Randomized Algorithms, Summer 2016 Lecture 4 (6 pages)

Bloom Filters and Locality-Sensitive Hashing

Instructor: Thomas Kesselheim and Kurt Mehlhorn

1 Notation

When we talk about the probability of an event, the underlying probability space is usually implicit. Sometimes, we want to make the underlying probability space explicit. Assume we have a set S and an event $\mathcal{E}$ depending on an element $x \in S$. We write

$$\mathbf{Pr}_{x \in S}\left[\mathcal{E}(x)\right].$$

to stress that the underlying probability space is S and that x is drawn uniformly at random from S. If x is drawn according to a more general distribution $\mathcal{D}$, we write

$$\mathbf{Pr}_{x \sim \mathcal{D}}\left[\mathcal{E}(x)\right].$$

For expectations, we use the analogous notation.

2 Bloom Filter

This section follows 5.5.3 in Mitzenmacher/Upfal.

Bloom Filters are a compact data structure for *approximate membership queries*. The data structure uses only linear space. It makes mistakes in the sense that non-members may be declared members with small probability.

Let $S = \{e_1, \ldots, e_m\}$ be the elements to be stored; S is a subset of some universe U. We use a boolean array T of length n. We assume that we have k *hash functions* that map U to the integers from 1 to n. The hash functions are assumed to be independent and random.

We initialize T to the all-zero array and then set $T[i]$ to one iff there is an element $e_j \in S$ and a hash function h_ℓ such that $h_\ell(e_j) = i$. Note that an array entry may be set to one by several elements.

In order to answer a membership query for e, we compute the hash values $h_\ell(e)$, $1 \le \ell \le k$, and return "YES" if and only if $T[h_\ell(e)] = 1$ for all ℓ.

Clearly, if $e \in S$, the answer will be YES. Also, if the answer is NO, then $e \notin S$. However, there is the possibility of false positives, i.e., the data structure might answer YES for elements $e \notin S$. What is the probability that e is a false positive?

Let us first compute the probability that a particular $T[i]$ stays zero. This is

$$\left(1 - \frac{1}{n}\right)^{km} = \left(1 - \frac{1}{n}\right)^{n \cdot km/n} \approx e^{-km/n}.$$

Set $p = e^{-km/n}$. Then pn is (approximately) the expected number of array entries that are zero. The events "$T[i]$ is zero" and "$T[j]$ is zero" are NOT independent.

We proceed under the simplifying assumption that a random fraction p of the table entries are zero (a random fraction $1 - p$ of the entries are one) after S has been stored in the table. Then the probability that $e \notin S$ is a false positive is

$$f = (1 - p)^k = (1 - e^{km/n})^k.$$

What is a good choice of k for given m and n? There are two effects working against each other. Having more hash functions helps to discover an error because all k functions must be fooled for a false positive. Having fewer hash functions leaves more table entries equal to zero and this makes it harder to fool a particular function. We use calculus to determine the optimum. Let $g = \ln f = k \cdot \ln(1 - e^{-km/n})$. Then

$$\frac{dg}{dk} = \ln(1 - e^{-km/n}) + k\frac{-e^{-km/n}}{1 - e^{-km/n}} \cdot (-\frac{m}{n}) = \ln(1 - e^{-km/n}) + \frac{km}{n}\frac{e^{-km/n}}{1 - e^{-km/n}}.$$

The derivative is zero for $km/n = \ln 2$; note that $\frac{dg}{dk} \mid_{km/n = \ln 2} = \ln(1/2) + (\ln 2) \cdot \frac{1/2}{1/2} = 0$. It is easy to check that this value of k is a local minimum.

For $k = \frac{n}{m} \ln 2 \approx 0.6185 \frac{n}{m}$, we have $p = 1/2$ and $f = 2^{-k}$. For example, if we want $k = 6$ and hence $f \leq 0.02$, we need $n \geq k/0.6185m \approx 10m$. *This is very significant.* Using only $10m$ bits, we can answer membership queries with the probability of false positives being below 0.02. However, in order to store S, we would need $m \log U$ bits.

Is our simplifying assumption justified? See Mitzenmacher and Upfal.

3 Locality-Sensitive Hashing.

This section follows Chapter 3 "Finding Similar Items" of the book "Mining of Massive Data Sets" by Jure Leskovec, Anand Rajarmadan, and Jeff Ullman. The book can be downloaded at `mmds.org`.

