7 Algorithms for Massive Data Problems

Massive Data, Sampling

This chapter deals with massive data problems where the input data (a graph, a matrix or some other object) is too large to be stored in random access memory. One model for such problems is the streaming model, where the data can be seen only once. In the streaming model, the natural technique to deal with the massive data is sampling. Sampling is done "on the fly". As each piece of data is seen, based on a coin toss, one decides whether to include the data in the sample. Typically, the probability of including the data point in the sample may depend on its value. Models allowing multiple passes through the data are also useful; but the number of passes needs to be small. We always assume that random access memory (RAM) is limited, so the entire data cannot be stored in RAM.

To introduce the basic flavor of sampling on the fly, consider the following primitive. From a stream of n positive real numbers $a_1, a_2, \ldots, a_n$, draw a sample element a_i so that the probability of picking an element is proportional to its value. It is easy to see that the following sampling method works. Upon seeing $a_1, a_2, \ldots, a_i$, keep track of the sum $a = a_1 + a_2 + \cdots + a_i$ and a sample a_j, $j \leq i$, drawn with probability proportional to its value. On seeing a_{i+1}, replace the current sample by a_{i+1} with probability $\frac{a_{i+1}}{a+a_{i+1}}$ and update a.

7.1 Frequency Moments of Data Streams

An important class of problems concerns the frequency moments of data streams. Here a data stream $a_1, a_2, \ldots, a_n$ of length n consists of symbols a_i from an alphabet of m possible symbols which for convenience we denote as $\{1, 2, \ldots, m\}$. Throughout this section, n, m, and a_i will have these meanings and s (for symbol) will denote a generic element of $\{1, 2, \ldots, m\}$. The frequency f_s of the symbol s is the number of occurrences of s in the stream. For a nonnegative integer p, the p^{th} frequency moment of the stream is

$$\sum_{s=1}^{m} f_s^p.$$

Note that the $p = 0$ frequency moment corresponds to the number of distinct symbols occurring in the stream. The first frequency moment is just n, the length of the string. The second frequency moment, $\sum_s f_s^2$, is useful in computing the variance of the stream.

$$\frac{1}{m}\sum_{s=1}^{m}\left(f_s - \frac{n}{m}\right)^2 = \frac{1}{m}\sum_{s=1}^{m}\left(f_s^2 - 2\frac{n}{m}f_s + \left(\frac{n}{m}\right)^2\right) = \frac{1}{m}\sum_{s=1}^{m} f_s^2 - \frac{n^2}{m^2}$$

In the limit as p becomes large, $\left(\sum_{s=1}^{m} f_s^p\right)^{1/p}$ is the frequency of the most frequent element(s).

We will describe sampling based algorithms to compute these quantities for streaming data shortly. But first a note on the motivation for these various problems. The identity and frequency of the the most frequent item or more generally, items whose frequency exceeds a fraction of n, is clearly important in many applications. If the items are packets on a network with source and destination addresses, the high frequency items identify the heavy bandwidth users. If the data is purchase records in a supermarket, the high frequency items are the best-selling items. Determining the number of distinct symbols is the abstract version of determining such things as the number of accounts, web users, or credit card holders. The second moment and variance are useful in networking as well as in database and other applications. Large amounts of network log data are generated by routers that can record the source address, destination address, and the number of packets for all the messages passing through them. This massive data cannot be easily sorted or aggregated into totals for each source/destination. But it is important to know if some popular source-destination pairs have a lot of traffic for which the variance is the natural measure.

7.1.1 Number of Distinct Elements in a Data Stream

Consider a sequence $a_1, a_2, \ldots, a_n$ of n elements, each a_i an integer in the range 1 to m where n and m are very large. Suppose we wish to determine the number of distinct a_i in the sequence. Each a_i might represent a credit card number extracted from a sequence of credit card transactions and we wish to determine how many distinct credit card accounts there are. The model is a data stream where symbols are seen one at a time. We first show that any deterministic algorithm that determines the number of distinct elements exactly must use at least m bits of memory.

Lower bound on memory for exact deterministic algorithm

Suppose we have seen the first $k \geq m$ symbols. The set of distinct symbols seen so far could be any of the 2^m subsets of $\{1, 2, \ldots, m\}$. Each subset must result in a different state for our algorithm and hence m bits of memory are required. To see this, suppose first that two different size subsets of distinct symbols lead to the same internal state. Then our algorithm would produce the same count of distinct symbols for both inputs, clearly an error for one of the input sequences. If two sequences with the same number of distinct elements but different subsets lead to the same state, then on next seeing a symbol that appeared in one sequence but not the other would result in subsets of different size and thus require different states.

Algorithm for the Number of distinct elements

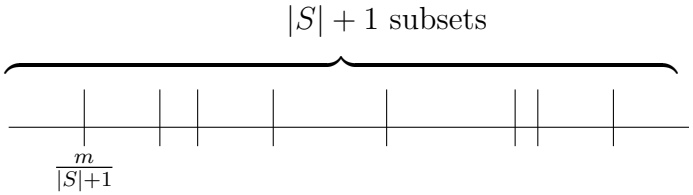

Figure 7.1: Estimating the size of S from the minimum element in S which has value approximately $\frac{m}{|S|+1}$. The elements of S partition the set $\{1, 2, \ldots, m\}$ into $|S|+1$ subsets each of size approximately $\frac{m}{|S|+1}$.

