

1 Introduction

In this lecture we will get to know several techniques that can be grouped by the general definition of sketching. When using the sketching technique each element is replaced by a more compact representation of itself. An alternative algorithm is run on the more compact representations. Finally, one has to show that this algorithm gives the same result as the original algorithm with high probability.

This technique is shown through two example problems:

- Syntactic Clustering of the Web: Given a large set of documents, output them grouped by “similarity” groups.
- Calculating Hamming Distance: Given two strings, calculate the range of the number of bits in which they differ.

Through these examples we will become familiar with the idea of sketching algorithms and get to know such important concepts as shingles.

2 Syntactic Clustering

The problem of syntactic clustering of documents is, given a large document set, identify groups of documents that are very similar in content. The motivation for the problem is to eliminate near duplicate documents. It turns out there are a lot of near-duplicate documents on the web. Examples include mirror sites, manuals, different versions of the same document and even plagiarism. The reason we want to eliminate duplicate documents is that it hurts search engines. It both increases index size and harms the quality of the results.

2.1 The “naive” approach

The “syntactic clustering of the web” problem, as defined by Broder et al in 97 [1], can be stated as follows. Let S be a collection of documents. Let sim be a similarity measure between two documents from the collection. sim maps two documents to a number between 1 and 0. The closer the sim is to 1, the more similar the documents are. Let us now define a graph G which will represent the documents and their relationships. Every document is represented as a vertex, an edge is defined between two vertices if the documents they represent are similar beyond a certain threshold. Now, if we find the connected components of G we will find the syntactic clusters, and that is exactly what we are looking for.

It is worthwhile to note that a run-on mistake may happen if a sequence of documents, each document similar to the next. However, the last document is not similar to the first. It may be assumed that such sets are unlikely in the web.

So it would appear we have solved our problem? However, the usual large data set problems occur when attacking this problem naively. The document collection is huge and it is natural to assume the access to this document collection should be done as a stream, and not by random access. The computer memory is tiny compared to S . Anything beyond $O(|S|)$ complexity is too much. Creating the graph G naively will take $|S|^2$, and that is just too much.

2.2 Let's Sketch

So we want to reduce the complexity of the algorithm by employing sketching. It seems natural to start by taking each document in S and replace it with an 8 byte representation. This gives us a more compact representation T of S .

This representation is obtained by employing a *compression scheme*, a randomized mapping $\phi: S \rightarrow T$. The *reconstruction function* is close to the similarity function but operates directly on the compact representation. For every pair p, q we have with high probability $\psi(\phi(p), \phi(q)) \approx \text{sim}(p, q)$.

1. $P \leftarrow$ empty table of size $|S|$
2. $G \leftarrow$ empty graph on $|S|$ nodes
3. for $i = 1, \dots, |S|$
4. read document p_i from the stream
5. $P[i] \leftarrow \phi(p_i)$
6. for $i = 1, \dots, |S|$
7. for $j = 1, \dots, |S|$
8. if $(\psi(P[i], P[j]) \geq t)$
9. add edge (i, j) to G
10. output connected components of G

Table 1: The framework for our sketching algorithm

The sketches can be calculated using one pass over the stream. The table P can be stored in a single file on a single machine. Creating G requires $|S|^2$ applications of ψ which is easier than the full-fledged computation of ψ , however, quadratic time is still a problem and will not work with large values of S . The connected components algorithm is still heavy but feasible.

2.3 Sketching using Locality Sensitive Hashing

An alternative method of attacking the syntactic clustering problem is by using *locality sensitive hashing* or LSH for short. LSH can also be thought of as a type of a sketch. Let us start with a formal definition:

Definition 2.1 Let $H = \{h|h : S \rightarrow T\}$ be a family of hash functions. H is locality sensitive w.r.t. sim if for all $p, q \in S$, $\Pr[h(p) = h(q)] = sim(p, q)$, where h is chosen at random. The chances of any p and q being mapped to the same number using a random h is equal to their similarity.

