# Lecture 11

## Streaming Algorithms (contd) + Randomly Sprinkled Topics

# Recap

- Streaming algorithms
  - Stream: $A = a_1, \dots, a_m$, each $a_i \in [n]$
  - Want to compute some property / statistic about the stream using space sublinear in $m$ and $n$

  - Missing elements problem
  - Computing frequency moments ($F_k$)
  - Finding the majority element

# Today

- Continue streaming algorithms
  - Generalize the majority elements to $k$-heavy hitters
  - Solving heavy hitters using "count-min" sketch

- Online expert learning and its applications

# RECAP: Majority Element

- Input: Stream $A = a_1, \dots, a_m$, where $a_i \in [n]$
- Q: Is there a value $i$ that appears more than $m/2$ times?

- Algorithm:
  - Store candidate $a^*$, and a counter $c$ (initially $c = 0$).
  - For $i = 1 \dots m$
    - If $c = 0$: Set $a^* = a_i$, and $c = 1$.
    - Else:
      - If $a^* = a_i$, $c \leftarrow c + 1$
      - If $a^* \neq a_i$, $c \leftarrow c - 1$

# RECAP: Majority Element

- Space: Clearly $O(\log m + \log n)$ bits
- Claim: If there exists a value $v$ that appears more than $m/2$ times, then $a^* = v$ at the end.
- Proof:
  - Take an occurrence of $v$ (say $a_i$), and let's pair it up:
    - If it decreases the counter, pair up with the unique element $a_j$ ($j < i$) that contributed the 1 we just decreased.
    - If it increases the counter:
      - If the added 1 is never taken back, QED!
      - If it is decreased by $a_j$ ($j > i$), pair up with that.
  - Because at least occurrence of $v$ is not paired, the "never taken back" case happens at least once.

# RECAP: Majority Element

- Space: Clearly $O(\log m + \log n)$ bits
- Claim: If there exists a value $v$ that appears more than $m/2$ times, then $a^* = v$ at the end.
- A simpler proof:
  - At any step, let $c' = c$ if $a^* = v$, and $c' = -c$ otherwise.
  - Every occurrence of $v$ must increase $c'$ by 1.
  - Every occurrence of a value other than $v$ either increases or decreases $c'$ by 1.
  - Majority $\Rightarrow$ more increments than decrements in $c'$.
  - Thus, a positive value at the end!

# RECAP: Majority Element

- Note 1: When a majority element does not exist, the algorithm doesn't necessarily find the mode.

- Note 2: If a majority element exists, it correctly finds that element. However, if there is no majority element, the algorithm does not detect that and still returns a value.
  - It can be trivially checked if the returned value is indeed a majority element if a second pass over the stream is allowed.
  - Surprisingly, we can prove that this cannot be done in 1-pass. (3 slides later!)

# $k$-Heavy Hitters

- Generalization:
  - Given $k$, which elements (if any) appear more than $m/k$ times?
  - Misra and Gries generalized the majority algorithm into a deterministic algorithm that
    - Returns a set $A$ of at most $k-1$ pairs $(v, \tilde{f}_v)$.
    - For every $(v, \tilde{f}_v) \in A$ where the true frequency of $v$ is $f_v$,
    $$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$
    - Corollary: Every $k$-heavy hitter is definitely covered in $A$. Although, some other elements might be present too.
      - A second pass can be used to eliminate false positives.
    - Space: $O\big(k(\log n + \log m)\big)$

# $k$-Heavy Hitters

- Misra & Gries Algorithm:
- $A \leftarrow \emptyset$; ($A$ contains up to $k-1$ pairs $\left(v, \tilde{f}_v\right)$)
- For each $i$:
  - If $a_i$ is in $A$: $\tilde{f}_{a_i} \leftarrow \tilde{f}_{a_i} + 1$
  - Else:
    - If $|A| < k - 1$: Add $(a_i, 1)$ to $A$
    - Else:
      - For each $\left(v, \tilde{f}_v\right) \in A$:
        - $\tilde{f}_v \leftarrow \tilde{f}_v - 1$
        - If $\tilde{f}_v = 0$, remove $(v, \tilde{f}_v)$ from $A$
- Output $A$

