsilon)\mu} \cdot \prod_i (1 + p_i(e^t - 1)) \\
&\leq e^{-t(1+\varepsilon)\mu} \cdot \prod_i e^{p_i(e^t-1)} \text{ (since } 1 + x \leq e^x) \\
&= e^{-t(1+\varepsilon)\mu + (e^t-1)\mu} \\
&= \left(\frac{e^\varepsilon}{(1+\varepsilon)^{1+\varepsilon}}\right)^\mu \text{ (set } t = \ln(1+\varepsilon)) \qquad (2)
\end{aligned}$$

There are two regimes of interest for Eqn. (2). When $\varepsilon < 2$, by Taylor's theorem we have $\ln(1+\varepsilon) = \varepsilon - \varepsilon^2/2 + O(\varepsilon^3)$. Then replacing $(1+\varepsilon)^{1+\varepsilon}$ by $e^{(1+\varepsilon)\ln(1+\varepsilon)}$ and applying Taylor's theorem, this leads to (2) being $\approx e^{-\Theta(\varepsilon^2\mu)}$ in that regime. When $\varepsilon > 2$ we have $\ln(1+\varepsilon) = \Theta(\ln(\varepsilon))$, in which case the tail bounds becomes $\varepsilon^{-\Theta(\varepsilon\mu)}$. The lower tail analysis for $\Pr(X < (1-\varepsilon)\mu)$ is similar, noting that $X < (1-\varepsilon)\mu$ iff $e^{-tX} > e^{-t(1-\varepsilon)\mu}$. We then apply Markov then analyze $\mathbb{E}e^{-tX}$ and eventually set $t = -\ln(1-\varepsilon)$. The right hand side then becomes $(e^{-\varepsilon}/(1-\varepsilon)^{1-\varepsilon})^\mu$. □

# 4 Analysis of Morris's Algorithm

Let $X_n$ denote $X$ in Morris's algorithm after $n$ updates. Let $\widetilde{n} = 2^{X_n} - 1$ be the output.

We first analyze the expectation, then the variance, and then use the concentration bounds from above to provide the overall analysis.

**Claim 5.** *For Morris's algorithm,* $\mathbb{E}2^{X_n} = n + 1$.

*Proof.* We will prove by induction. Consider the base case where $n = 0$. We have initialized $X \leftarrow 0$ and have yet to increment it. Thus, $X_n = 0$, and $\mathbb{E}2^{X_n} = n + 1$. Now suppose that $\mathbb{E}2^{X_n} = n + 1$ for some fixed $n$.

We have

$$\begin{aligned}
\mathbb{E}2^{X_{n+1}} &= \sum_{j=0}^{\infty} \Pr(X_n = j) \cdot \mathbb{E}(2^{X_{n+1}} \mid X_n = j) \\
&= \sum_{j=0}^{\infty} \Pr(X_n = j) \cdot \left(2^j \left(1 - \frac{1}{2^j}\right) + \frac{1}{2^j} \cdot 2^{j+1}\right) \\
&= \sum_{j=0}^{\infty} \Pr(X_n = j) 2^j + \sum_{j=0}^{\infty} \Pr(X_n = j) \\
&= \mathbb{E}2^{X_n} + 1 \\
&= (n + 1) + 1.
\end{aligned}$$

This completes the inductive step. □

It is now clear why we output our estimate of $n$ as $\tilde{n} = 2^X - 1$: it is an unbiased estimator of $n$. Moreover, since $X \approx \log n$, the expected amount of space we use is $O(\log \log n)$.

In order to show (1) however, we will also control on the variance of our estimator. This is because, by Chebyshev's inequality,

$$\Pr(|\tilde{n} - n| > \varepsilon n) < \frac{1}{\varepsilon^2 n^2} \cdot \mathbb{E}(\tilde{n} - n)^2 = \frac{1}{\varepsilon^2 n^2} \cdot \mathbb{E}(2^{X_n} - 1 - n)^2.$$

When we expand the above square, we find that we need to control $\mathbb{E}2^{2X_n}$. The proof of the following claim is by induction, similar to that of Claim 5.

**Claim 6.** *For Morris's algorithm, we have*

$$\mathbb{E}2^{2X_n} = \frac{3}{2}n^2 + \frac{3}{2}n + 1. \tag{3}$$

*Proof.* We again prove this by induction. It is clearly true for $n = 0$. Then

$$\begin{aligned}
\mathbb{E}2^{2X_{n+1}} &= \sum_{j=0}^{\infty} \Pr(2^{X_n} = j) \cdot \mathbb{E}(2^{2X_{n+1}} \mid 2^{X_n} = j) \\
&= \sum_{j=0}^{\infty} \Pr(2^{X_n} = j) \cdot \left(\frac{1}{j} \cdot 4j^2 + \left(1 - \frac{1}{j}\right) \cdot j^2\right) \\
&= \sum_{j=0}^{\infty} \Pr(2^{X_n} = j) \cdot (j^2 + 3j) \\
&= \mathbb{E}2^{2X_n} + 3 \cdot \mathbb{E}2^{X_n} \\
&= \left(\frac{3}{2}n^2 + \frac{3}{2}n + 1\right) + (3n + 3) \\
&= \frac{3}{2}(n + 1)^2 + \frac{3}{2}(n + 1) + 1
\end{aligned}$$

