Minwise-Independent Permutations with Insertion and Deletion of Features

Rameshwar Pratap
Indian Institute of Technology (IIT) Hyderabad, Telangana, India.
rameshwar@cse.iith.ac.in

Raghav Kulkarni
Chennai Mathematical Institute (CMI) Chennai, India.
kulraghav@gmail.com

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Overview:

Broder *et. al.* [1] introduces the minHash algorithm that computes a low-dimensional sketch of high-dimensional binary data that closely approximates pairwise Jaccard similarity. minHash has been commonly used by practitioners in various big data applications.

In many real-life applications, the data is dynamic, and its feature sets evolve over time. We consider the case **when features are dynamically inserted and deleted in the dataset.**

A naive solution repeatedly recomputes minHash *w.r.t.* the updated dimension – an expensive task requiring fresh random permutations. **We initiate this study and suggest algorithms that make the** minHash **sketches adaptable to dynamic insertion and deletion of features.** We show a rigorous theoretical analysis of our algorithms. Empirically we observe a significant speed-up in the running time while simultaneously offering comparable performance *w.r.t.* baselines.

MinHash [1] - Sketching Algorithm for Jaccard Similarity:

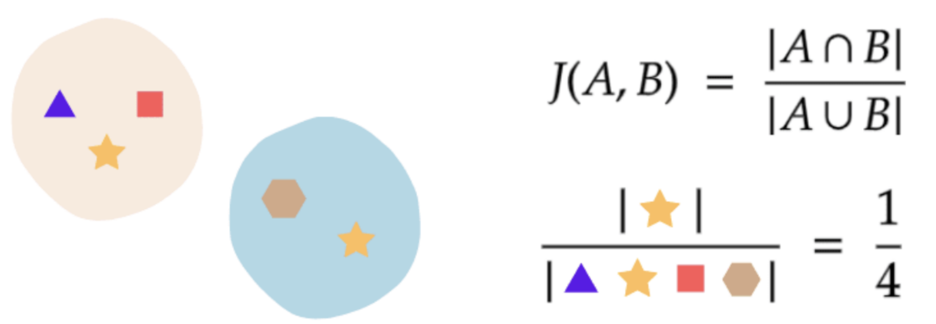
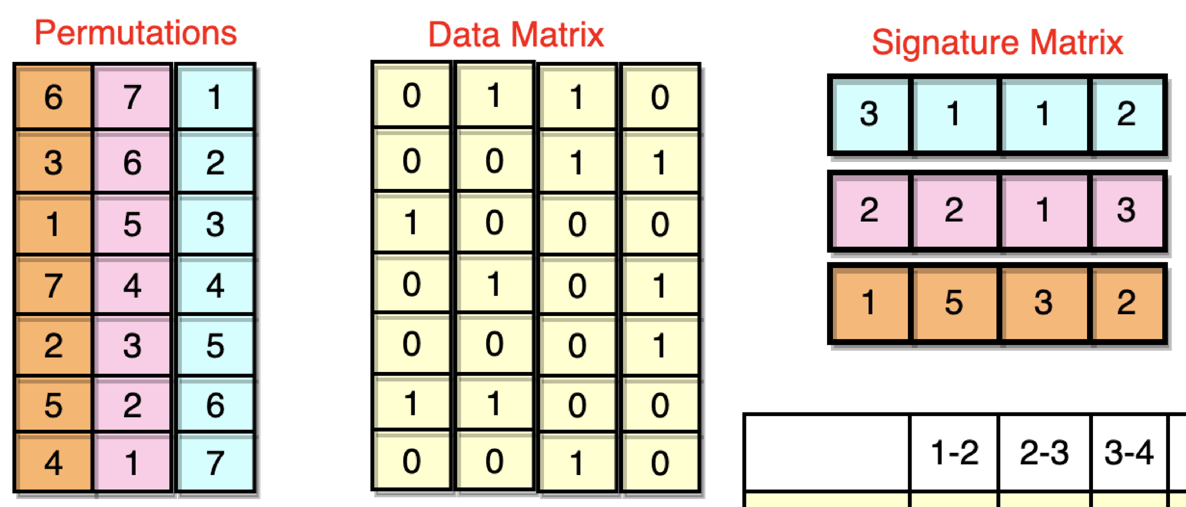


Figure 1: Jaccard Similarity.