# $k$-Heavy Hitters

- The only non-trivial change is that when our storage is full, and we encounter a new element, we decrease the counter of *every* stored element.
- Claim: For every $(v, \tilde{f}_v) \in A,\ f_v - \frac{m}{k} \le \tilde{f}_v \le f_v$
- Proof:
  - Similar to majority proof. Call an occurrence of $v$ "wasted" if it either decreases counts of $k-1$ values stored, or it increases count of $v$ which is decreased later.
  - Every wasted occurrence of $v$ causes $k-1$ other unique wasted occurrences. (WHY?)
  - At most $m/k$ wasted occurrences.

# $k$-Heavy Hitters

- Claim: "Find an element that appears more than $m/k$ times, or say that none does" cannot be solved in sublinear space in a single pass.
- Proof:
  - I'll prove for $k > m/2$ (i.e., "appear at least twice"). I leave it to you to extend this to other values of $k$.
  - Let $a_1, \dots, a_{n/2}$ be a sequence that contains distinct members of $\{1, \dots, n\}$.
  - On the next value, the algo acts as a membership test.
  - Thus, it must be able to distinguish between all possible $\binom{n}{n/2}$ subsets.

# $\phi$-Heavy Hitters

- Problem: Given a stream of length $m$, find all values that appear at least $\phi m$ times.

- $\epsilon$-approximate version: Return a set that
  - Contains every value which appears at least $\phi m$ times,
  - And does not contain any value that appears less than $(\phi - \epsilon)m$ times.

# $\phi$-Heavy Hitters

- In the Misra-Gries algorithm…
  - Suppose we can set $k = 1/\epsilon$, and guarantee that for every $(v, \tilde{f}_v)$ included in the final set $A$,
  $$f_v - \epsilon m \le \tilde{f}_v \le f_v$$
  - Then, return all $v \in A$ such that $\tilde{f}_v \ge (\phi - \epsilon)m$.
  - This guarantees that every $v$ with $f_v \ge \phi m$ is included, and every $v$ included satisfies $f_v \ge (\phi - \epsilon)m$.
  - This uses space $O\left(\left(\frac{1}{\epsilon}\right)(\log m + \log n)\right)$ and does not use randomization.

# Sketching for Heavy Hitters

- A generic method that provides an alternative approach to heavy hitters (with some pros/cons over Misra-Gries algorithm) and applies to many other streaming problems.
- A sketch $sk$ is a function for which there exists a space-efficient combining algorithm $COMB$:

$$COMB\big(sk(A_1), sk(A_2)\big) = sk(A_1 A_2)$$

- Frequency counting through sketching

# Simple Hash Count Sketch

- Set $k = 2/\epsilon$.
- $C \leftarrow$ length $k$ integer array, initially 0.
- Choose $h: [n] \rightarrow [k]$ from a "2-universal family of hash functions".
- For each $i = 1, \dots, m$:
  - $C[h(a_i)] \leftarrow C[h(a_i)] + 1$
- Output: $\tilde{f} \leftarrow (C, h)$
  - $\tilde{f}_v = C[h(v)]$

# Simple Hash Count Sketch

- This also uses $O\left(\left(\frac{1}{\epsilon}\right)(\log n + \log m)\right)$ space.
- Let us analyze the relationship between $f_v$ and $\tilde{f}_v$ for any value $v$.
- Clearly, $C[h(v)]$ is incremented for every occurrence of $v$, and is never decremented.
  - So $\tilde{f}_v \geq f_v$.
- But it is also incremented every time $v'$ appears where $h(v) = h(v')$.

# Simple Hash Count Sketch

- Choosing a 2-universal hash function ensures that the buckets assigned to every pair of values are perfectly random.
  - This implies $\Pr[h(v) = h(v')] = 1/k$.