Let $a_1, a_2, \ldots, a_n$ be a sequence of elements where each $a_i \in \{1, 2, \ldots, m\}$. The number of distinct elements can be estimated with $O(\log m)$ space. Let $S \subseteq \{1, 2, \ldots, m\}$ be the set of elements that appear in the sequence. Suppose that the elements of S were selected uniformly at random from $\{1, 2, \ldots, m\}$. Let *min* denote the minimum element of S. Knowing the minimum element of S allows us to estimate the size of S. The elements of S partition the set $\{1, 2, \ldots, m\}$ into $|S|+1$ subsets each of size approximately $\frac{m}{|S|+1}$. See Figure **??**. Thus, the minimum element of S should have value close to $\frac{m}{|S|+1}$. Solving $min = \frac{m}{|S|+1}$ yields $|S| = \frac{m}{min} - 1$. Since we can determine *min*, this gives us an estimate of $|S|$.

The above analysis required that the elements of S were picked uniformly at random from $\{1, 2, \ldots, m\}$. This is generally not the case when we have a sequence $a_1, a_2, \ldots, a_n$ of elements from $\{1, 2, \ldots, m\}$. Clearly if the elements of S were obtained by selecting the $|S|$ smallest elements of $\{1, 2, \ldots, m\}$, the above technique would give the wrong answer. If the elements are not picked uniformly at random, can we estimate the number of distinct elements? The way to solve this problem is to use a hash function h where

$$h : \{1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, M-1\}.$$

To count the number of distinct elements in the input, count the number of elements in the mapped set $\{h(a_1), h(a_2), \ldots\}$. The point being that $\{h(a_1), h(a_2), \ldots\}$ behaves like a random subset so the above heuristic argument using the minimum to estimate the number of elements applies. If we needed $h(a_1), h(a_2), \ldots$ to be completely independent, the space needed to store the hash function would too high. Fortunately, only 2-way independence is needed. We recall the formal definition of 2-way independence below. But first recall that a hash function is always chosen at random from a family of hash functions and phrases like "probability of collision" refer to the probability in the choice of hash function.

Universal Hash Functions

A set of hash functions

$$H = \{h \mid h : \{1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, M-1\}\}$$

is *2-universal* if for all x and y in $\{1, 2, \ldots, m\}$, $x \neq y$, and for all z and w in $\{0, 1, 2, \ldots, M-1\}$

$$\text{Prob}\big(h(x) = z \text{ and } h(y) = w\big) = \tfrac{1}{M^2}$$

for a randomly chosen h. The concept of a 2-universal family of hash functions is that given x, $h(x)$ is equally likely to be any element of $\{0, 1, 2, \ldots, M-1\}$ and for $x \neq y$, $h(x)$ and $h(y)$ are independent.

We now give an example of a 2-universal family of hash functions. For simplicity let M be a prime. For each pair of integers a and b in the range [0,M-1], define a hash function

$$h_{ab}(x) = ax + b \mod (M)$$

To store the hash function h_{ab}, store the two integers a and b. This requires only $O(\log M)$ space. To see that the family is 2-universal note that $h(x) = z$ and $h(y) = w$ if and only if

$$\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} z \\ w \end{pmatrix} \mod (M)$$

If $x \neq y$, the matrix $\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}$ is invertible modulo M and there is only one solution for a and b. Thus, for a and b chosen uniformly at random, the probability of the equation holding is exactly $\frac{1}{M^2}$.

Analysis of distinct element counting algorithm

Let $b_1, b_2, \ldots, b_d$ be the distinct values that appear in the input. Then $S = \{h(b_1), h(b_2), \ldots h(b_d)\}$ is a set of d random and 2-way independent values from the set $\{0, 1, 2, \ldots, M-1\}$. We now show that $\frac{M}{\min}$ is a good estimate for d, the number of distinct elements in the input, where min=min(S).

Lemma 7.1 *Assume M >100d. With probability at least $\frac{2}{3}$, $\frac{d}{6} \leq \frac{M}{\min} \leq 6d$, where min is the smallest element of S.*

Proof: First, we show that Prob $\left(\frac{M}{\min} > 6d\right) < \frac{1}{6}$.

$$\text{Prob}\left(\frac{M}{\min} > 6d\right) = \text{Prob}\left(\min < \frac{M}{6d}\right) = \text{Prob}\left(\exists k,\ h(b_k) < \frac{M}{6d}\right)$$
$$\leq \left(\sum_{i=1}^{d} \text{Prob}\big(h(b_i) \leq M/6d\big)\right) \leq d/(6d) = 1/6.$$

Finally, we show that $\text{Prob}\left(\frac{M}{\min} < \frac{d}{6}\right) < \frac{1}{6}$. $\text{Prob}\left(\frac{M}{\min} < \frac{d}{6}\right) = \text{Prob}\left[\min > \frac{6M}{d}\right] = \text{Prob}\left(\forall k,\ h(b_k) > \frac{6M}{d}\right)$ For $i = 1, 2, \ldots, d$ define the indicator variable $y_i = \begin{cases} 0 & \text{if } h(b_i) > \frac{6M}{d} \\ 1 & \text{otherwise} \end{cases}$ and let $y = \sum_{i=1}^{d} y_i$. Now $\text{Prob}(y_i = 1) \geq \frac{6}{d}$, $E(y_i) \geq \frac{6}{d}$, and $E(y) = 6$. For 2-way independent random variables, the variance of their sum is the sum of their variances. So $\text{Var}(y) = d\text{Var}(y_1)$. Further, it is easy to see since y_1 is 0 or 1 that $\text{Var}(y_1) = E\left[(y_1 - E(y_1))^2\right] = E(y_1^2) - E^2(y_1) = E(y_1) - E^2(y_1) \leq E(y_1)$. Thus $\text{Var}(y) \leq E(y)$. Now by the Chebychev inequality,

$$\begin{aligned} \text{Prob}\left(\frac{M}{\min} < \frac{d}{6}\right) &= \text{Prob}\left(\min > \tfrac{6M}{d}\right) = \text{Prob}\left(\forall k\ h(b_i) > \frac{6M}{d}\right) \\ &= \text{Prob}(y = 0) \leq \text{Prob}(|y - E(y)| \geq E(y)) \\ &\leq \frac{\text{Var}(y)}{E^2(y)} \leq \frac{1}{E(y)} \leq \frac{1}{6} \end{aligned}$$