	1-2	2-3	3-4	1-3	1-4	2-4
Jaccard	1/4	1/5	1/5	0	0	1/5
MinHash	1/3	1/3	0	0	0	0

Figure 2: MinHash [1].

Let S_d be the set of all permutations on $[d]$. We say that $F \subseteq S_d$ is **min-wise independent [1]** if for any set $U \subseteq [d]$ and any $u \in U$, when π is chosen at random in F, we have

- $\Pr[\min\{\pi(U)\} = \pi(u)] = 1/|U|.$

For a permutation $\pi \in F$ chosen at random and a set $U \subseteq [d]$ minHash [1] is defined as follows

- $\text{minHash}_\pi(U) = \arg\min_{u \in U} \pi(u).$

For two data points, $U, V \subseteq [d]$, and π is chosen at random in F, due to minHash we have

- $\Pr[\text{minHash}_\pi(U) = \text{minHash}_\pi(V)] = |U \cap V|/|U \cup V|.$

Problem Statement & Our Contributions:

Problem Statement: (i) Focus on the problem of **making** minHash **adaptable to dynamic insertions and deletions of features.** (ii) Consider the cases **when data is sparse, and features are inserted/deleted at randomly chosen positions from** 1 **to** d.

- **Contribution 1:** We present algorithms that makes minHash sketch adaptable to single/multiple feature insertions. **Our algorithm takes the current permutation and the corresponding** minHash **sketch; values and positions of the inserted features as input and outputs the** minHash **sketch corresponding to the updated dimension.**
- **Contribution 2:** We also suggest algorithms that makes minHash sketch adaptable for single/multiple feature deletions. **It takes the data points, current sketch, and permutations used to generate the same positions of the deleted features and outputs the** minHash **sketch corresponding to the updated dimension.**

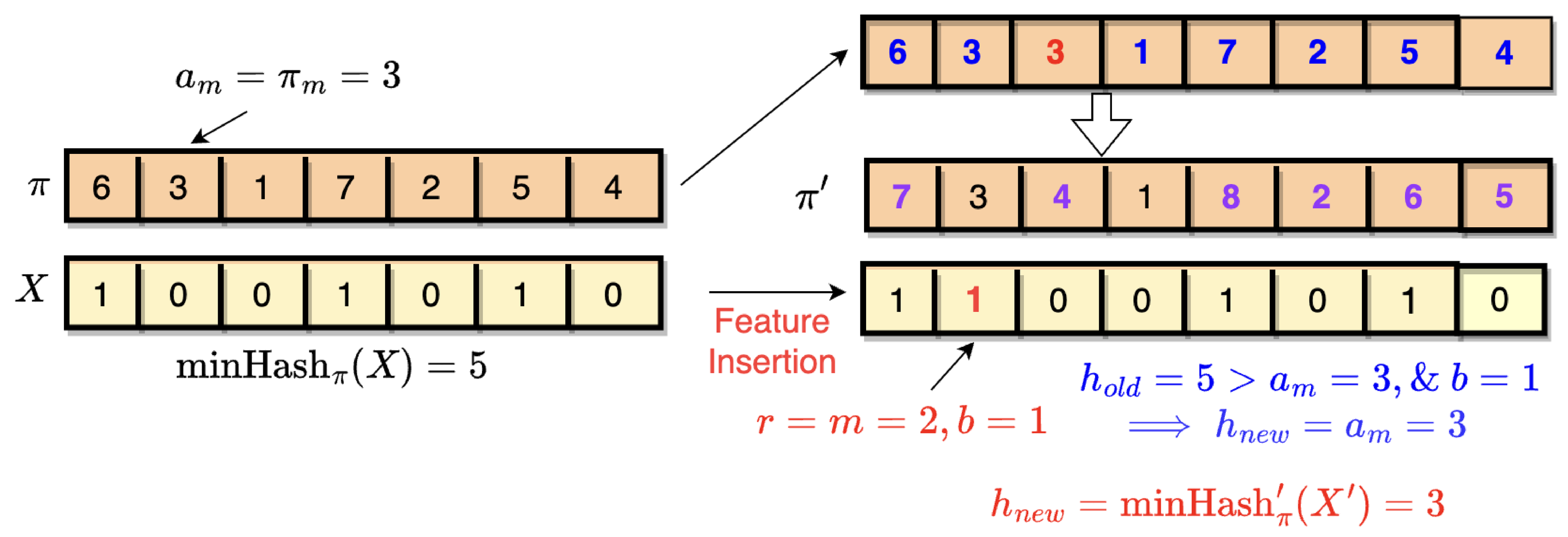
Algorithm for One Feature Insertion:

Algorithm 1: liftPerm(π, r).

```
Input: d-dim permutation π, a number r.
Output: (d+1)-dim. permutation π′.
for i ∈ {1,...,d+1} do
    if i ≤ r then
        π′(i) = π(i)
    else
        π′(i) = π(i−1)
    end
end
for i ∈ {1,...,d+1}/{r} do
    if π′(i) ≥ π′(r) then
        π′(i) = π′(i) + 1
    end
end
return π′
```

Algorithm 2: liftHash(π, m, b, h_{old}).

```
Input: h_old := minHash_π(X), π, m ∈ [d], b ∈ {0,1}.
Output: h_new := liftHash(π, m, b, h_old).
Denote a_m = π(m).
/* m is the position of the inserted feature */
if h_old < a_m then
    h_new = h_old
else
    if b = 1 then
        h_new = a_m
    end
    if b = 0 then
        h_new = h_old + 1
    end
end
return h_new
```



Theorem 1. *Let $\pi = (a_1 \ldots, a_d)$ be a minwise independent permutation, where $a_i \in [d]$, and r be a random number from $[d]$. Then for any $X \in \{0,1\}^d$ with $|X| \leq k$, the permutation $\pi' = (a'_1 \ldots, a'_{d+1})$, where $a'_i \in [d+1]$, obtained from Algorithm 1 is minwise Independent permutation, with probability at least $1 - O(k/d)$.*

Theorem 2. *Let π'_m be the $(d+1)$-dimensional permutation outputted by Algorithm 1 by setting $r = m$. Then, the sketch obtained from Algorithm 2 is the same to the sketch obtained with the permutation π'_m on X', that is, $h_{new} := \text{liftHash}(\pi, m, b, h_{old}) = \text{minHash}_{\pi'_m}(X')$.*

- Algorithm 1 liftPerm is implicit and requires for proof of correctness.
- Algorithm 2 liftHash gives the updated sketch.
- Theorem 2 shows that π' outputted by liftPerm is minwise-independent permutation, *w.h.p.*
- Theorem 3, show that the updated sketch $h_{new} = \text{minHash}_{\pi'}(X')$.
- We extend this for multiple feature insertion and also give algorithms for single/multiple feature deletion.

Experiments:

We perform our experiments on *"Bag-of-Words"* dataset [2], namely: NYTimes news articles (number of points = 500, dimension = 102660), Enron emails (number of points = 2000, dimension= 28102), and KOS blog entries (number of points = 2000, dimension = 6960).

We use two metrics: a) RMSE**: to examine accuracy, and b) running time: to measure the efficiency.**

Experimental Setting for Feature Insertion: We first create a 500 dimensional minHash sketch using 500 independent permutations. Let n features are inserted at random positions. For each position, we insert bit 1 with probability 0.1 and 0 with probability 0.9. We run the liftHash algorithm after each feature insertion, and repeat it n times. We multipleLiftHash algorithm on the initial 500 dimensional sketch with the parameter n. We compare our methods with vanilla minHash by generating a 500 dimensional sketch corresponding to the updated datasets after feature insertions.

Insights: Both of our algorithms offer **comparable performance (under** RMSE**)** with respect to *vanilla* minHash. Simultaneously, we obtain **significant speedups in running time** compared to running minHash from scratch. A similar performances are also obtained for feature deletion algorithms.

Table 1: Speedup of our algorithms *w.r.t* their vanilla minHash version.

