Assignement 1

Let's recall the considerations we took in the first homework, and in particular:

The probability that the signatures in *at least one band and all lines* of that band agree with each other and consequently becomes a *candidate pair* equals $1 - (1 - s^r)^b$

Therefore, the probability that the signatures of two sets S_1 and S_2 with Jaccard similarity J are identical in at least one band is $1 - (1 - J^r)^b$.

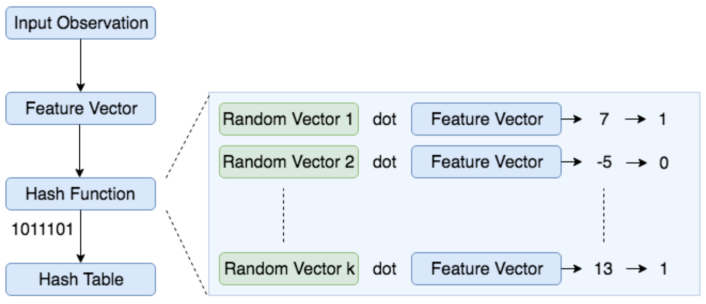
Assignement 2

Let's start from the beginning: the LSH family works on the principle that if there are two points in feature space closer to each other, they are very likely to have same hash. Thus, it is great to identify approximate nearest neighbors.

Now, let's assume that we are representing a set S of points in $\mathcal{R}^d$ using locality sensitive hashing (in particular, we are using the banding technique with bucketing). Moreover, we are considering points in a metric space (so let's assume we are interested in cosine similarity/distance) $\Rightarrow$ *Simhash / Random projections* technique.

Simhash is based on the following: we construct a table of all possible bins where each bin is made up of similar items, and it can be represented by a bitwise hash value, which is a number made up of a sequence of 1's and 0's (Ex: 110110, 111001)[1]. Therefore, two items in S with same bitwise hash values are more likely to be similar than those with different hashes. A basic algorithm to generate a bitwise hash table is:

1. Create k random vectors of length d each, where k is the size of bitwise hash value and d is the dimension of the feature vector;
2. For each random vector, compute the dot product of the random vector and the given item in S. If the result of the dot product is positive, assign the bit value as 1 else 0 (that depends on which side of the hyperplane the vector lies);
3. Concatenate all the bit values computed for k dot products;
4. Repeat the above two steps for all items in S to compute hash values for all of them;
5. Group items with same hash values together to create a LSH table;


Input Observation
Feature Vector
Hash Function
1011101
Hash Table
Random Vector 1 dot Feature Vector → 7 → 1
Random Vector 2 dot Feature Vector → -5 → 0
Random Vector k dot Feature Vector → 13 → 1

In this way we can decreases the dimensions of each item to $k << d$. However, because of the randomness, it is not likely that all similar items are grouped correctly. Thus, to overcome this limitation, a common practice is to create multiple hash tables and consider an observation a to be similar to b, if they are in same bin in at least one of the tables.

[1]We may also use -1 instead of 0

In conclusion, once we have a feature vector for all the items in S and an appropriate number of LSH Hash tables, given a new point $p \in \mathcal{R}^d$, we can find the best k candidates to be (approximately) the nearest k neighbours of p using the following algorithm:

1. Construct the feature vector for the new point p and query the LSH tables;
2. Compare the feature vector of p with the matches returned in the step above, using *cosine similarity* (therefore we are using only a subset!);
3. Return the k most similar matches;

Assignement 3

Since s is a d-dimensional random vector with entries drawn *uniformly and independently* from $\{-1, 1\}$, we have a *Rademacher* vector. Now, assuming $x \in \mathbb{R}^d$, we have

$$\mathbb{E}[F(x)] = \mathbb{E}[s^T x] = \mathbb{E}[\sum_{i=1}^{d} s_i x_i] = \sum_{i=1}^{d} \mathbb{E}[s_i x_i] = \sum_{i=1}^{d} x_i \underbrace{\mathbb{E}[s_i]}_{=0} = 0$$

