Cours ENSL: Big Data – Streaming, Sketching, Compression
Today: Plagiarism Detection

Olivier Beaumont, Inria Bordeaux Sud-Ouest
Olivier.Beaumont@inria.fr
Last time: Sketching – Streaming
Application Context

- Internet of Things (IoT) compress locally, send summaries
- Datacenters: dedicated to linear algorithms, inefficient for other classes
- In both contexts, there is a strong need to have data driven algorithms (where placement is imposed by data) and whose complexity is linear
Positionning

- w.r.t. traditional courses on algorithms
  - Exact algorithms for polynomial problems
  - Approximation algorithms for NP-Complete problems
  - Potentially exponential algorithms for difficult problems (going through an ILP for example)

- Here, we will consider extreme contexts
  - not enough space to transmit input data (sketching) or
  - not enough space to store the data stream (streaming)
  - not enough time to use an algorithm other than a linear complexity one (plagiarism)

- Compared to the more "classical" context of algorithms:
  - we aim at solving simple problems and
  - we are looking for approximate solutions only because we have very strong time or space constraints.
Compression associated to a specific function $f$

- More formally, given $f$,
- we want to compress the data $X$ but still be able to compute $\sim f(X)$.
- Sketching: we are looking for $C_f$ and $g$ such that
  - the storage space $C_f(X)$ is small (compression)
  - from $f(X)$, we can recover $f(X)$, ie $g(C_f(X)) \sim f(X)$
- Streaming: additional difficulty, the update is performed on the fly.
  - we cannot compute $C_f(X \cup \{y\})$ from $X \cup \{y\}$
  - since we cannot store $X \cup \{y\}$
  - so we need another function $h$ such that $h(C_f(X), \{y\}) = C_f(X \cup \{y\})$
- and one last difficulty:
- very often, it is impossible to do in deterministic and exact / deterministic and approximate
- but only with a randomized and approximation algorithm.
- How to write this?
  - Given $\alpha$ and $\epsilon$, we are looking for an estimator $Z$ such that
  - $Pr(|Z - f(X)| \geq \epsilon f(X)) \leq \alpha$. 

• $Z$ random variable with positive values
• $E(Z)$ is the expectation of $Z$
• definitions and properties?
  • $E(Z) = \int \lambda P(Z = \lambda) d\lambda$ or $E(Z) = \sum j \cdot P(Z = j)$
  • $E(Z) = \int P(Z \geq \lambda) d\lambda$ or $E(Z) = \sum j \cdot P(Z \geq j)$
  • $E(aX + bY) = aE(X) + bE(Y)$
• To measure the distance from $Z$ to $E(Z)$, we use the variance $V(Z)$
  • Definition?
  • $V(Z) = E((Z - E(Z))^2) = E(Z^2) - E(Z)^2$
• How to measure the difference between $Z$ to $E(Z)$?
  1. Markov: $\Pr(Z \geq \lambda) \leq \frac{E(Z)}{\lambda}$
  2. Chebyshev: $\Pr(|Z - E(Z)| \geq \lambda E(Z)) \leq \frac{V(Z)}{\lambda^2 E(Z)^2}$
  3. Chernoff: If $Z_1, \ldots, Z_n$ are Independent Bernouilli rv with $p_i \in [0,1]$ and $Z = \sum Z_i$, then
     $\Pr(|Z - E(Z)| \geq \lambda E(Z)) \leq 2 \exp(-\frac{\lambda^2 E(Z)}{3})$. 
Morris Algorithm: Counting the number of events

- **Step 1: Find an estimator** $Z$
  - $Z$ must be small (of order of $\log \log n$)
  - we need to define an additional function $g$
  - such that $E(g(Z)) = n$

- **Morris algorithm**
  - $Z \rightarrow 0$
  - At each event, $Z \rightarrow Z + 1$ with probability $1/2^Z$
  - When queried, return $f(Z) = 2^Z - 1$