This completes the inductive step. □

**Bounding the failure probability.** Now note $\mathrm{Var}[Z]$ in general is equal to $\mathbb{E}Z^2 - (\mathbb{E}Z)^2$. Also, $\mathrm{Var}[2^{X_n} - 1] = \mathrm{Var}[2^{X_n}]$. These together imply that

$$\mathrm{Var}[2^{X_n}] = \mathbb{E}[2^{2X_n}] - (\mathbb{E}[2^{2X_n}])^2 = \frac{3}{2}n^2 + \frac{3}{2}n + 1 - (n+1)^2 = \frac{1}{2}n^2 - \frac{1}{2}n < \frac{1}{2}n^2$$

and thus

$$\Pr(|\tilde{n} - n| > \varepsilon n) < \frac{1}{\varepsilon^2 n^2} \cdot \frac{n^2}{2} = \frac{1}{2\varepsilon^2},$$

which is not particularly meaningful since the right hand side is only smaller than 1 when $\varepsilon > 1/\sqrt{2}$, and otherwise it says nothing. But we really want it to work for any $\varepsilon$.

## 4.1 Morris+

To decrease the failure probability of Morris's basic algorithm, we instantiate $s$ independent copies of Morris's algorithm and average their outputs. That is, we obtain independent estimators $\tilde{n}_1, \ldots, \tilde{n}_s$ from independent instantiations of Morris's algorithm, and our output to a query is

$$\tilde{n}^+ = \frac{1}{s} \cdot \sum_{i=1}^{s} \tilde{n}_i$$

Since each $\tilde{n}_i$ is an unbiased estimator of $n$, so is their average. Furthermore, since variances of independent random variables add, and multiplying a random variable by some constant $c = 1/s$ causes the variance to be multiplied by $c^2$, the right hand side of (4) becomes

$$\Pr(|\tilde{n}^+ - n| > \varepsilon n) < \frac{1}{2s\varepsilon^2} < \delta$$

for $s = 1/(2\varepsilon^2\delta) = \Theta(1/(\varepsilon^2\delta))$. This is pretty good, but we can do even better.

## 4.2 Morris++

There is a simple technique to reduce the dependence on the failure probability $\delta$ from $1/\delta$ down to $\log(1/\delta)$. This method is known as a **median-of-means** estimator. The technique is as follows.

We run $t$ instantiations of Morris+, which we denote $\tilde{n}_1^+, \tilde{n}_2^+, \ldots \tilde{n}_t^+$. For each, we will achieve failure probability $\frac{1}{3}$ by taking the mean of $s = \Theta(1/\varepsilon^2)$ Morris estimators. We then output the median estimate from all the $t$ Morris+ instantiations.

$$\tilde{n}^{++} = \mathrm{median}(\tilde{n}_1^+, \tilde{n}_2^+, \ldots \tilde{n}_t^+).$$

We can calculate the expected space usage. Each Morris run takes space $O(\log\log n)$ in expectation. Each Morris+ run uses $s \approx 1/\varepsilon^2$ copies of Morris, leading to space roughly $\log\log(n)/\varepsilon^2$ for each. We will see shortly that there are $t \approx \log(1/\delta)$ copies of Morris+ in the overall Morris++ algorithm. Therefore, we have that the expected space of Morris++ will roughly be

$$s \cdot t \cdot \log\log n \;\simeq\; \frac{\log\log n}{\varepsilon^2} \cdot \log(1/\delta).$$

**Analysis.** Say that the $i$th Morris+ estimate **succeeds** if $|\tilde{n}^+ - n| < \varepsilon n$, and otherwise it fails.

The expected number of Morris+ instantiations that succeed is at least $2t/3$. For the median to be a bad estimate, less than half the Morris+ instantiations can succeed (or more than half must fail). This implies that number of succeeding instantiations deviated from its expectation by at least $t/6$.

To analyze Morris++, we define the following $t$ indicator variables:

$$Y_i = \begin{cases} 1, & \text{if the } i\text{-th Morris+ instantiation succeeds.} \\ 0, & \text{otherwise.} \end{cases}$$

Then by the Chernoff bound,

$$\Pr\left(\sum_{i=1}^{t} Y_i \le \frac{t}{2}\right) \le \Pr\left(\left|\sum_{i=1}^{t} Y_i - \mathbb{E}\sum_{i=1}^{t} Y_i\right| \ge \frac{t}{6}\right) \le 2e^{-ct} < \delta,$$

for a constant $c$, where $c = 1/48$ seems to work. The final inequality "$< \delta$" holds by setting the number of estimators to be $t = \Theta(\log(1/\delta))$.

This implies that $|\tilde{n}^{++} - n| < \varepsilon n$ with probability at least $1 - \delta$, as desired.

**Overall space complexity.** Note the space is a ranadom variable. We will not show it here, but one can show that the total space complexity is, with probability $1 - \delta$, at most

$$O(\varepsilon^{-2} \log(1/\delta)(\log\log(n/(\varepsilon\delta))))$$

bits. In particular, for constant $\varepsilon, \delta$ (say each $1/100$), the total space complexity is $O(\log\log n)$ with constant probability. This is exponentially better than the $\log n$ space achieved by storing a counter.

**An improvement.** One issue with the above is that the space is $\Omega(\varepsilon^{-2}\log\log n)$ for $(1+\varepsilon)$-approximation, but the obvious lower bound is only $O(\log(\log_{1+\varepsilon} n)) = O(\log(1/\varepsilon) + \log\log n)$. This can actually be achieved. Instead of incrementing the counter with probability $1/2^X$, we do it with probability $1/(1+a)^X$ and choose $a > 0$ appropriately. We leave it to the reader as an exercise to find the appropriate value of $a$ and to figure out how to answer queries.

# References

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 10 1978.