Since $\frac{M}{\min} > 6d$ with probability at most $\frac{1}{6}$ and $\frac{M}{\min} < \frac{d}{6}$ with probability at most $\frac{1}{6}$, $\frac{d}{6} \leq \frac{M}{\min} \leq 6d$ with probability at least $\frac{2}{3}$. ∎

7.1.2 Counting the Number of Occurrences of a Given Element.

To count the number of occurrences of an element in a stream requires at most $\log n$ space where n is the length of the stream. Clearly, for any length stream that occurs in practice, we can afford $\log n$ space. For this reason, the following material may never be used in practice, but the technique is interesting and may give insight into how to solve some other problem.

Consider a string of 0's and 1's of length n in which we wish to count the number of occurrences of 1's. Clearly if we had $\log n$ bits of memory we could keep track of the exact number of 1's. However, we can approximate the number with only $\log \log n$ bits.

Let m be the number of 1's that occur in the sequence. Keep a value k such that 2^k is approximately the number of occurrences m. Storing k requires only $\log \log n$ bits of memory. The algorithm works as follows. Start with k=0. For each occurrence of a 1, add one to k with probability $1/2^k$. At the end of the string, the quantity $2^k - 1$ is the estimate of m. To obtain a coin that comes down heads with probability $1/2^k$, flip a fair coin, one that comes down heads with probability ½, k times and report heads if the fair coin comes down heads in all k flips.

Given k, on average it will take 2^k ones before k is incremented. Thus, the expected number of 1's to produce the current value of k is $1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1$.

7.1.3 Counting Frequent Elements

The Majority and Frequent Algorithms

First consider the very simple problem of n people voting. There are m candidates, $\{1, 2, \ldots, m\}$. We want to determine if one candidate gets a majority vote and if so who. Formally, we are given a stream of integers $a_1, a_2, \ldots, a_n$, each a_i belonging to $\{1, 2, \ldots, m\}$, and want to determine whether there is some $s \in \{1, 2, \ldots, m\}$ which occurs more than $n/2$ times and if so which s. It is easy to see that to solve the problem exactly on read only once streaming data with a deterministic algorithm, requires $\Omega(n)$ space. Suppose n is even and the first $n/2$ items are all distinct and the last $n/2$ items are identical. After reading the first $n/2$ items, we need to remember exactly which elements of $\{1, 2, \ldots, m\}$ have occurred. If for two different sets of elements occurring in the first half of the stream, the contents of the memory are the same, then a mistake would occur if the second half of the stream consists solely of an element that is in one set, but not in the other. Thus, $\log_2 \binom{m}{n/2}$ bits of memory, which if $m > n$ is $\Omega(n)$, are needed.

The following is a simple low-space algorithm that always finds the majority vote if there is one. If there is no majority vote, the output may be arbitrary. That is, there may be "false positives", but no "false negatives".

Majority Algorithm

> Store a_1 and initialized a counter to one. For each subsequent a_i, if a_i is the same as the currently stored item, increment the counter by one. If it differs, decrement the counter by one provided the counter is nonzero. If the counter is zero, then store a_i and set the counter to one.

To analyze the algorithm, it is convenient to view the decrement counter step as "eliminating" two items, the new one and the one that caused the last increment in the counter. It is easy to see that if there is a majority element s, it must be stored at the end. If not, each occurrence of s was eliminated; but each such elimination also causes another item to be eliminated and so for a majority item not to be stored at the end, we must have eliminated more than n items, a contradiction.

Next we modify the above algorithm so that not just the majority, but also items with frequency above some threshold are detected. We will also ensure that there are no false positives as well as no false negatives. Indeed the algorithm below will find the frequency (number of occurrences) of each element of $\{1, 2, \ldots, m\}$ to within an additive term of $\frac{n}{k+1}$ using $O(k \log n)$ space by keeping k counters instead of just one counter.

Algorithm Frequent

Maintain a list of items being counted. Initially the list is empty. For each item, if it is the same as some item on the list, increment its counter by one. If it differs from all the items on the list, then if there are less than k items on the list, add the item to the list with its counter set to one. If there are already k items on the list decrement each of the current counters by one. Delete an element from the list if its count becomes zero.

Theorem 7.2 *At the end of Algorithm Frequent, for each $s \in \{1, 2, \ldots, m\}$, its counter on the list is at least the number of occurrences of s in the stream minus n/(k+1). In particular, if some s does not occur on the list, its counter is zero and the theorem asserts that it occurs fewer than n/(k+1) times in the stream.*

Proof: View each decrement counter step as eliminating some items. An item is eliminated if it is the current a_i being read and there are already k symbols different from it on the list in which case it and k other items are simultaneously eliminated. Thus, the elimination of each occurrence of an $s \in \{1, 2, \ldots, m\}$ is really the elimination of $k + 1$ items. Thus, no more than $n/(k + 1)$ occurrences of any symbol can be eliminated. Now, it is clear that if an item is not eliminated, then it must still be on the list at the end. This proves the theorem. ∎

Theorem **??** implies that we can compute the true relative frequency, the number of occurrences divided by n, of every $s \in \{1, 2, \ldots, m\}$ to within an additive term of $\frac{n}{k+1}$.