- **Proof scheme**
  - $E(2^Z) = n + 1$ and $V(2^Z) \leq n^2/2$ and apply Chebychev to obtain $Pr(|f(X_n) = \tilde{n} - n| \geq \epsilon n) \leq \frac{1}{2\epsilon^2}$
  - Do the same experiment plenty of times to obtain $Pr(|f(X_n) = \tilde{n} - n| \geq \epsilon n) \leq \alpha$
  - To obtain a lower constant, use above scheme to get $Pr(|f(X_n) = \tilde{n} - n| \geq \epsilon n) \leq 1/3$ and then use the median of several experiments to conclude
2nd example: how to count the number of unique visitors

Context

- We assume that visitors are identified by their address \((i_k \in [1, n])\)
- We observe a flow of \(m\) visits \(i_1, \ldots, i_m\) with \(i_k \in [1, n]\)
- How many different visitors?
- Deterministic and trivial algorithms:
  - solution in \(n: n\) bit array
  - solution in \(m \log n\): keep the whole stream!
- We have proved (when \(m = \Theta(n)\)) that
  - we cannot do better with exact and deterministic algorithms
  - we cannot do better with approximated and deterministic algorithms
- With only \(O(\log^k n)\) (for some \(k\)), it is possible by using both randomization and approximation.
We start with an idealized algorithm (one that cannot be implemented in practice).

- Let us choose a random $h$ function from $[1, n]$ to $[0, 1]$
- Why idealized?
  - Problem 1: to store such a random function, you must define the images for in each of the $n$ points... at least $\Omega(n)$ bits
  - Problem 2: and in addition we would have to store real values!
  - Let us assume for now that storing such a function costs $\Theta(1)$
- How to keep track of the number of unique visitors?
  - We keep $Z \rightarrow \min_{i \in \text{stream}} h(i)$. Intuition?
    - If you see the same visitor $k$ times, it won’t change $Z$
    - If we see $t$ different visitors, then the values taken by $h$ split $[0, 1]$ in $t + 1$ intervals... and all should have the same size in expectation... and this size is $\frac{1}{t+1}$ including the first!
  - so we should return $\frac{1}{Z} - 1$!
Non Idealized algorithm – Hash functions

- To make it practical, we will consider a large (but not too large) family of functions $\mathcal{H}$ from $[1, p] \rightarrow [1, p]$.
- Notion of $k$-wise independence
  - $\forall i_1, \ldots, i_k, \forall j_1, \ldots, j_k, i_k \neq i_l$, and if we pick a random $h$ function in $\mathcal{H}$, then
  - $P(h(i_1) = j_1$ and $h(i_k) = j_k) = 1/p^k$
  - A larger value for $k$ provides a "better" family.
- Example: the set of polynomials $\mathcal{H}^k_{\text{poly}}$ of degree $k$ in $F_p$
  - Evaluation cost? For degree $k$, $k$ mult & and adds
  - Independence? How many polynomials such that $(h(i_1) = j_1$ and $h(i_k) = j_k$)
  - Exactly one, Lagrange polynomial: $P = \sum_{r=1}^{k} \prod_{l \neq r} (X-i_l) \times j_r$
  - Choice? Picking a function at random in $\mathcal{H}^k_{\text{poly}} \rightarrow$ picking $k + 1$ coefficients.
  - And thus the family $\mathcal{H}^k_{\text{poly}}$ is $k$-independent.
- Can be easily generalized to different sizes $F_p \rightarrow F_q$. 


Finding Similar Itemsets
General Idea

- 2 type of difficulties related to
  - the number of objects: $N$ objects $\rightarrow N^2$ comparisons
  - the objects themselves: large texts,...