3.1 The Jaccard Similarity of Two Sets

Let S and T be two sets from a common ground set U. We define their *Jaccard similarity* as

$$J(S,T) = \frac{|S \cap T|}{|S \cup T|},$$

as the ratio between the size of their intersection and the size of their union. For $S = \{a, c, f\}$ and $T = \{b, c, f\}$ we have $J(S,T) = 2/4 = 1/2$.

Exercise 1. *Extend the definition to multi-sets.*

Exercise 2. *Let $d(S,T) = 1 - J(S,T)$ be the Jaccard distance of two sets S and T. Prove that d is a metric.*

3.2 MinHash

Assume that the ground set is ordered.

For every permutation π of the ground set U, we define a function h_π that maps any subset S of U to its minimal member, i.e.,

$$h_\pi(S) = \min\{\pi(x) \mid x \in S\}.$$

Let H be the set of such functions.

Theorem 4.1. *For any two sets S and T*

$$J(S,T) = \mathbf{Pr}_{h_\pi \in H}\left[h_\pi(S) = h_\pi(T)\right].$$

Proof. Let $d = |S \cap T|$ and $m = |S \cup T|$. The probability that the first element of a random permutation of $S \cup T$ comes from $S \cap T$ is d/m. □

We can therefore estimate the Jaccard similarity of two sets S and T by choosing a number of permutations, say π_1 to π_k, and computing $\frac{1}{k}\sum_{1 \leq i \leq k}[h_{\pi_i}(S) = h_{\pi_i}(T)]$.

Lemma 4.2. *The Jaccard-distance $d_J(S,T) = 1 - J(S,T)$ is a metric.*

Proof. We have

$$d_J(S,T) = 1 - J(S,T) = \frac{|S \Delta T|}{|S \cup T|} = \mathbf{Pr}\left[h_\pi(S) \neq h_\pi(T)\right].$$

Clearly, $d_J(S,S) = 0$ and $d_J(S,T) = d_J(T,S)$. Let S, T, U be three sets. For any permutation π: if $h_\pi(S) \neq h_\pi(T)$ then $h_\pi(S) \neq h_\pi(U)$ or $h_\pi(T) \neq h_\pi(U)$. Thus

$$d_J(S,T) = \mathbf{Pr}\left[h_\pi(S) \neq h_\pi(T)\right] \leq \mathbf{Pr}\left[h_\pi(S) \neq h_\pi(U)\right] + \mathbf{Pr}\left[h_\pi(T) \neq h_\pi(U)\right] = d_J(S,U) + d_J(U,T).$$

□

Remark 4.3. *The set H is too big for practical purposes. Note that $|H| = |U|!$ and hence $|U| \log |U|$ bits are required to specify a function in H. There is significant research on identifying smaller sets of functions which give essentially the same performance as the full set.*

3.3 Locality-Sensitive Hashing

LSH generalizes what MinHash achieves for Jaccard distance.

Let M be a metric space with distance function d and let F be a set of functions defined on M. Let d_1 and d_2 be two distance values with $d_1 < d_2$ and let p_1 and p_2 be two reals with $1 \geq p_1 > p_2 \geq 0$. F is said to be (d_1, d_2, p_1, p_2)-*sensitive* if for every x and y in M:

- If $d(x, y) \leq d_1$ then $\mathbf{Pr}_{x \in F}[h(x) = h(y)] \geq p_1$.
- If $d(x, y) \geq d_2$ then $\mathbf{Pr}_{x \in F}[h(x) = h(y)] \leq p_2$.

In words, if x and y are "close" (distance at most d_1) then they are hashed to the same value with probability at least p_1 and if x and y are "far" (distance at least d_2) then they are hashed to the same value with probability at most p_2.

The definition above "translates" a question about the distance of two points (which might be hard to compute and requires both points as arguments) into a probabilistic question about the equality of hash values. When we store elements in buckets according to hash values then elements that are close are more likely in the same bucket than elements that are far. We will use this in a data structure for nearest neighbor search in the next section.