7.1.4 The Second Moment

This section focuses on computing the second moment of a stream with symbols from $\{1, 2, \ldots, m\}$. Let f_s denote the number of occurrences of symbol s in the stream. The second moment of the stream is given by $\sum_{s=1}^{m} f_s^2$. To calculate the second moment, for each symbol s, $1 \leq s \leq m$, independently set a random variable x_s to ± 1 with probability 1/2. Maintain a sum by adding x_s to the sum each time the symbol s occurs in the stream. At the end of the stream, the sum will equal $\sum_{s=1}^{m} x_s f_s$. The expected value of the sum will be zero where the expectation is over the choice of the ± 1 value for the x_s.

$$E\left(\sum_{s=1}^{m} x_s f_s\right) = 0.$$

Although the expected value of the sum is zero, its actual value is a random variable and the expected value of the square of the sum is given by

$$E\left(\sum_{s=1}^{m} x_s f_s\right)^2 = E\left(\sum_{s=1}^{m} x_s^2 f_s^2\right) + 2E\left(\sum_{s \neq t} x_s x_t f_s f_t\right) = \sum_{s=1}^{m} f_s^2,$$

The last equality follows since $E(x_s x_t) = E(x_s)E(x_t) = 0$ for $s \neq t$. Thus

$$a = \left(\sum_{s=1}^{m} x_s f_s \right)^2$$

is an estimator of $\sum_{s=1}^{m} f_s^2$. One difficulty, which we will come back to, is that to store all the x_i requires space m and we want to do the calculation in $\log m$ space.

How good this estimator is depends on its variance.

$$\text{Var}(a) \leq E \left(\sum_{s=1}^{m} x_s f_s \right)^4 = E \left(\sum_{1 \leq s,t,u,v \leq m} x_s x_t x_u x_v f_s f_t f_u f_v \right)$$

The first inequality is because the variance is at most the second moment and the second equality is by expansion. In the second sum, since the x_s are independent, if any one of s, u, t, or v is distinct from the others, then the expectation of the whole term is zero. Thus, we need to deal only with terms of the form $x_s^2 x_t^2$ for $t \neq s$ and terms of the form x_s^4. Note that this does not need the full power of mutual independence of all the x_s, it only needs 4-way independence, that any four of the $x_s's$ are mutually independent. I.e., for any distinct s, t, u, and v in $\{1, 2, \ldots, m\}$ and any a, b, c, and d in {-1, +1}

$$\text{Prob}(x_s = a, \;\; x_t = b, \;\; x_u = c, \;\; x_v = d) = \frac{1}{16}.$$

Each term in the above sum has four indices, s, t, u, v, and there are $\binom{4}{2}$ ways of choosing two indices that have the same x value. Thus,

$$\begin{aligned} \text{Var}(a) &\leq \binom{4}{2} E \left(\sum_{s=1}^{m} \sum_{t=s+1}^{m} x_s^2 x_t^2 f_s^2 f_t^2 \right) + E \left(\sum_{s=1}^{m} x_s^4 f_s^4 \right) \\ &= 6 \sum_{s=1}^{m} \sum_{t=s+1}^{m} f_s^2 f_t^2 + \sum_{s=1}^{m} f_s^4 \\ &\leq 3 \left(\sum_{s=1}^{m} f_s^2 \right)^2 . \end{aligned}$$

The only drawback with the algorithm we have described so far is that we need to store the vector $\mathbf{x}$ in memory so that we can do the running sums. This is too space-expensive. We need to do the problem in space dependent upon the logarithm of the size of the alphabet m, not m itself.

In the next section, we will see that the computation can be done in $O(\log m)$ space by using a pseudo-random vector $\mathbf{x}$ instead of a truly random one. This pseudo-randomness and limited independence has deep connections, so we will go into the connections as well.

Error-Correcting codes, polynomial interpolation and limited-way independence

Consider the problem of generating a random m-vector $\mathbf{x}$ of ± 1's so that any subset of four coordinates is mutually independent. An m-dimensional vector may be generated from a truly random "seed" of only $O(\log m)$ mutually independent bits. Thus, we need only store the $\log m$ bits and can generate any of the m coordinates when needed. Thus the 4-way independent random m-vector can be stored using only $\log m$ bits. The first fact needed for this is that for any k, there is a finite field F with exactly 2^k elements, each of which can be represented with k bits and arithmetic operations in the field can be carried out in $O(k^2)$ time. Here, k will be the ceiling of $\log_2 m$. We also assume another basic fact about polynomial interpolation; a polynomial of degree at most three is uniquely determined by its value over any field F at four points. More precisely, for any four distinct points $a_1, a_2, a_3, a_4 \in F$ and any four possibly not distinct values $b_1, b_2, b_3, b_4 \in F$, there is a unique polynomial $f(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3$ of degree at most three, so that with computations done over F, $f(a_1) = b_1, f(a_2) = b_2, f(a_3) = b_3$, and $f(a_4) = b_4$.

The definition of the pseudo-random ± 1 vector $\mathbf{x}$ with 4-way independence is simple. Choose four elements f_0, f_1, f_2, f_3 at random from F and form the polynomial $f(s) = f_0 + f_1 s + f_2 s^2 + f_3 s^3$. This polynomial represents $\mathbf{x}$ as follows. For $s = 1, 2, \ldots, m$, x_s is the leading bit of the k-bit representation of $f(s)$. Thus, the m-dimensional vector $\mathbf{x}$ requires only $O(k)$ bits where $k = \lceil \log m \rceil$.