- Applications
  - pages with a lot of text in common (mirror sites, approximate mirror)
  - plagiarism (today)
  - group news that deal with the same event
  - Amazon, Netflix: users with the same taste
  - dual: Amazon, Netflix: products with the same fans

- we will concentrate on texts, the first step only is application specific
  - Order of magnitude: $10^6$ documents, size a few MB not huge (a few TB)
  - Distributed over a datacenter: large number of nodes $10^3 - 10^6$ nodes

- Two goals:
  - avoid moving data between the nodes (small and shared bandwidth)
  - avoid performing $10^{12}$ comparisons: both for time and data movements
Techniques

- 3 steps
  1. Shingling: conversion of a large text into a (large) set
  2. Min-hashing: assign to each text a (small) similarity-preserving signature
  3. Locality Sensitive Hashing: detect suspect pairs by collision detection

- randomized approximation algorithm \(\rightarrow\) errors: false positive and false negative

- what is crucial in our context? Complexity: we want to deal with linear complexity algorithms only!

- Remark: we assume that the output is (at most) of linear size (otherwise, we have no chance!)
• $k$-shingle
  • sequence of $k$ successive characters in the text
  • $\{abcab\}$ et $k = 2$ ?

• Observation (admitted) close texts $\rightarrow$ a lot of shingles in common

• A text: represented by the set of shingles it contains

• How many shingles ? with $k = 10$

• The data structure should enable to perform comparisons easily...
  • $30^{10}$ shingles $= 2^{49}$
  • size if stored as a vector of bits 70 TB
  • what happens in practice? Solution? shingles $\rightarrow$ tokens
  • first use of hashing functions: adapt the size, control collisions
• Each text is associated to a set of items
• We need to define a similarity between sets.
• Jaccard Similarity: $Sim(C_1, C_2) = \frac{C_1 \cap C_2}{C_1 \cup C_2}$
• $d(C_1, C_2) = 1 - Sim(C_1, C_2)$ is a distance (proof later)
  • one vector per document
  • one row per token
• How to compute the similarity between two documents?
• Problems:
  • we do not want to deal with $N^2$ pairs
  • we cannot centralize all $N$ pairs at a single node
Minhashing

- Let us suppose that \((C_1, C_2)\) are stored at the same place
- Goal: build a small similarity preserving signature for each document.
- General Idea: build a random game whose expected value (to win) is \(Sim(C_1, C_2)\)
- Do we really need to have \((C_1, C_2)\) at the same place to play the game?
- Do we really need permutations?
- How many hash functions do we need in order to obtain a good precision?
Locality Sensitive Hashing

- So far: we have a very compact summary of each document (250 × 4B integers = 1kB)
- Last step: given a suspicion threshold $s \in [0, 1]$, return all pairs $(C_1, C_2)$ such that $\text{Sim}(C_1, C_2) > s$
- Without doing all comparisons!
- Order of magnitude:
  - $10^6$ documents $\rightarrow 1GO$, Ok en mémoire
  - $10^{12}$ comparisons with $10^{-6}$s per comparison 12 days )-;
- Goal: go from quadratic to linear complexity
- using hash functions again and collision detection
- now, we want close vectors to collide, and distant vectors not to collide
locally sensitive hashing

- split the summary (250 integers) into $b$ blocks of size $r$ ($rb = 250$)
- let $h_k$ be the hash function associated to the $k$-th block
- collision: (almost) only if both vectors coincide on this block. Solution: use a large number of buckets (with respect to $10^6$) $\rightarrow 10^9$ is Ok in practice, very few false positive.
- a pair $(C_1, C_2)$ is suspicious if $\exists k, h_k(B^1_k) = h_k(B^2_k)$ where $B^i_k$ is the $k$-th block of $C_i$
- what happens ?
  - if $r$ is too small ? too many false positive
  - if $r$ is too big ? we will miss similar itemsets and get false negative
- Given $r$ (and thus $b$) and $s = Sim(C_1, C_2)$, what is the probability that a collision occurs ?
with $N = 100$ and $r = 3, 4, 5$

false positive? false negative?
with $N = 1000$ and $r = 3, 4, 5$
• distributed documents.
• keep everything local (until the computation of signatures)
• keep everything local (compute the hashing of each block) 1 document + 1 block $\rightarrow$ 4B !
• gather all information related to one block number to the same node (4B + 1B for the index) $\rightarrow$ 5MB
• detect all suspicious pairs (very few and send them to a specific node)... 
• very few communications !
Locality Sensitive Hashing and Nearest Neighbors Search
(k-) nearest neighbors