Lemma 4.4. *Let $0 \leq d_1 < d_2 \leq 1$ be arbitrary. The set H of Minhash-functions is $(d_1, d_2, 1-d_1, 1-d_2)$-sensitive for Jaccard-distance.*

Proof. For any $d \in [0, 1]$ and any two sets S and T, we have

$$\mathbf{Pr}_{h \in H}[h(S) = h(T)] = J(S, T) = 1 - d_J(S, T).$$

The claim follows. □

3.4 Approximate Nearest Neighbor Search

This follows the Wikipedia entry on LSH.

Let M be some metric space. We assume that it takes time $O(d)$ to compute the distance of two points. Let F be a (d_1, d_2, p_1, p_2)-sensitive family. We describe a data structure that succeeds in finding a point from a set $S \subseteq M$ within distance d_2 from a query point q with probability at least $1 - (1 - p_1^k)^L$ if there exists a point within distance d_1; here k and L are parameters that influence the preprocessing time and the space requirement.

An Application: Screening Fingerprints Suppose we have a large collection (several million) of fingerprints of criminals. We want to query a fingerprint x for membership in the collection without searching the entire collection, say at a border control. We are willing to accept false negatives, i.e., fingerprints that are in the collection, but are declared non-members, and false positives, i.e., fingerprints that are not in the collection, but are declared members. Actually, the method will identify a subset of the collection which then has to be inspected in detail. So false positives are only a problem in the sense that somebody may be kept under custody unnecessarily long until the detailed inspection of the subset. False negatives are not a problem in fast screening. If the rate of false negatives is sufficiently low, say below 5 %, criminals might not be willing to take the risk of being tested.

In a preprocessing step, fingerprints are converted into arrays of size 100 by 100. Some of the array entries are marked as interesting. Interesting squares correspond to interesting features in the fingerprint, e.g., merging of two lines. Interesting features in a fingerprint are called minutiae. We make the following assumptions.

- About 20% of the squares are interesting. We convert this into a probabilistic statement by assuming that a random square is interesting with probability 0.2 and the events of being interesting are independent for different squares.
- If two fingerprints come from the same finger then for any particular square the events of being interesting are highly correlated. More precisely, if a square is interesting in one fingerprint, then with probability 0.8 it is also interesting in the other fingerprint.

A fingerprint is a bitstring of length 10^4, in which about $2 \cdot 10^3$ bits are one. Fingerprints from the same finger have a small hamming distance because they have about 1600 one-bits in common. Thus there Hamming distance is about 800. On the other hand fingerprints coming from different fingers share only about $(0.2)^2 \cdot 10^4 = 400$ one-bits. Thus their Hamming distance is about 3200. We set up the data structure for $d_1 = 1000$ and $d_2 = 2500$.

An Application: Discovering Similar Documents There are many documents on the web that differ only in insignificant ways, e.g., the printing date or the printing location. Search engines may want to index only one copy of such documents.

The Data Structure We have L tables. For each table we choose k random functions f_1 to f_k from F and consider the function f that maps any point p to $(f_1(p), \ldots, f_k(p))$. We partition S into buckets according to the value of f, i.e, for each tuple $(z_1, \ldots, z_k)$ we store all elements $p \in S$ with $(z_1, \ldots, z_k) = (f_1(p), \ldots, f_k(p))$ in a linear list associated with the tuple. Since the total length of the lists is $O(n)$, we can use standard hashing to organize the lists in linear space. More precisely, we use $(f_1(p), \ldots, f_k(p))$ as the key for p in a standard hashing scheme.

The preprocessing time is $O(nLkt)$ where t is the time to evaluate a function $h \in F$ on an input point p. The space requirment is $O(nL)$ plus the space for storing data points.

The Query Algorithm Let q be the query point. For each of the L tables, we compute $f(q) = (f_1(q), \ldots, f_k(q))$ and inspect all elements in the bucket associated with $f(q)$. We stop once an element with distance at most d_2 from q is found.

The query time is $O(L(kt + dnp_2^k))$. We are inspecting L tables. For each table we need to compute $f(q)$. This takes time tk. We then need to inspect the bucket associated with $f(q)$. Let us estimate the number of unsuccessful comparisons, i.e., comparisons between q and $p \in S$ with $d(q,p) > d_2$. Recall that we stop once a successful comparison occurs. For a point $p \in S$ whose distance from q exceeds d_2, the probability that p is contained in the bucket selected by q is p_2^k. Thus we need to compare q to an expected number of np_2^k points. Each comparison takes time $O(d)$.