- Thus, $\tilde{f}_v$ is incorrectly incremented by $f_{v'}$ for every $v' \neq v$ with probability $1/k$.
  - Thus, $E\left[\tilde{f}_v\right] \leq f_v + m/k$.
  - Using Markov's inequality, $\Pr\left[\tilde{f}_v \geq f_v + \epsilon m\right] \leq 1/2$.

# Count-Min Sketch

- Count-Min sketch simply diminishes the error probability by keeping $\log\left(\frac{1}{\delta}\right)$ different copies of $C$, each with a random hash function.
- Because $\tilde{f}_v \geq f_v$ in each of them, the best estimate is obtained by taking the minimum of $C[h(v)]$ over all counters $C$.
- The probability that this is an over-estimate by $\epsilon m$ is now at most $\delta$.

# Misra-Gries vs Count-Min

- Two reasons why Misra-Gries is better:
  - Misra-Gries stores $O\left(\frac{1}{\epsilon}\right)$ numbers, while Count-Min stores $O\left(\frac{\log\left(\frac{1}{\delta}\right)}{\epsilon}\right)$ numbers.
  - Misra-Gries runs deterministically while Count-Min uses randomization.
- One reason why they're incomparable:
  - Misra-Gries provides a lower bound on frequency, while Count-Min provides an upper bound.

# Misra-Gries vs Count-Min

- Reasons to use Count-Min:
  - Count-Min is extremely fast as we just compute a hash, and update one value in each of a small number of counters $C$.
    - Misra-Gries may need to go over $1/\epsilon$ values and decrease them.
  - Using counters for sketching is a general-purpose idea that is useful for doing many things.
  - For instance, in Count-Min, you can easily allow "deletions" in addition to "insertions".

# Random Remarks

- Count-Min has applications when working with large databases.
  - You can process dataset with entries that go up to a billion, keep a small number of hash functions that map every entry to a small value (in thousands), and return a pretty accurate count.
- For solving such problems, there are two other popular approaches.
  - One is to compute “approximate quantiles”. An example is the approximate median question on A3.
  - Another is to use random projections, when the input stream is viewed as a vector in a high dimensional space.

# A semi-streaming model

- Introduced by Feigenbaum et al. in 2005 for graph problems in a streaming model
  - Graph $G = (V, E)$ with $|V| = n$, $|E| = m$
  - Vertices or edges arrive in a stream (two very different models!)
  - We want to compute a graph solution (e.g., matching)
    - Must need $\Omega(n)$ space.
    - Goal: use $\tilde{O}(n)$ space (hides polylog factors), not $O(m)$ space.

- This is studied for single as well as multi-pass algorithms.

# Streaming vs Online

- At first glance, it might seem that streaming is less restrictive than online setting.
  - Because you don't have to make irrevocable decisions.

- But is it obvious that every online algorithm can be simulated as a streaming algorithm?
  - An online algorithm does not have to abide by $\tilde{O}(n)$ space requirement.
  - It might remember all previously seen edges to make a new decision.
  - It's not clear if an online algorithm can really exploit this additional space allowance.

# Revisiting Bipartite Matching

- Edge-arrival model:
  - AFAIK, there is no semi-streaming algorithm (even randomized) with worst-case bound better than ½ that is achieved by greedy
  - A slightly better approximation if edges arrive in a random order.

- Vertex-arrival model:
  - Ranking (KVV) can be simulated as a randomized semi-streaming algorithm.
  - Surprisingly, Goel et al. [2011] show that there is a *deterministic* semi-streaming algorithm with the same $1 - {}^{1}/_{e}$ worst-case bound.
    - Contrast this with the fact that we can't beat ½ in the online model.
    - That is, if we *make* matching decisions as vertices arrive, we can't beat ½.
    - But we can store $\tilde{O}(n)$ bit information as we process the stream, and output a $1 - {}^{1}/_{e}$ approximate matching at the very end.

# Streaming vs Online

- For online algorithms, we noticed a significant difference between worst-case arrivals and ROM.
  - This is because we have to make irrevocable decisions as the vertices arrive.

- For streaming algorithms, we can still define ROM
  - But there is less advantage because we still get to "see" the entire input before returning an output.