Lemma 7.3 *The* $\mathbf{x}$ *defined above has 4-way independence.*

Proof: Assume that the elements of F are represented in binary using ± 1 instead of the traditional 0 and 1. Let s, t, u, and v be any four coordinates of $\mathbf{x}$ and let $\alpha, \beta, \gamma, \delta \in \{-1, 1\}$. There are exactly 2^{k-1} elements of F whose leading bit is α and similarly for β, γ, and δ. So, there are exactly $2^{4(k-1)}$ 4-tuples of elements b_1, b_2, b_3, $b_4 \in F$ so that the leading bit of b_1 is α, the leading bit of b_2 is β, the leading bit of b_3 is γ, and the leading bit of b_4 is δ. For each such b_1, b_2, b_3, and b_4, there is precisely one polynomial f so that $f(s) = b_1$, $f(t) = b_2$, $f(u) = b_3$, and $f(v) = b_4$. The probability that $x_s = \alpha$, $x_t = \beta$, $x_u = \gamma$, and $x_v = \delta$ is precisely

$$\frac{2^{4(k-1)}}{\text{total number of } f} = \frac{2^{4(k-1)}}{2^{4k}} = \frac{1}{16}$$

as asserted. ∎

The variance can be reduced by a factor of r by taking the average of r independent trials. With r independent trials the variance would be at most $\frac{4}{r}E^2(a)$, so to achieve relative error ε in the estimate of $\sum_{s=1}^{m} f_s^2$, $O(1/\varepsilon^2)$ independent trials suffice. If ε is $\Omega(1)$, then r is in $O(1)$, so it is not the number of trials r which is the problem. It is the m. /newline

Lemma **??** describes how to get one vector $\mathbf{x}$ with 4-way independence. However, we need $r = O(1/\varepsilon^2)$ vectors. Also the vectors must be mutually independent. But this is easy, just choose r independent polynomials at the outset.

To implement the algorithm with low space, store only the polynomials in memory. This requires $4k = O(\log m)$ bits per polynomial for a total of $O(\log m/\varepsilon^2)$ bits. When a symbol s in the stream is read, compute each polynomial at s to obtain the value for the corresponding value of the x_s and update the running sums. x_s is just the leading bit of the polynomial evaluated at s; this calculation is in $O(\log m)$ time. Thus, we repeatedly compute the x_s from the "seeds", namely the coefficients of the polynomials.

This idea of polynomial interpolation is also used in other contexts. Error-correcting codes is an important example. Say we wish to transmit n bits over a channel which may introduce noise. One can introduce redundancy into the transmission so that some channel errors can be corrected. A simple way to do this is to view the n bits to be transmitted as coefficients of a polynomial $f(x)$ of degree $n-1$. Now transmit f evaluated at points $1, 2, 3, \ldots, n+m$. At the receiving end, any n correct values will suffice to reconstruct the polynomial and the true message. So up to m errors can be tolerated. But even if the number of errors is at most m, it is not a simple matter to know which values are corrupted. We do not elaborate on this here.

7.2 Matrix Algorithms using sampling

How does one deal with a large matrix? An obvious suggestion is to take a sample of the matrix. Uniform sampling does not work in general. For example, if only a small fraction of the matrix entries are large, uniform sampling may miss them. So the sampling probabilities need to take into account the magnitude of the entries. It turns out that sampling the rows/columns of a matrix with probabilities proportional to the length squared of the row/column is a good idea in many contexts. We present two examples here; matrix multiplication and the sketch of a matrix.

7.2.1 Matrix Multiplication Using Sampling

Suppose A is an $m \times n$ matrix and B is an $n \times p$ matrix and the product AB is desired. We show how to use sampling to get an approximate product faster than the traditional

multiplication. Let $A(:,k)$ denote the k^{th} column of A. $A(:,k)$ is a $m \times 1$ matrix. Let $B(k,:)$ be the k^{th} row of B. $B(k,:)$ is a $1 \times n$ matrix. It is easy to see that

$$AB = \sum_{k=1}^{n} A(:,k)B(k,:).$$

Note that for each value of k, $A(:,k)B(k,:)$ is an $m \times p$ matrix each element of which is a single product of elements of A and B. An obvious use of sampling suggests itself. Sample some values for k and compute $A(:,k)\,B(k,:)$ for the sampled k's and use their suitably scaled sum as the estimate of AB. It turns out that nonuniform sampling probabilities are useful. Define a random variable z that takes on values in $\{1,2,\ldots,n\}$. Let p_k denote the probability that z assumes the value k. The p_k are nonnegative and sum to one. Define an associated random matrix variable that has value

$$X = \frac{1}{p_k} A(:,k)\,B(k,:) \tag{7.1}$$

with probability p_k. Let $E(X)$ denote the entry-wise expectation.

$$E(X) = \sum_{k=1}^{n} \text{Prob}(z=k)\frac{1}{p_k} A(:,k)\,B(k,:) = \sum_{k=1}^{n} A(:,k)B(k,:) = AB.$$

This explains the scaling by $\frac{1}{p_k}$ in X.