The success probability is at least $1 - (1 - p_1^k)^L$ if some point $p \in S$ with distance at most d_1 from q exists. Indeed, consider a fixed table. The probability that p is mapped into the same bucket as q by all f_i $1 \leq i \leq k$ is p_1^k. Hence the probability that p is not mapped into the same bucket as q for a particular table is $1 - p_1^k$.

Therefore the probability that p is not mapped into the same bucket as q in any table is $(1 - p_1^k)^L$. Thus the algorithm succeeds with probability at least $1 - (1 - p_1^k)^L$.

Selecting k and L Let $\rho = \frac{\log p_1}{\log p_2}$ and set $k = \frac{\log n}{\log 1/p_2}$ and $L = n^\rho$. Note that $\rho < 1$. Then one obtains the following performance guarantees:

- preprocessing time: $O(n^{1+\rho}kt)$;
- space: $O(n^{1+\rho})$, plus the space for storing data points;
- query time: $O(n^\rho(kt + d))$; note that $np_2^k = n2^{\log p_2 \cdot \frac{\log n}{\log 1/p_2}} = n \cdot \frac{1}{n} = 1$;
- success probability: $1 - 1/e$; note that $(1 - p_1^k)^L = (1 - \frac{1}{n^\rho})^{n^\rho} \approx 1/e$.

3.5 Probability Amplification

We introduce two methods for constructing new families of hash functions from existing families. The effect of these constructions is illustrated by Figure 1. We used both methods in the data structure for nearest neighbor search.

And-Construction Let k be an integer and let f_1 to f_k be k random functions from F. Let $f(x) = f(y)$ iff $f_i(x) = f_i(y)$ for $1 \leq i \leq k$ and let F_{and}^k be the set of such functions.

Lemma 4.5. *F_{and}^k is (d_1, d_2, p_1^k, p_2^k)-sensitive.*

Proof. Obvious. □

Or-Construction Let k be an integer and let f_1 to f_k be k random functions from F. Let $f(x) = f(y)$ iff $f_i(x) = f_i(y)$ for some $i \in [k]$ and let F_{or}^k be the set of such functions.

Lemma 4.6. *F_{and}^k is $(d_1, d_2, 1 - (1 - p_1)^k, (1 - (1 - p_2)^k)$-sensitive.*

Proof. We have $[f(x) \neq f(y)]$ iff $[f_i(x) \neq f_i(y)]$ for all i. □

p	$1-(1-p^4)^4$
0.2	0.0064
0.4	0.0985
0.6	0.4260
0.8	0.8785

p	$(1-(1-p)^4)^4$
0.2	0.1215
0.4	0.5740
0.6	0.9015
0.8	0.9936

Figure 1: The left table shows the effect of a 4-way AND-construction followed by a 4-way OR-Construction. The right tables shows the effect of a 4-way OR-construction followed by a 4-way AND-Construction.

3.6 LSH for Hamming Distance

$M = \{0,1\}^n$ for some n and the distance between two strings is the number of positions in which they differ. Let F be the set of projections of M onto the coordinates, i.e., f_i maps a string x onto its i-th coordinate.

Let d_1 and d_2 be positive integers with $0 \leq d_1 < d_2 \leq d$. The family of projections is $(d_1, d_2, 1 - d_1/d, 1 - d_2/d)$ sensitive.

3.7 LSH for Cosine Distance

The *cosine-distance* between two vectors from the same ambient space is the angle between them. Note that the two vectors span a plane and the angle is defined as the angle in this plane. Algebraically,

$$\cos\varphi = \frac{\langle x, y\rangle}{|x| \cdot |y|}$$

Assume our underlying space is $\mathbb{R}^n$. For every direction v (= a point on the n-dimensional hypersphere), define

$$h_v(x) = sign(\langle v, x\rangle).$$

and let F be the set of such functions.

