Feature Hashing with Insertion and Deletion

Rameshwar Pratap,[1] Hrushikesh Sarode,[1] Suryakant Bhardwaj,[1] and Raghav Kulkarni[2]

[1] Indian Institute of Technology, Mandi
{rameshwar.pratap, sarodehrishikesh18,bhardwajsuryakant2016}@gmail.com
[2] Chennai Mathematical Institute, India
kulraghav@gmail.com

Abstract. Feature hashing algorithms [10, 8] are well-known algorithmic techniques for handling large dimensionality of the datasets. These methods reduce the dimensionality of data points while preserving the pairwise distance/similarity between the data points. However, these feature hashing algorithms are not adaptable to dynamic insertion and deletion of features. In this work, we suggest algorithms using which existing feature hashing algorithms can be made adapted to dynamic feature insertion and deletion. Our algorithms fit in the framework of both real-valued [10], and binary [8] feature hashing algorithms. We show a rigorous theoretical analysis of our algorithms, and complement it with extensive experiment on real world datasets. Our algorithms suggest better/comparable performance *w.r.t.* baselines while simultaneously offering a significant speed up in the running time. Our proposal is easy to implement and can be adopted in practice.

Keywords: Feature Hashing, Dimensionality Reduction, Inner Product.

1 Introduction

Recent technological advancements have witnessed a dramatic increase in our ability to collect data from various sources such as biology, WWW, IOT, social media, and finance, *etc.* The dimensionality of many of these datasets are in millions, and some even trillions [1, 7]. Analyzing such high dimensional datasets incurs heavy computational resources and suffers from the *"curse of dimensionality"* [9]. To tackle this challenge, several dimensionality reduction algorithms have been proposed which can significantly reduce the dimension of the data while approximating the pairwise distances between the data points [8, 5, 4, 10, 2, 3]. This essentially leads to several benefits – lower memory requirement and faster running time of the algorithm, lower storage requirement for storing the sketch of the datasets, faster inference/prediction time, smaller model size, *etc.*

These dimensionality reduction algorithms broadly fall into two categories – a) random projection, and b) feature hashing. The random projection approach is based on projecting the data matrix onto a random matrix whose entries are

sampled from a Gaussian distribution [5, 3]. The projected matrix is of low dimension, and approximates the pairwise similarity corresponding to the original data points. The feature hashing approach [10, 8] is based on randomly assigning each feature (dimension) into several bins, and the sketch value for each bin is computed by aggregating all the feature values fallen into a particular bin. Combining all such sketch values into a vector gives a low-dimensional representation of the input. For convenience, we can also think of dimensionality reduction algorithms as a map that takes the original data point as input and outputs their low-dimensional representation. We state our problem statement as follows:

Problem statement: *In this work, we focus on the techniques which can make existing dimensionality reduction algorithms adaptable to dynamic feature insertions and deletions.*

We believe that the dynamic feature insertions and deletions are quite a relevant and fundamental problem in the areas of machine learning, and data mining. We motivate this with the example below: Consider *"Bag-of-Word" (BoW)* representation of text. Here, we first create a dictionary of the important words in the corpus. Then embedding of each document is generated using the dictionary based on the frequency of the words present in the document. As the dimensionality of this representation is huge due to the large dictionary size, we seek dimensionality reduction algorithms for generating their low-dimensional representation. It is quite natural that we may have to add/remove a few words in/from the dictionary on several occasions. To handle this issue, one obvious approach is to run the dimensionality reduction algorithms from scratch on the updated dictionary, which is expensive. We, therefore, aim to develop the techniques which take the existing dimensionality reduction map as input and output a map corresponding to the updated dictionary. This technique is useful if it offers comparable accuracy along with the faster running time *w.r.t.* to running the dimensionality reduction algorithm from scratch.