- probability is over random choice of h from H .
- Probability of collision = similarity between p and q

Now, assuming we have such a fabulous hash function how can we use it to solve our problem? The following algorithm is proposed

1. $P \leftarrow$ empty table of size $|S|$
2. for $i = 1, \dots, |S|$
3. read document p_i from the stream
4. $P[i] \leftarrow h(p_i)$
5. sort P and group by value
6. output groups

Table 2: Syntactic Clustering Using LSH

It is important to note that collisions will happen but search engines do not attempt to give perfect results.

As for performance, the hash values can be computed in one pass. The table P can be stored in a single file on a single machine. Sorting and grouping takes $O(|S| \log |S|)$ simple comparisons. The output consists of the groups of similar pages. This is ensured by the LSH property of our hashing function.

What are we still missing? Only a good hash function.

2.4 Building an LSH using Shingles and Min-Wise Independent Permutations

In order to build our LSH function we will first introduce the important technique of shingling.

2.4.1 Shingling

The idea of shingles is simple. A document is *tokenized*, transformed into a stream of tokens. The tokens in this case are words, html tags etc. A *w-shingle* is a sequence of w tokens. $S_w(p)$ is defined as the set of all the distinct contiguous subsequences of tokenization(p) of length w.

Example. If $p = \text{"a rose is a rose is a rose"}$. $S_w(p) = \{(a \text{ rose is a}), (rose is a rose), (is a rose is)\}$. The rest of the potential shingles are repetitive.

■

This is similar to block edit distance. *Block Edit Distance* counts the minimum number substrings, or *blocks*, needed to be moved around in order to transform one string into the other.

Finally, shingles can be used to check the similarity of two documents in pretty straightforward manner. We define *resemblance*:

$$resemblance_w(p, q) = \frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|}$$

That's nice, however, the subject of this class is sketches. We would like to create a sketch of a document, possibly using the notion of shingles. Then we would like to calculate resemblance using the sketches of the documents rather than the entire documents. That's what we do in the next section.

2.4.2 Using LSH for Resemblance

Let us start with a number of definitions. Let Σ be the set of all tokens. We define π to be a random permutation of the set of all w-length tokens sequences, Σ^w . π induces an order on a subset of this set as well $X \subseteq \Sigma^w$.

It is straightforward to see that for each $x \in X$, $\Pr(\min(\pi(X)) = x) = \frac{1}{|X|}$.

We define $h(p)$ as follows

$$h(p) = \min(\pi(S_w(p)))$$

The min function returns the first, lowest member of the permutation.

This brings us to the lemma that will help us use LSH, sketching and shingles for resemblance.

Lemma 2.2 $\Pr[\min(\pi(S_w(p))) = \min(\pi(S_w(q)))] = \frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|}$

Proof.

For the proof we will start on the left side and work our way to the right side.

$$\Pr[\min(\pi(S_w(p))) = \min(\pi(S_w(q)))] =$$

The min of $S_w(p)$ is the min of $S_w(q)$ too if and only if the minimum of both sets belongs to both sets.

$$\Pr[\min(\pi(S_w(p) \cup S_w(q))) \in S_w(p) \cap S_w(q)] =$$

Which is the probability that any of the cut between them is the minimum

$$\sum_{x \in S_w(p) \cap S_w(q)} \Pr[\min(\pi(S_w(p) \cup S_w(q))) = x] = \blacksquare$$

The probability that any given x is the minimum is just the size of the set

$$\sum_{x \in S_w(p) \cap S_w(q)} \frac{1}{|S_w(p) \cup S_w(q)|} =$$

Which brings us to the right side of the equation

$$\frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|}$$

There is still one problem with this algorithm, a problem that has come up often with these types of solutions. Storing π is very expensive. We introduce a new concept to solve this final problem, *min-wise independent permutations*.