Lemma 4.7. *Let S_n be the n-dimensional hypersphere, let x and y be nonzero vectors in $\mathbb{R}^n$, and let φ be the angle between them. Then*

$$\mathbf{Pr}_{v \in S_n}\left[h_v(x) = h_v(y)\right] = 1 - \varphi/\pi$$

Proof. Let H_v be the hyperplane through the origin with normal vector v and let L_v be the intersection of H_v with the plane spanned by x and y. We may assume that L_v is a line as the case that L_v is the entire plane has probability zero. The scalar products of v with x and y have the same sign iff x and y lie on the same side of H_v if and only iff x and y lies on the same side of L_v. This restrict the direction of the normal of L_v to a double cone with opening angle $\pi - \varphi$. □

Lemma 4.8. *The family of inner product functions is $(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180)$-sensitive for the cosine-distance.*

Lemma 4.9. *In order to choose a point v on the n-dimensional sphere choose v_1 to v_n independently according to the normal distribution, i.e., $\mathbf{Pr}\left[v_i = z\right] = \frac{1}{\sqrt{\pi}} e^{-z^2}$, and return*

$$\frac{1}{\sqrt{v_1^2 + \ldots + v_n^2}} \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}.$$

Proof. For reals z_1 to z_n, we have

$$\mathbf{Pr}\left[v_1 = z_1, \ldots, v_n = z_n\right] = \prod_{1 \leq i \leq n} \frac{1}{\sqrt{\pi}} e^{-z_i^2} = \frac{1}{\sqrt{\pi}^n} e^{-\sum_i z_i^2} = \frac{1}{\sqrt{\pi}^n} e^{-R^2},$$

where $R^2 = \sum_{1 \leq i \leq n} z_i^2$, i.e., all points on the sphere of radius R are equally likely. □

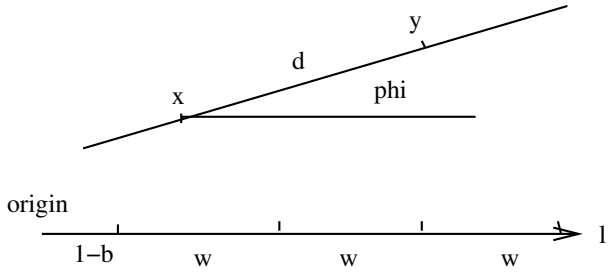

Figure 2: The projection of points x and y onto a random direction ℓ. The angle φ between ℓ and the line through x and y is uniformly random. The line is divided into buckets of width w. The first bucket starts at distance $1 - b$ from the origin. b is random in $[0, w]$.

3.8 LSH for Euclidean Distance

Let v be a random direction (think of v as a coordinate direction). We partition the direction into intervals of length w and give the origin a coordinate equal to b. We hash x onto the index of the interval containing its projection onto direction v. Formally, for a direction v and a scalar b, we define

$$h_{v,b}(x) = \left\lfloor \frac{v^T x + b}{w} \right\rfloor.$$

Lemma 4.10. *The family of functions $h_{v,b}$ is $(w/2, 2w, 1/2, 1/3)$-sensitive.*

Proof. Let d be the distance between x and y. Consider the situation in the plane spanned by the direction v and the line through x and y. Let φ be the angle between v and the line through x and y, see Figure 2. The angle is random.

If $d \leq w/2$, x and y have the same hash value with probability at least $1/2$.

Let d' be the distance between the projections of x and y onto ℓ. Then $d' = d \cos \varphi$. If $d \geq 2w$, then $d' \leq w$ implies $\varphi \in [60°, 90°]$. Thus with probability at least $2/3$, x and y are not hashed into the same bucket. □

Remark 4.11. *We have given the analysis for $d_1 = w/2$ and $d_2 = 2w$. For any d_1 and d_2 with $d_1 < d_2$, the family is (d_1, d_2, p_1, p_2)-sensitive for some $p_1 > p_2$.*

We close with the observation that the random offset b compensates the discrete measurement; this observation is also useful otherwise.

Lemma 4.12. *For every v,*

$$\mathbf{E}_{b \in [0,w]} \left[h_{v,b}(x) \right] = \frac{v^T x}{w}.$$

Proof. Let $z = \frac{v^T x}{w} - \left\lfloor \frac{v^T x}{w} \right\rfloor$. Then $h_{v,b}(x) = \left\lfloor \frac{v^T x}{w} \right\rfloor$ if and only if $b \leq 1 - z$. Thus

$$\mathbf{E}_{b \in [0,w]} \left[h_{v,b}(x) \right] = (1 - z) \left\lfloor \frac{v^T x}{w} \right\rfloor + z(\left\lfloor \frac{v^T x}{w} \right\rfloor + 1) = \frac{v^T x}{w}.$$

□