# Online Expert Learning

- Setup:
  - On each day, we want to decide whether to invest in the market.
  - We have, at our disposal, $n$ experts that give their prediction of 1 (invest) or 0 (don't) every day.
  - Some experts may be better than some other experts, but we don't know.
  - We would like to take their advice, and decide to invest or not.
  - Our goal is to do almost as good as the best expert in hindsight!

# Online Expert Learning

- Formally, there are $n$ experts and $T$ time steps.
- At each time period $t$:
  - Every expert $i$ gives his prediction.
  - You look at all the predictions, and make a decision.
  - Then you find out what the right decision for step $t$ was.

- Simplest idea:
  - Keep a weight for each expert.
  - Decrease the weight every time the expert makes a mistake.
  - Use a weighted majority for prediction.

# Online Expert Learning

- Weighted Majority:
  - Fix $\eta \leq 1/2$.
  - Start with $w_i^{(1)} = 1$.
  - In time step $t$, predict 1 if the total weight of experts predicting 1 is larger than the total weight of experts predicting 0, and vice-versa.
  - At the end of time step $t$, set $w_i^{(t+1)} \leftarrow w_i^{(t)} \cdot (1 - \eta)$ for every expert that made a mistake.

# Online Expert Learning

- Theorem: Let $m_i^{(t)}$ and $M^{(t)}$ be the number of mistakes made by expert $i$ and the algorithm in the first $t$ rounds. Then for every $i$ and $T$:

$$M^{(T)} \leq 2(1+\eta)\, m_i^{(T)} + \frac{2\ln n}{\eta}$$

- Proof:
  - Consider $\Phi^{(\mathrm{t})} = \sum_i w_i^{(t)}$.
  - If the algorithm makes a mistake in round $t$, at least half the total weight decreases by a factor of $1-\eta$. Hence:
  - $\Phi^{t+1} \leq \Phi^{t}\left(\frac{1}{2}+\frac{1}{2}(1-\eta)\right) = \Phi^{t}\left(1-\frac{\eta}{2}\right)$.

# Online Expert Learning

- Theorem: Let $m_i^{(t)}$ and $M^{(t)}$ be the number of mistakes made by expert $i$ and the algorithm in the first $t$ rounds. Then for every $i$ and $T$:

$$M^{(T)} \leq 2(1+\eta)\, m_i^{(T)} + \frac{2\ln n}{\eta}$$

- Proof:
  - Thus: $\Phi^{(T+1)} \leq n\left(1 - \frac{\eta}{2}\right)^{M^{(T)}}$.
  - However, the best expert $i$ has $w_i^{(T+1)} = (1-\eta)^{m_i^{(t)}}$
  - Use $\Phi^{(T+1)} \geq w_i^{(T+1)}$ and $-\ln(1-\eta) \leq \eta + \eta^2$ (because $\eta \leq 1/2$).

# Online Expert Learning

- The beauty of this is that it makes no statistical assumptions about how the experts make mistakes.
- You can have adversarial mistakes, and still the algorithm is guaranteed (i.e., no randomization) to make only about twice as many mistakes as the best expert *in hindsight.*
- It can be shown that this bound is tight for any deterministic algorithm.

# Randomized Weighted Majority

- Using randomization, we can eliminate the factor of 2, and do almost as good as the best expert.
- Simple Change:
  - Let $W_1^{(t)}$ be the total weight of experts predicting 1, and $W_0^{(t)}$ be the total weight of experts predicting 0.
  - The deterministic version predicts 1 if $W_1^{(t)} > W_0^{(t)}$, and vice-versa.
  - The randomized version will predict 1 with probability $\frac{W_1^{(t)}}{W_1^{(t)}+W_0^{(t)}}$, and predict 0 with the remaining probability.