2.4.3 Min-Wise Independent Permutations

Using the previous sections, the algorithm presents the usual problem of storing π : $O(|\Sigma|^w \log |\Sigma|^w)$ bits. The idea is to use small families of permutations.

Definition 2.3 *The family $\Pi = \{\pi | \pi \text{ is a permutation on } \Sigma^w\}$ is min-wise independent if for all subsets $X \subseteq \Sigma^w$ and for all $x \in X$*

$$\Pr(\min(\pi(X)) = x) = \frac{1}{|X|}$$

Approximate min-wise is known. Perfect min-wise is unknown.

Read more on explicit construction of small families of "approximately" min-wise independent permutations in [3].

2.5 Summary

So we can use LSH for syntactic clustering. To build the LSH we use shingles and min-wise independent permutations which together give us a good similarity measure between documents.

Known sketching schemes include:

- Resemblance [Broder et al 97]
- Hamming distance [Broder et al 98]

- Cosine Similarity [Charikar 02]
- Earth mover distance [Charikar 02]
- Edit Distance [Bar-Yossef et al 04]

There are many applications to sketching:

- clustering
- nearest neighbor schemes [2]
- data streams
- differential backup
- synchronization
- low distortion embedding
- simultaneous messages communication complexity.

3 Finding the Hamming Distance

Let us start with a formal definition:

Definition 3.1 *Let x, y be binary strings of length n . The Hamming Distance $HD(x, y)$ is defined as the number of positions in which x and y differ:*

$$HD(x, y) = |\{i | x_i \neq y_i\}|$$

We are not trying to solve the general problem returning the exact Hamming distance for any two strings. We receive two strings whose distance is guaranteed to be either at least $2k$ or at the most k . All we are required to decide is which is true. More formally our goals is:

- if $HD(x, y) \leq k$, output "accept" with probability $\geq 1 - \delta$
- if $HD(x, y) \geq 2k$, output "reject" with probability $\geq 1 - \delta$

Again, one of the above conditions is guaranteed to hold. To do this, the KOR algorithm uses $O(\log(\frac{1}{\delta}))$ size sketch. The KOR algorithm as can be expected has two stages, compression and reconstruction.

The first stage of the algorithm is the compression of the sketch. For this we choose shared randomness n i.i.d. random bits. r_1, \dots, r_n , where:

$$r_i = \begin{cases} 1 & \text{w.p. } \frac{1}{2k} \\ 0 & \text{w.p. } 1 - \frac{1}{2k} \end{cases}$$

Using the shared randomness we define a basic sketch

$$h(x) = \left(\sum_i x_i r_i \right) \bmod 2$$

The full sketch can then be defined as

$$\phi(x) = (h_1(x), \dots, h_t(x))$$

where $t = O(\log(\frac{1}{\delta}))$. h_1, \dots, h_t are generated independently like h .

Table 3: KOR: Compression stage

1. for $j = 1, \dots, t$ do
2. if $(h_j(x) = h_j(y))$ then
3. $z_j \leftarrow 1$
4. else
5. $z_j \leftarrow 0$
6. if $avg(z_1, \dots, z_t) > 11/18$ output "accept" and else output "reject"

Table 4: KOR: Reconstruction stage

The reconstruction is then done by comparing each hash function and evaluating the average as follows:

Great, we have an algorithm. The only question left to answer is why it works?

$$\begin{aligned} \Pr[h(x) = h(y)] &= \\ &= \Pr\left[\sum_{i=1}^n x_i r_i + \sum_{i=1}^n y_i r_i = 0\right] \\ &= \Pr\left[\sum_{i=1}^n (x_i + y_i) r_i = 0\right] \\ &= \Pr\left[\sum_{i, x_i \neq y_i} r_i = 0\right] \end{aligned}$$

The number of terms in the sum is $HD(x, y)$. So the probability that $h(x)$ equals $h(y)$ is the probability that the parity of $HD(x, y)$ of our random bits is 0. The question now is, given $HD(x, y)$ independent random bits each with probability $\frac{1}{2k}$ to be 1, what is the probability that their parity is 0?