Experiment	Method	NYTimes Max.	NYTimes Avg.	Enron Max.	Enron Avg.	KOS Max.	KOS Avg.
Feature Insertions	multipleLiftHash	54.91×	51.96×	9.61×	9.17×	24.4×	23.11×
	liftHash	91.23×	87.38×	13.96×	12.66×	35.00×	35.50×
Feature Deletions	multipleDropHash	109.5×	105.31×	18.6×	17.01×	46.02×	43.94×
	dropHash	78.34×	72.79×	15.95×	14.89×	38.24×	35.71×

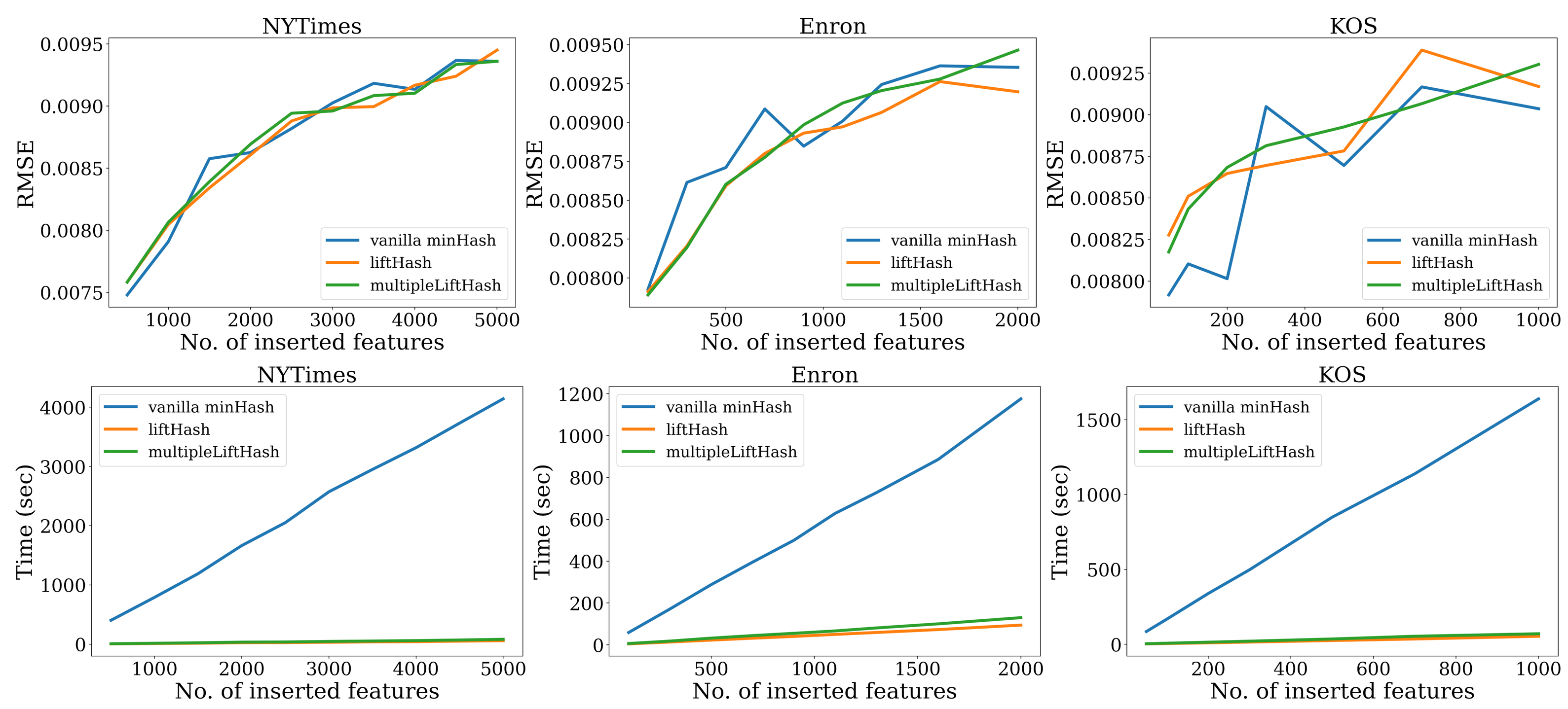


Figure 3: Comparison among liftHash, multipleLiftHash, and vanilla minHash on the task of feature insertions. Vanilla minHash corresponds to computing minHash on the updated dimension.

Open questions: Major open questions of this work are to propose algorithms (i) when features are inserted or deleted adversarially, and (ii) when the dataset is not sparse.

References

[1] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 327–336, New York, NY, USA, 1998. Association for Computing Machinery.

[2] M. Lichman. UCI machine learning repository, 2013.