And therefore we can proof:

$$\mathbb{E}[F(x)^2] = \mathbb{E}[(s^T x)^2] = \mathbb{E}\left[\left(\sum_{i=1}^{d} s_i x_i\right) \cdot \left(\sum_{j=1}^{d} s_j x_j\right)\right] = \mathbb{E}\left[\sum_{i=1}^{d}\sum_{j=1}^{d} s_i s_j x_i x_j\right] =$$

$$= \sum_{i=1}^{d} x_i^2 + \mathbb{E}\left[\sum_{i=1}^{d}\sum_{j\neq i} s_i s_j x_i x_j\right] = \sum_{i=1}^{d} x_i^2 + \sum_{i=1}^{d}\sum_{j\neq i} \mathbb{E}[s_i s_j x_i x_j] = \|x\|^2 + \sum_{i=1}^{d}\sum_{j\neq i} x_i x_j \mathbb{E}[s_i s_j] =$$

$$= \|x\|^2 + \sum_{i=1}^{d}\sum_{j\neq i} x_i x_j \underbrace{\mathbb{E}[s_i]}_{=0} \underbrace{\mathbb{E}[s_j]}_{=0} \quad = \quad \|x\|^2$$

Assignement 4

Let's assume we have a stream S of items. We can define the Reservoir sampling algorithm to maintain, at any point in time, a uniform sample of k items from stream S, as follows:

- Store the first k elements as the reservoir elements;
- For the i-th element, where $i > k$, generate a random number from 0 to i, and let it be j. If j is in range 0 to $k - 1$, replace the corresponding element inside the reservoir array;

Thus, we have a probability k/i to add the new element to the reservoir array, and each element inside the reservoir array has $1/k$ probability to be replaced. Simple proof:
For the last $n - k$ stream items, where n is the length of the stream, we have:

- Let's first consider the last item of the stream $\Rightarrow$ P[last item is in the final reservoir] = P[one of the first k indexes is picked for the last item] = k/n

- Let's now consider the second last item of the stream $\Rightarrow$ P[second last item is in the final reservoir] = (P[one of the first k indexes is picked in iteration for stream[$n-2$]]) X (P[index picked in iteration for stream[$n-1$] is not same as index picked for stream[$n-2$]]) = $[k/(n-1)] * [(n-1)/n] = k/n$
- and so on...

For the first k stream items, instead, we have:

- P[item is not picked when items stream[k], stream[$k+1$], ... stream[$n-1$] are considered] = $[k/(k+1)]$ x $[(k+1)/(k+2)]$ x $[(k+2)/(k+3)]$ x ... x $[(n-1)/n]$ = k/n

Thus, we always have the probability k/n.

Assignement 5

Let's assume a stream S of items from a discrete universe U (without losing generality $U = [n] = \{0, ..., n-1)$, where the same item can appear multiple times. We want an algorithm to keep a sample of 1 item from S such that, at any point of the stream, each of the distinct items observed so far has the same probability of being the sampled item.

Let's first suppose that we have enough space to store all the distinct elements in the stream[2]. Trivially, at any point of the stream, we can pick a sample from this list, having that each item has the same probability $\frac{1}{\textit{length list}} = \frac{1}{\textit{\# distinct elements}}$.

Now, let's suppose we do not have enough space to store all the distinct elements. This time we can recall the notion of *min-wise independent family of permutations*:

We say that $\mathcal{F} \subseteq \mathcal{S}_n$ is min-wise independent if for any set $X \subseteq [n]$ and any $x \in X$, when π is chosen at random in $\mathcal{F}$ we have

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

In other words we require that all the elements of any fixed set X have an equal chance to become the minimum element of the image of X under π, where π is an hash function.

Thus, assuming we take the correct hash function, we may use the following algorithm:

- If i is the first element:
  1. $sample = i$
  2. $h(sample) = h(i)$
- else:
  - if $h(i) < h(sample)$
    1. $sample = i$
    2. $h(sample) = h(i)$

Indeed, since at each iteration we store the couple $(sample, h(sample))$ with the lowest hash, and we are using the min-wise independent family of permutations, we can ensure there are no duplicates and there is always the same probability to be a sample for each element in the stream.

[2]We may also use hashes for faster comparison, such as in a Bloom filter