# Randomized Weighted Majority

- This is equivalent to:
  - "Pick an expert with probability proportional to his weight, and go with his prediction."
  - Probability of picking expert $i$ in step $t$ is $p_i^{(t)} = \frac{w_i^{(t)}}{\Phi^{(t)}}$.
- Let $b_i^{(t)} = 1$ if expert $i$ makes a mistake at step $t$, and $0$ otherwise.
  - The algorithm makes a mistake with probability $\sum_i p_i^{(t)} b_i^{(t)} = \boldsymbol{p}^{(t)} \cdot \boldsymbol{b}^{(t)}$ (vector notation)
  - E[#mistakes after $T$ rounds] = $\sum_{t=1}^{T} \boldsymbol{p}^{(t)} \cdot \boldsymbol{b}^{(t)}$

# Randomized Weighted Majority

- Let's now consider the function $\Phi^{(t)}$.

$$\Phi^{(t+1)} = \sum_i w_i^{(t+1)} = \sum_i w_i^{(t)} \cdot \left(1 - \eta b_i^{(t)}\right)$$
$$= \Phi^{(t)} - \eta \Phi^{(t)} \sum_i p_i^{(t)} \cdot b_i^{(t)} = \Phi^{(t)}\left(1 - \eta\, \boldsymbol{p}^{(t)} \cdot \boldsymbol{b}^{(t)}\right)$$
$$\leq \Phi^{(t)} \exp\left(-\eta\, \boldsymbol{p}^{(t)} \cdot \boldsymbol{b}^{(t)}\right)$$

- Apply this iteratively, and you get
$$\Phi^{(T+1)} \leq n \cdot \exp(-\eta \cdot E[\#\text{mistakes}])$$
- Also use that the weight of the best expert is at least $(1-\eta)^{m_i^{(T)}}$.

# Randomized Weighted Majority

- Theorem: If $M^{(T)}$ is the expected number of mistakes made by randomized weighted majority in the first $T$ rounds, then for every $i$ and $T$:

$$M^{(T)} \leq (1+\eta) m_i^{(T)} + \frac{2 \ln n}{\eta}$$

- Note that setting $\eta = \sqrt{\frac{\ln n}{T}}$ gives (best expert's mistake)+$O\left(\sqrt{T \cdot \ln n}\right)$
- Average regret = per-round additional mistakes = $O\left(\sqrt{\frac{\ln n}{T}}\right)$ = sublinear (goes to 0 as $T \to \infty$)

# Applications

- Generalizations:
  - "Multiplicative Weights Algorithm", where the cost of selecting expert $i$ in step $t$ is $m_i^{(t)} \in [-1,1]$ (real-valued).
  - "Sleeping experts" variant where we want to do as well as the best expert in the last $T'$ rounds.

- Fundamental tool that can be used as black-box within other algorithms.
- Let's see some interesting applications of RWM.

# Learning Disjunctions

- Setup:
  - Binary variables $x = (x_1, \dots, x_n)$, and an unknown disjunction $f$ over a subset of variables, e.g., $f(x) = x_3 \vee x_5 \vee x_9$
- Goal:
  - Given a sequence of variable values, predict the outcome of $f$.
  - Make the fewest mistakes over time.
- Simple idea:
  - Start with $h(x) = x_1 \vee \cdots \vee x_n$
  - You never predict $0$ when the true answer is $1$.
  - If you predict $1$ when the true answer was $0$, take all $x_i$ which were $1$ on that example, and throw them out.
  - At most $n$ mistakes, which is optimal to distinguish between $2^n$ functions (halving argument).

# Learning Disjunctions: $r$-way

- Suppose we know that the target function $f$ is an $r$-way disjunction (disjunction of $r$ variables).
  - Can we do better?

- In principle, there are $O(n^r)$ functions, so by halving argument, a lower bound is $\Omega(r \log n)$.
  - Can we achieve this, efficiently?

- Yes! Using Winnow algorithm.

# Winnow Algorithm

- Algorithm:
  - Maintain $h(x)$ which predicts 1 iff $\sum_i w_i x_i \geq n$
  - Initialize $w_i = 1$ for all $i$.
  - Mistake on true answer = 1:
    - Make $w_i \leftarrow 2w_i$ for every $x_i = 1$
  - Mistake on true answer = 0:
    - Make $w_i \leftarrow 0$ for every $x_i = 1$

- This gives multiplicatively more weights to positive $x_i$'s that could have helped, but you didn't pay them enough attention.