Define the variance of X as the sum of the variances of all its entries.

$$\text{Var}(X) = \sum_{i=1}^{m}\sum_{j=1}^{p} \text{Var}(x_{ij}) \le \sum_{ij} E\left(x_{ij}^2\right) \le \sum_{ij}\sum_{k} p_k \frac{1}{p_k^2} a_{ik}^2 b_{kj}^2.$$

We can simplify the last term by exchanging the order of summations to get

$$\text{Var}(X) \le \sum_{k} \frac{1}{p_k} \sum_{i} a_{ik}^2 \sum_{j} b_{kj}^2 = \sum_{k} \frac{1}{p_k} |A(:,k)|^2 |B(k,:)|^2.$$

What is the best choice of p_k? It is the one which minimizes the variance. In the above calculation, we discarded $\sum_{ij} E^2(x_{ij})$, but this term is just $\sum_{ij}(AB)_{ij}^2$ since $E(X) = AB$ and is independent of p_k. So we should choose p_k to minimize $\sum_k \frac{1}{p_k}|A(:,k)|^2|B(k,:)|^2$. It can be seen by calculus that the minimizing p_k are proportional to $|A(:,k)||B(k,:)|$. In the important special case when $B = A^T$, pick columns of A with probabilities proportional to the squared length of the columns.

Such sampling has been widely used and it goes under the name of "length squared sampling". We will use it here. Even in the general case when B is not A^T, it simplifies the bounds. If p_k is proportional to $|A(:,k)|^2$, i.e, $p_k = \frac{|A(:,k)|^2}{||A||_F^2}$, then

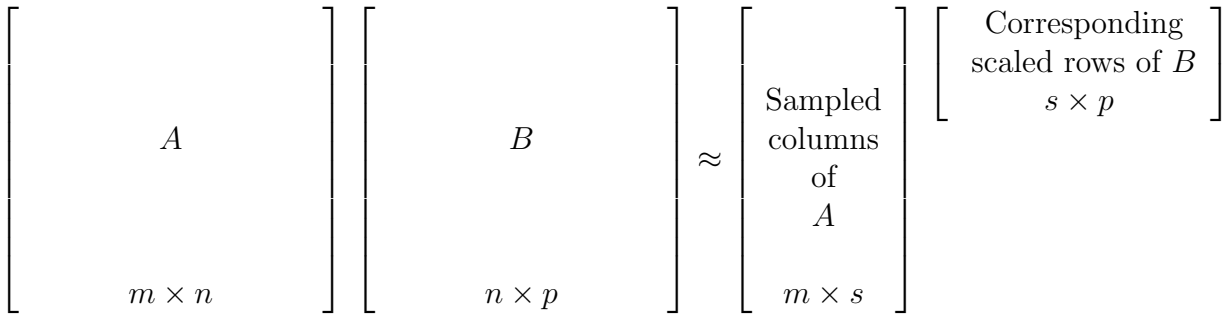

Figure 7.2: Approximate Matrix Multiplication using sampling

$$\mathrm{Var}(X) \leq ||A||_F^2 \sum_k |B(k,:)|^2 = ||A||_F^2 ||B||_F^2.$$

To reduce the variance, do s independent trials. Each trial i, $i = 1, 2, \ldots, s$ yields a matrix X_i as in (**??**). We take $\frac{1}{s}\sum_{i=1}^{s} X_i$ as our estimate of AB. Since the variance of a sum of independent random variables is the sum of variances, the variance of $\frac{1}{s}\sum_{i=1}^{s} X_i$ is $\frac{1}{s}\mathrm{Var}(X)$ and so is at most $\frac{1}{s}||A||_F^2||B||_F^2$.

To implement this, suppose $k_1, k_2, \ldots, k_s$ are the k 's chosen in each trial. It is easy to see that

$$\frac{1}{s}\sum_{i=1}^{s} X_i = \frac{1}{s}\left(\frac{A(:,k_1)\,B(k_1,:)}{p_{k_1}} + \frac{A(:,k_2)\,B(k_2,:)}{p_{k_2}} + \cdots + \frac{A(:,k_s)\,B(k_s,:)}{p_{k_s}}\right) = C\tilde{B},$$

where, C is the $m \times s$ matrix of the chosen columns of A and $\tilde{B}$ is an $s \times p$ matrix with the corresponding rows of B scaled, namely, $\tilde{B}$ has rows $B(k_1,:)/(sp_{k_1})$, $B(k_2,:)/(sp_{k_2})$, $\ldots$ $B(k_s,:)/(sp_{k_s})$. This is represented in Figure **??**.
We summarize our discussion in Lemma **??**.

Lemma 7.4 *Suppose A is an $m \times n$ matrix and B is an $n \times p$ matrix. The product AB can be estimated by $C\tilde{B}$, where, C is an $m \times s$ matrix consisting of s columns of A picked according to length-squared distribution and $\tilde{B}$ is the $s \times p$ matrix consisting of the corresponding rows of B scaled as above. The error is bounded by:*

$$E\left(||AB - C\tilde{B}||_F^2\right) \leq \frac{||A||_F^2\,||B||_F^2}{s}.$$

7.2.2 Sketch of a Large Matrix

The main result of this section will be that for any matrix, a sample of columns and rows, each picked according to length squared distribution is a sufficient sketch of the matrix. Let A be an $m \times n$ matrix. Pick s columns of A according to length squared distribution. Let C be the $m \times s$ matrix containing the picked columns. Similarly, pick r

rows of A according to length squared distribution on the rows of A. Let R be the $r \times n$ matrix of the picked rows. From C and R, we can find a matrix U so that $A \approx CUR$. The schematic diagram is given in Figure **??**.

The proof makes crucial use of the fact that the sampling of rows and columns is with probability proportional to the squared length. One may recall that the top k singular vectors of the SVD of A, give a similar picture; but the SVD takes more time to compute, requires all of A to be stored in RAM, and does not have the property that the rows and columns are directly from A. The last property - that the approximation involves actual rows/columns of the matrix rather than linear combinations - is called an *interpolative approximation* and is useful in many contexts. However, the SVD does yield the best 2-norm approximation. Error bounds for the approximation CUR are weaker.