- Metric space with distance, set $P$ of points
- preprocessing allowed on $P$
- Query: given a point, find its ($k$) closest neighbor(s)
  - example for spam classification: start with a huge annotated emails
  - one word = one item
  - return the $k$ closest emails, majority vote to determine if it is spam or not
- Approach # 1: no preprocessing, just look through all possible items
  - space $O(dn)$
  - query $O(dn)$
- Approach # 2: if $d=1$
  - space $O(n)$ just keep the boundaries
  - query $O(\log n)$ just a basic binary search
- Approach # 3: if $d=2$
  - Voronoï diagrams: space $O(n)$ and computing cost $O(n \log n)$
  - query time easy (locate the cell)
  - As dimension increase, the description increases exponentially with $d$
- all exact (known) approaches in high dimension either have
  - exponential space $O(n^d)$
  - or exponential query time !
  - (same for $kd$-trees)
  - in very. large dimension, the naive algorithm is not that bad !
Given a set of $P$ points, construct a data structure such that
- on query $q$, we return $p$ in $P$ such that $d(p, q) \leq c \min_{p' \in P} d(p', q)$

$(r_1, r_2)$PLEB problem: point location in equal balls
- given a set $P$ of points and $r_1, r_2$
- construct a data structure to answer as follows
- If $\exists p \in P$ st $d(p, q) \leq r_1$, return YES and any $p' \in P$ s.t. $d(p', q) \leq r_2$
- If there is no $p \in P$ st $d(p, q) \leq r_2$, return NO
- elif don't care what algorithm returns
Locality Sensitive Hashing (Indyk, Motwani)

- Usually, we want hashing functions to "shuffle" items as much as possible
  - When writing $P(h(i_1) = j_1 \& h(i_2) = j_1) = 1/p^2$, we say that the distance between images should not depend on the distance between initial points.

- Here, we want to detect "collisions"
  - we want close points to have a high probability to collide
  - we want distant points to have a low probability to collide
  - just as in the context of plagiarism.

- General idea
  - hash items into many different buckets (with different functions)
  - declare that there is a collision if two items fall into the buckets

- Formal definition $\mathcal{H}$ a family of hash function $U \rightarrow S$
  - (where $U$ is the set of points, $S$ the set of buckets)
  - is said to be $(r_1, r_2, p_1, p_2)$ locality sensitive if
    - If $d(p, q) \leq r_1$, then $P_{h \in \mathcal{H}}(h(p) == h(q)) \geq p_1$ and
    - If $d(p, q) \geq r_2$, then $P_{h \in \mathcal{H}}(h(p) == h(q)) \leq p_2$
  - of course $r_1 < r_2$, $p_1 > p_2$
Example

- Let $H^d = \{0, 1\}^d$ equipped with Hamming distance (number of different coordinates)
- Let $H = \{h_i, \forall i, \text{ where } h_i(b_1, \ldots, b_d) = b_i\}$
- $H$ is $(r, cr, 1 - r/d, 1 - cr/d)$ locality sensitive
  - if $p, q$ are at distance at most $r$, they have at least $(d - r)$ coordinates in common and thus a probability at least $1 - r/d$ to be hashed similarly,
  - if $p, q$ are at distance at least $cr$, they have at most $(d - cr)$ coordinates in common and thus a probability at most $1 - cr/d$ to be hashed similarly.
Master Theorem of LSH

**Theorem**

Suppose $\exists (r_1, r_2, p_1, p_2)$-LSH family, then there is an algorithm for $(r_1, r_2)$-PLEB with answer queries with constant probability (it might be wrong), and that uses space $O(dn + n^{1+\rho})$ and query time $O(n^\rho)$ (evaluation of hash functions), where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$ (complexity decreases when $\rho$ decreases, i.e., when $p_2 << p_1$).

**Sketch of the proof — Algorithm**

- let $(k, l)$ be parameters (t.b.d. later), let $G$ be a family of hash functions from $U$ to $S^k$ (new buckets), $g(p) = (h_{g_1}(p), \ldots, h_{g_k}(p))$ each $h_{g_i}$ being randomly chosen in $H$.