We notice that dimensionality reduction with insertion and deletion can be handled easily in the case of random projection-based algorithms. Let $\mathcal{D} \in \mathbb{R}^{n \times d}$ be the data matrix, and $\mathcal{R} \in \mathbb{R}^{d \times k}$ be the random matrix whose each entry is sampled from a Gaussian distribution, where $k \ll d$. The low-dimensional representation of input $\mathcal{D}' \in \mathbb{R}^{n \times k}$ can be obtained by projecting input data onto the random matrix, that is, $\mathcal{D}' = \mathcal{D}\mathcal{R}$. In the case of feature addition/deletion, we need to add/delete the same number of Gaussian vectors in the rows of $\mathcal{R}$, and need to update the corresponding entries of $\mathcal{D}'$. However, it is unclear that an analogous technique can be applied for the other class of dimensionality reduction algorithms, in particular, the feature hashing algorithms [10, 8]. To the best of our knowledge, the feature hashing algorithm for dynamic feature addition and deletion has not been studied before. We, therefore, consider solving this problem in this work.

An important observation pertaining to feature hashing algorithms. We observe that for both feature hashing algorithms [10, 8] the mapping from features $\{1, 2, \ldots, d\}$ to bins $\{1, 2, \ldots, k\}$ should be uniformly random, that is, $\forall x \in [d], \forall y \in [k]$, probability that x is mapped to y should be equal to $1/k$. This

condition helps in ensuring that the pairwise similarity estimate in the reduced dimension closely approximates to their actual similarity in the full dimension. We discuss this for both real-valued [10] and binary feature hashing [8] as follows:
Real-valued Feature Hashing [10]: Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ be two input vectors which get compressed into $\mathbf{u_s}, \mathbf{v_s} \in \mathbb{R}^k$ using [10] (stated in Definition 1). Then, the variance of their estimate is given by the following expression (Lemma 2 of [10]):

$$\mathrm{Var}[\langle \mathbf{u_s}, \mathbf{v_s} \rangle]^2 = \sum_{i \neq j} \left({u_i}^2 {v_j}^2 + u_i v_i u_j v_j \right) \mathbb{E}_h[\delta_{h(i),h(j)}],$$

where $h(i)$ is a function which maps each feature $i \in [d]$ to one of the k bins, and $\delta_{h(i),h(j)}$ is an indicator random variable which takes value 1 if $h(i) = h(j)$, and 0 otherwise. The minimum value of the expression $\mathbb{E}_h[\delta_{h(i),h(j)}]$ is $1/k$, which is obtained when the mapping $h()$ randomly assigns each feature from $\{1, 2, \ldots, d\}$ to one of the k bins uniformly at random. This minimizes the variance of the estimate, and as a consequence, gives an accurate estimate of the similarity.
Binary Feature Hashing [8]: This algorithm takes high dimensional binary vectors as input and outputs their low dimensional binary sketch which closely estimates the similarities of the corresponding original data points. This method is based on randomly assigning each feature from $\{1, \ldots d\}$ to one of the k bins. We define an event as *"collision"* if two non-zero entries – features with value 1 – fall into the same bin. We would like to minimize the number of collisions as they lead to information loss. The number of collisions is minimized if each entry from $[d]$ is mapped to one of k bins uniformly at random. Therefore, similar to the real feature hashing, this random mapping helps in achieving an accurate estimate. We defer the reader to [8] for the details.

Thus, to minimize the error in the similarity estimate after dimensionality reduction, it is necessary to ensure that the mapping from $\{1, \ldots, d\}$ features to $\{1, \ldots, k\}$ bins remains close to uniform, when the features are dynamically added or deleted. In this work, we present an efficient way of doing this.
Our Contribution: We summarise our key contributions as follows:

- We suggest algorithms using which existing feature hashing algorithms can be adapted to dynamic feature insertion and deletion. Our algorithms fit in the framework of both real-valued [10], and binary [8] feature hashing.
- We present a theoretical analysis of our algorithms and complement it with extensive experiments on several real-world datasets. Our algorithms provide significant speedups in the running time and simultaneously offer better/comparable accuracy with respect to baselines algorithms.