We briefly touch upon two motivations for such a sketch. Suppose A is the document-term matrix of a large collection of documents. We are to "read" the collection at the outset and store a sketch so that later, when a query represented by a vector with one entry per term arrives, we can find its similarity to each document in the collection. Similarity is defined by the dot product. In Figure **??** it is clear that the matrix-vector product of a query with the right hand side can be done in time $O(ns + sr + rm)$ which would be linear in n and m if s and r are $O(1)$. To bound errors for this process, we need to show that the difference between A and the sketch of A has small 2-norm. Recall that the 2-norm $||A||_2$ of a matrix A is $\max_{|\mathbf{x}|=1} |A\mathbf{x}|$. The fact that the sketch is an interpolative approximation means that our approximation essentially consists a subset of documents and a subset of terms, which may be thought of as a representative set of documents and terms.

A second motivation comes from recommendation systems. Here A would be a customer-product matrix whose $(i, j)^{th}$ entry is the preference of customer i for product j. The objective is to collect a few sample entries of A and based on them, get an approximation to A so that we can make future recommendations. A few sampled rows of A (all preferences of a few customers) and a few sampled columns (all customers' preferences for a few products) give a good approximation to A provided that the samples are drawn according to the length-squared distribution.

It remains now to describe how to find U from C and R. We describe this assuming RR^T is invertible. Through the rest of this section, we make the assumption that RR^T is invertible. This case will convey the essential ideas. Also, note that since r in general will be much smaller than n and m, unless the matrix A is degenerate, it is likely that the r rows in the sample R will be linearly independent giving us invertibility of RR^T.

We begin with some intuition. Write A as AI, where, I is the $n \times n$ identity matrix. Pretend for the moment that we approximate the product AI by sampling s columns of A according to length-squared. Then, as in the last section, write $AI \approx CW$, where, W consists of a scaled version of the s rows of I corresponding to the s columns of A that

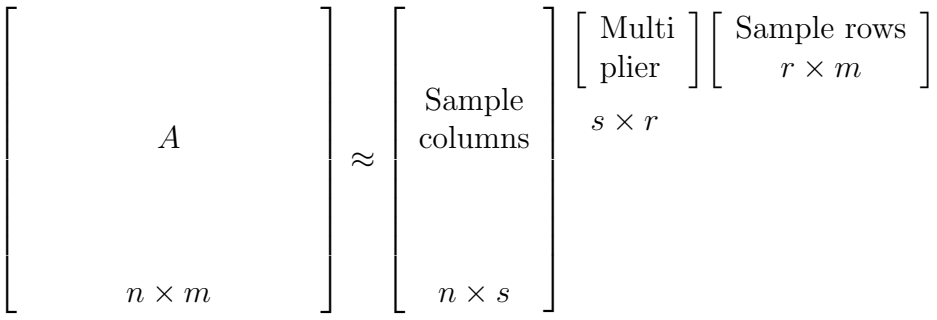

Figure 7.3: Schematic diagram of the approximation of A by a sample of s columns and r rows.

were picked. Lemma **??** bounds the error $||A - CW||_F^2$ by $||A||_F^2||I||_F^2/s = ||A||_F^2 \frac{n}{s}$. But clearly, we would like the error to be a fraction of $||A||_F^2$ which would require $s \geq n$, which clearly is of no use since this would pick as many or more columns than the whole of A.

We modify the intuition. We assumed that RR^T is invertible. Then it is easy to see, Lemma **??**, that $R^T(RR^T)^{-1}R$ (we denote $R^T(RR^T)^{-1}R$ by P for convenience) acts as the identity matrix on the space V spanned by the rows of R. Lets use this identity-like matrix P instead of I in the above discussion. Using the fact that R is picked according to length squared we will show the following proposition later.

Proposition 7.5 *$A \approx AP$ and the error $E\left(||A - AP||_2\right)$ is at most $||A||_F^2/r$.*

We then use Lemma **??** to argue that instead of doing the multiplication AP, we can use the sampled columns of A and the corresponding rows of P. The sampled s columns of A form C. We have to take the corresponding s rows of $P = R^T(RR^T)^{-1}R$, which is the same as taking the corresponding s rows of R^T, and multiplying this by $(RR^T)^{-1}R$. It is easy to check that this leads to an expression of the form CUR. Further, by Lemma **??**, the error is bounded by

$$E\left(||AP - CUR||_2^2\right) \leq E\left(||AP - CUR||_F^2\right) \leq \frac{||A||_F^2||P||_F^2}{s} \leq \frac{r}{s}||A||_F^2, \qquad (7.2)$$

since we will show later that:

Proposition 7.6 *$||P||_F^2 \leq r$.*

Putting (**??**) and Proposition **??** together (and using $||A-CUR||_2 \leq ||A-AP||_2+||AP-CUR||_2$ which implies that $||A - CUR||_2^2 \leq 2||A - AP||_2^2 + 2||AP - CUR||_2^2$), we get the main result:

Theorem 7.7 *Suppose A is any $m \times n$ matrix and r and s are positive integers. Suppose C is a $m \times s$ matrix of s columns of A picked according to length squared sampling and*

similarly R is a matrix of r rows of A picked according to length squared sampling. Then, we can find from C, R an $s \times r$ matrix U so that

$$E\left(||A - CUR||_2^2\right) \leq ||A||_F^2 \left(\frac{2}{r} + \frac{2r}{s}\right).$$

Choosing $s = r^2$, the bound becomes $O(1/r)||A||_F^2$ and if want the bound to be at most $\varepsilon||A||_F^2$ for some small $\varepsilon > 0$, it suffices to choose $r \in \Omega(1/\varepsilon)$.