We have r_{j_1}, \dots, r_{j_m} independent random bits, where $m = HD(x, y)$. For convenience we will drop the extra subscript and mark them r_1, \dots, r_m . For each such r_j we have $\Pr(r_j = 1) = \frac{1}{2k}$.

So what is $\Pr((\sum_{j=1}^{HD(x,y)} r_j) \bmod 2 = 0)$? We will use the following lemma:

Lemma 3.2 *Let r_1, \dots, r_m be m i.i.d. distributed variables*

- 0 with probability $1 - \epsilon$
- 1 with probability ϵ

then

$$\Pr[(\sum_1^m r_j) \bmod 2 = 0] = \frac{1}{2}(1 + (1 - 2\epsilon)^m)$$

Proof. We can view the distribution of each bit as a mixture of two distributions:

- Dist A (with probability $1 - 2\epsilon$): the bit 0 with probability 1
- Dist B (with probability 2ϵ): a uniformly chosen bit

It is rather easy to see that this is the same distribution. However, the thing to notice here is that if all the chosen bits follow dist A scheme, the parity is zero. If one or more of the bits follows dist B we have 0 with probability $\frac{1}{2}$. Thus:

$$\begin{aligned} \Pr[(\sum_1^m r_j) \bmod 2 = 0] &= \\ (1 - 2\epsilon)^m + (1 - (1 - 2\epsilon)^m) \cdot \frac{1}{2} &= \\ \frac{1}{2}(1 + (1 - 2\epsilon)^m) & \\ \Pr[(\sum_1^m r_j) \bmod 2 = 0] &= \frac{1}{2}(1 + (1 - 2\epsilon)^m) \end{aligned}$$

■

Applying the lemma to our problem with $\epsilon = \frac{1}{2k}$

$$\Pr[h(x) = h(y)] = \Pr[\sum_{i:x_i \neq y_i} r_i = 0] = \frac{1}{2}(1 + (1 - \frac{1}{k})^{HD(x,y)})$$

Since:

$$\lim_{x \rightarrow +\infty} (1 - \frac{1}{x})^x = \frac{1}{e}$$

- if $HD(x, y) \leq k$ then $\Pr[h(x) = h(y)] \geq \frac{1}{2}(1 + \frac{1}{e}) \approx \frac{12}{18}$
- if $HD(x, y) \geq 2k$ then $\Pr[h(x) = h(y)] \leq \frac{1}{2}(1 + (\frac{1}{e})^2) \approx \frac{10}{18}$

To phrase formal conditions for decisions we will define Z_j

$$Z_j = \begin{cases} 1 & \text{if } h_j(x) = h_j(y) \\ 0 & \text{otherwise} \end{cases}$$

So: if $HD(x, y) \leq k$ then $E[Z] \geq \frac{12}{18}$
 if $HD(x, y) \geq 2k$ then $E[Z] \leq \frac{10}{18}$
 By Chernoff $t = O(\log(\frac{1}{\delta}))$ enough guarantee
 if $HD(x, y) \leq k$, then $Z > \frac{11}{18}$ w.p $1 - \delta$
 if $HD(x, y) \geq 2k$, then $Z \leq \frac{11}{18}$ w.h.p $1 - \delta$

References

- [1] M. S. Manasse A. Z. Broder, S. C. Glassman and G. Zweig. Syntactic clustering of the web. 1997.
- [2] R. Ostrovsky E. Kushilevitz and Y. Rabani. Efficient search for approximate nearest neighbor in high dimension spaces. *Journal of Computer and System Sciences (JCSS)*, 30(2), 2000.
- [3] Piotr Indyk. A small approximately min-wise independent family of hash functions. 1999.