# Winnow Algorithm

- Mistakes on positives:
  - Each mistakes doubles at least one of $r$ relevant weights.
  - Any such weight can be doubled at most $\log n$ times.
  - At most $r \cdot \log n$ mistakes.
- Mistakes on negatives:
  - Initially, total weight is $n$.
  - Each mistake on positive adds $\leq n$ to the total weight.
  - Each mistake on negative removes $\geq n$.
  - #mistakes-on-neg $\leq$ 1 + #mistakes-on-pos
- Overall: At most $1 + 2r \cdot \log n$ mistakes!

# Learning Disjunction: $k$-of-$r$

- Suppose we want to learn a $k$-of-$r$ function.
  - True iff $k$ out of a set of $r$ variables are true.
  - E.g., $f(x) = (x_3 + x_9 + x_{10} + x_{12} \geq 2)$
- Algorithm (Winnow adaptation):
  - Maintain $h(x)$: predict positive iff $\sum_i w_i x_i \geq n$
  - Let $\epsilon = 1/(2k)$.
  - Initialize $w_i \leftarrow 1$ for all $i$
    - Mistake on pos: $w_i \leftarrow w_i(1+\epsilon)$ for all $x_i = 1$
    - Mistake on neg: $w_i \leftarrow w_i/(1+\epsilon)$ for all $x_i = 1$
- Theorem: This makes $O(r\, k \log n)$ mistakes.
- Idea: Think of the algo as adding/removing "chips".

# Winnow: Extensions

- Algorithm:
  - Maintain $h(x)$: predict positive iff $\sum_i w_i x_i \geq n$
  - Let $\epsilon = 1/(2k)$.
  - Initialize $w_i \leftarrow 1$ for all $i$
    - Mistake on pos: $w_i \leftarrow w_i(1+\epsilon)$ for all $x_i = 1$
    - Mistake on neg: $w_i \leftarrow w_i/(1+\epsilon)$ for all $x_i = 1$
- Analysis (chip argument):
  - Each mistake on positive adds $\geq k$ relevant chips.
  - Each mistake on negative removes $\leq k$-1 relevant chips.
  - At most $r\left(\frac{1}{\epsilon}\right)\log n$ relevant chips in total.
  - $k \cdot M_p - (k-1) \cdot M_n \leq \left(\frac{r}{\epsilon}\right)\log n$

# Winnow: Extensions

- Algorithm:
  - Maintain $h(x)$: predict positive iff $\sum_i w_i x_i \geq n$
  - Let $\epsilon = 1/(2k)$.
  - Initialize $w_i \leftarrow 1$ for all $i$
    - Mistake on pos: $w_i \leftarrow w_i(1+\epsilon)$ for all $x_i = 1$
    - Mistake on neg: $w_i \leftarrow w_i/(1+\epsilon)$ for all $x_i = 1$
- Analysis (weight argument):
  - Each mistake on positive adds at most $\epsilon n$ weight.
  - Each mistake on negative removes at least $\frac{\epsilon n}{1+\epsilon}$ weight.
  - $n + \epsilon n \cdot M_p - \frac{\epsilon n}{1+\epsilon} \cdot M_n \geq 0$

# Winnow: Extensions

- Algorithm:
  - Maintain $h(x)$: predict positive iff $\sum_i w_i x_i \geq n$
  - Let $\epsilon = 1/(2k)$.
  - Initialize $w_i \leftarrow 1$ for all $i$
    - Mistake on pos: $w_i \leftarrow w_i(1+\epsilon)$ for all $x_i = 1$
    - Mistake on neg: $w_i \leftarrow w_i/(1+\epsilon)$ for all $x_i = 1$
- Analysis (combined):
  - $k \cdot M_p - (k-1) \cdot M_n \leq \left(\frac{r}{\epsilon}\right) \log n$
  - $n + \epsilon n \cdot M_p - \frac{\epsilon n}{1+\epsilon} \cdot M_n \geq 0$
  - Solve to get that $M_p$ and $M_n$ are both $O(r\,k \log n)$