We now briefly look at the time needed to compute U. The only involved step in computing U is to find $(RR^T)^{-1}$. But note that RR^T is an $s \times s$ matrix and since s is to much smaller than n, m, this is fast. Now we prove all the claims used in the discussion above.

Lemma 7.8 *If RR^T is invertible, then $R^T(RR^T)^{-1}R$ acts as the identity matrix on the row space of R. I.e., for every vector $\mathbf{x}$ of the form $\mathbf{x} = R^T\mathbf{y}$ (this defines the row space of R), we have $R^T(RR^T)^{-1}R\mathbf{x} = \mathbf{x}$.*

Proof: For $\mathbf{x} = R^T\mathbf{y}$, since RR^T is invertible

$$R^T(RR^T)^{-1}R\mathbf{x} = R^T(RR^T)^{-1}RR^T\mathbf{y} = R^T\mathbf{y} = \mathbf{x}$$

■

Now we prove Proposition (**??**). First suppose $\mathbf{x} \in V$. Then we can write $\mathbf{x} = R^T\mathbf{y}$ and so $P\mathbf{x} = R^T(RR^T)^{-1}RR^T\mathbf{y} = R^T\mathbf{y} = \mathbf{x}$, so for $\mathbf{x} \in V$, we have $(A - AP)\mathbf{x} = \mathbf{0}$. So, it suffices to consider $\mathbf{x} \in V^\perp$. For such $\mathbf{x}$, we have $(A - AP)\mathbf{x} = A\mathbf{x}$ and we have

$$|(A - AP)\mathbf{x}|^2 = |A\mathbf{x}|^2 = \mathbf{x}^T A^T A\mathbf{x} = \mathbf{x}^T(A^T A - R^T R)\mathbf{x} \leq ||A^T A - R^T R||_2 |\mathbf{x}|^2,$$

so we get $||A - AP||_2^2 \leq ||A^T A - R^T R||_2$, so it suffices to prove that $||A^T A - R^T R||_2 \leq ||A||_F^2/r$ which follows directly from Lemma (**??**) since we can think of $R^T R$ as a way of estimating $A^T A$ by picking (according to length-squared distribution) columns of A^T, i.e., rows of A. This proves Proposition **??**.

Proposition **??** is easy to see: Since by Lemma **??**, P is the identity on the space V spanned by the rows of R, we have that $||P||_F^2$ is the sum of its singular values squared which is at most r as claimed.

7.3 Sketches of Documents

Suppose one wished to store all the web pages from the WWW. Since there are billions of web pages, one might store just a sketch of each page where a sketch is a few hundred bits that capture sufficient information to do whatever task one had in mind. A web page or a document is a sequence. We begin this section by showing how to sample a set and then how to convert the problem of sampling a sequence into a problem of sampling a set.

Consider subsets of size 1000 of the integers from 1 to 10^6. Suppose one wished to compute the resemblance of two subsets A and B by the formula

$$\text{resemblance}\,(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Suppose that instead of using the sets A and B, one sampled the sets and compared random subsets of size ten. How accurate would the estimate be? One way to sample would be to select ten elements uniformly at random from A and B. However, this method is unlikely to produce overlapping samples. Another way would be to select the ten smallest elements from each of A and B. If the sets A and B overlapped significantly one might expect the sets of ten smallest elements from each of A and B to also overlap. One difficulty that might arise is that the small integers might be used for some special purpose and appear in essentially all sets and thus distort the results. To overcome this potential problem, rename all elements using a random permutation.

Suppose two subsets of size 1000 overlapped by 900 elements. What would the overlap be of the 10 smallest elements from each subset? One would expect the nine smallest elements from the 900 common elements to be in each of the two subsets for an overlap of 90%. The resemblance(A, B) for the size ten sample would be 9/11=0.81.

Another method would be to select the elements equal to zero mod m for some integer m. If one samples mod m the size of the sample becomes a function of n. Sampling mod m allows us to also handle containment.

In another version of the problem one has a sequence rather than a set. Here one converts the sequence into a set by replacing the sequence by the set of all short subsequences of some length k. Corresponding to each sequence is a set of length k subsequences. If k is sufficiently large, then two sequences are highly unlikely to give rise to the same set of subsequences. Thus, we have converted the problem of sampling a sequence to that of sampling a set. Instead of storing all the subsequences, we need only store a small subset of the set of length k subsequences.

Suppose you wish to be able to determine if two web pages are minor modifications of one another or to determine if one is a fragment of the other. Extract the sequence of words occurring on the page. Then define the set of subsequences of k consecutive words from the sequence. Let $S(D)$ be the set of all subsequences of length k occurring in document D. Define resemblance of A and B by

$$\text{resemblance}\,(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

And define containment as

$$\text{containment}\,(A,B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$$

Let W be a set of subsequences. Define $\min(W)$ to be the s smallest elements in W and define $\text{mod}\,(W)$ as the set of elements of w that are zero mod m.

Let π be a random permutation of all length k subsequences. Define $F(A)$ to be the s smallest elements of A and $V(A)$ to be the set mod m in the ordering defined by the permutation.

Then

$$\frac{F(A) \cap F(B)}{F(A) \cup F(B)}$$

and

$$\frac{|V(A) \cap V(B)|}{|V(A) \cup V(B)|}$$

are unbiased estimates of the resemblance of A and B. The value

$$\frac{|V(A) \cap V(B)|}{|V(A)|}$$

is an unbiased estimate of the containment of A in B.