- **Preprocessing:**
  - (1) choose $g_1, \ldots, g_l$ (other parameter) independently from $G$
  - (2) for each $p \in P$, store $g_1(p), \ldots, g_l(p)$

- **On query**
  - (1) search the points of $P$ in $g_1(q), \ldots, g_l(q)$, but stop after the first $2l$ points in (the unlikely) case there are more than $2l$.
  - (2) If there is one point $p$ such that $d(p, q) \leq r_2$ return it and return YES, otherwise return NO
Proof (continued)

• Intuition (1): if \( q \) and \( p \) are "close", then one the \( g_i \) will send them into the same \( k \)-bin.
• Intuition (2): it is unlikely that they are \( 2l \) distant (useless) points in the set (that would prevent to find the useful point)
• (2) There are at most \( 2l - 1 \) points st \( d(p, q) > r_2 \) and \( \exists j, g_j(p) = g_j(q) \) with constant probability
  • Let \( k = \log_{1/p_2} n \), what is the expected number of points st (2) holds ?
  • If \( d(p, q) > r_2 \) then for each \( h \), the probability of collision is at most \( p_2 \)
  • so the expected number of times \( p \) is written in any \( g_j \) is \( p_2^k \), and the expected number of times it is written for a given \( g \) is \( lp_2^k \)
  • and the number of "bad points" written for \( g \) is therefore at most \( nlp_2^k = l \)
  • Markov says \( P(X > \lambda) < E(X)/\lambda \) so \( P(\text{more than } 2l \text{ bad points}) \leq 1/2 \)
• (1) If \( p \in P \) with \( d(p, q) \leq r_1 \), then \( \exists j, g_j(p) = g_j(q) \) with constant probability
  • If \( d(p, q) \leq r_1 \) then for each \( h \), the prob of collision for \( h \) is at least \( p_1 \)
  • the probability of collisions in one bucket is \( p_1^k \)
  • the probability of a collision in at least one of the \( l \) buckets is \( 1 - (1 - p_1^k)^l \)
  • choice of \( l \) ? if we set \( l = n^p \) then the probability is at least \( 1 - 1/e \approx 0.63 \).
• to increase the probability, use the classical and tricks
• Check space and time complexities
Conclusion on sketching/streaming/compression

- Goal: data flow \( X \) and a function to evaluate \( f \)
- Streaming: maintain a summary \( C_f(X) \) enough to compute \( f(X) \approx g(C_f(X)) \)
  - Solution: Use approximation randomized algorithms
  - \( \forall \epsilon, \delta \), \( Pr(\text{relative error} \geq \epsilon) \leq \delta \)
  - enough (and often necessary) to change space complexities (from \( \log n \rightarrow \log \log n \), \( n \rightarrow \log n \), from \( n^d \) to \( n^\rho d \))
  - at the price of sometimes large constants
  - but constants are pessimistic
  - and very small \( \epsilon \) and \( \alpha \) are not always required (plagiarism)
- General Idea:
  - Do less communications (same a lot of energy, time)
  - But more local computations (cheap)
  - crucial for IoT and datacenters
- Method:
  - Find an estimator \( Z \) tel que \( E(Z) = \) what we want to estimate
  - go for + and ++ versions to control the probability
  - hash functions are a very powerful and versatile tool:
    - to shuffle potentially correlated entries (Unique Visitors)
    - to adapt the size of sets (Plagiarism)
    - to create short summaries (Min-Hashing)
    - to detect close items (Locality Sensitive Hashing)
A few resources on the web

- Algorithms for Big Data (CS 229r), Jelani Nelson (Harvard, now MIT)
- Algorithms for Big Data (CS 598) Chandra Chekuri (Urbana Champaign)
- Data Streams Algorithms (CS711) Andrew McGregor (UMass Amherst).
- Dealing with Massive Data (COMS 6998) Sergei Vassilvitskii (Columbia).