2 Background

Definition 1. *(Feature Hashing for real-valued data – Definition 1 of [10]) Let* $\mathbf{u} \in \mathbb{R}^d$*. We compress* $\mathbf{u}$ *into a* k *dimensional real valued vector* $\mathbf{u_s}$ *as follows, where* $k \ll d$*. For* $i = 1$ *to* d*, we randomly assign the* i*-th position to the bucket*

number $b(i) \in \{1, \dots k\}$. Then, for $j = 1$ to k, the j-th coordinate of the compressed vector $\mathbf{u_s}$ is computed as follows: $u_{s\,j} = \sum_{i:b(i)=j} u_i x_i$, where each x_i is a random variable that takes a value between $\{-1, +1\}$ each with probability $1/2$.

We state the guarantee of algorithm stated in Definition 1 as follows:

Theorem 1. *(Estimation of Inner product – Lemma 2 and Corollary 4 of [10]) Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ be two input vectors which got compressed into $\mathbf{u_s}, \mathbf{v_s} \in \mathbb{R}^k$ using the feature hashing algorithm stated in Definition 1. Then*

$$\mathbb{E}[\langle \mathbf{u_s}, \mathbf{v_s} \rangle] = \langle \mathbf{u}, \mathbf{v} \rangle, \textit{and } \mathrm{Var}[\langle \mathbf{u_s}, \mathbf{v_s} \rangle]^2 = \frac{1}{k} \sum_{i \neq j} \left({u_i}^2 {v_j}^2 + u_i v_i u_j v_j \right).$$

Further, let us define

$$\sigma := \max \left(\mathrm{Var}[\langle \mathbf{u}, \mathbf{u} \rangle], \mathrm{Var}[\langle \mathbf{v}, \mathbf{v} \rangle], \mathrm{Var}[\langle \mathbf{u} - \mathbf{v}, \mathbf{u} - \mathbf{v} \rangle] \right);$$

$$\eta := \min \left(\frac{||\mathbf{u}||_\infty}{||\mathbf{u}||_2}, \frac{||\mathbf{v}||_\infty}{||\mathbf{v}||_2}, \frac{||\mathbf{u} - \mathbf{v}||_\infty}{||\mathbf{u} - \mathbf{v}||_2} \right), \textit{and } \Delta = ||\mathbf{u}||_2 + ||\mathbf{v}||_2 + ||\mathbf{u} - \mathbf{v}||_2.$$

Then,

$$\Pr \left[|\langle \mathbf{u_s}, \mathbf{v_s} \rangle - \langle \mathbf{u}, \mathbf{v} \rangle| > \frac{(\sqrt{2}\sigma + \epsilon)\Delta}{2} \right] < 3 \exp \left(-\frac{\sqrt{\epsilon}}{4\eta} \right).$$

Definition 2. *(Feature Hashing for binary data – Definition 4 of [8]) Let $\mathbf{u} \in \{0, 1\}^d$. We compress $\mathbf{u}$ into a k-dimensional binary vector $\mathbf{u_s}$ as follows, where $k \ll d$. Let π be a random mapping from $\{1, \dots d\}$ to $\{1, \dots k\}$. Then the vector $\mathbf{u} \in \{0, 1\}^d$ is compressed into a vector $\mathbf{u_s} \in \{0, 1\}^k$ as $u_{s\,j} = \bigvee_{i:\pi(i)=j} u_i$.*

We state the guarantee of algorithm stated in Definition 2 as follows:

Theorem 2 (Estimation of Inner product – Theorem 1 of [8]). *Suppose we want to estimate the Inner Product of d-dimensional binary vectors, whose sparsity is at most s, with probability at least $1 - \rho$. We can use the algorithm stated in Definition 2 to construct k-dimensional binary sketches, where $k = s\sqrt{(s/2)\ln(2/\rho)}$. If $\mathbf{u_s}$ and $\mathbf{v_s}$ denote the sketches of vectors $\mathbf{u}$ and $\mathbf{v}$, respectively, then $\langle \mathbf{u}, \mathbf{v} \rangle$ can be estimated with accuracy $O(\sqrt{s \ln(6/\rho)})$.*

3 Algorithms and Analysis

3.1 Feature Deletion

We consider the feature deletion problem in two cases: 1) In the first case, we assume that the number of deleted features is such that the current value of the reduced dimension is appropriate to accommodate the remaining number of features. Therefore, we use the same value of the reduced dimension. 2) In the second case, we assume that the number of deleted features are such that the reduced dimension can be made smaller to accommodate the remaining number of features. Therefore, we use a smaller value of reduced dimension in the algorithm. We consider these two cases in the following subsections and call them `Feature deletion without bin shrinking`, and `Feature deletion with bin shrinking`, respectively.

3.1.1 Feature deletion without bin shrinking: Here, we use the same value of reduced dimension to accommodate the remaining features, after feature deletion. Once a feature is deleted from the original data, one bin from the reduced dimension loses one feature. We compensate for the loss of that bin by assigning it to a randomly selected feature from the remaining set of features. If l is the number of deleted features, we repeat this process l times. We require a few notations in order to state our algorithm, which we discuss as follows:

- `Original Mapping` stores the mapping from the original dimension d to the reduced dimension k.
- `Positions of deleted features` consist of a list of features that are required to be deleted from the uncompressed data.
- `Updated Mapping` stores the updated mapping from the updated dimension d' (after deleting the features listed in `Positions of deleted features`) to the reduced dimension k.

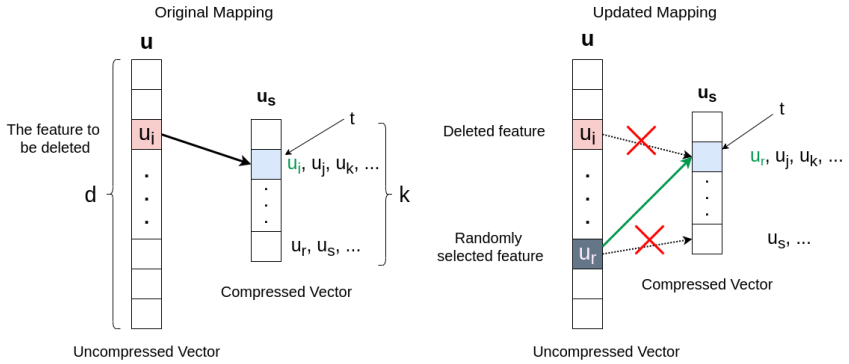

Fig. 1. Illustration of Algorithm 1 – `feature deletion without bin shrinking`. In order to compensate the deletion of i^{th} feature, we first locate the bin (say t) where it is mapped currently. We randomly select a feature r from the remaining set of features, assign it to the t^{th} bin, and delete the mapping between i^{th} and t^{th} bin.

We summarise our algorithm in Algorithm 1 and Figure 1 suggests a pictorial illustration of the same. We state a proof correctness of our algorithm in Theorem 3. We defer its proof to the appendix due to the space limit.
Notation: In the remaining paper, we use $\Pr(x \mapsto y)$ to denote the probability of mapping the feature x to the bin y, $[d]$ denotes the set $\{1, 2, \ldots, d\}$.

Theorem 3. *For every x, y such that $x \in [d]/\{i\}, y \in [k]$ in the `Updated Mapping` obtained via Algorithm 1, probability that the feature x is mapped to the bin y is $\frac{1}{k} \pm O\left(\frac{1}{d}\right)$, where i is the index of the deleted feature.*

For more than one feature deletion, we have the following Corollary which follows from Theorem 3, and the probability union bound.

Corollary 1. *For all x, y s.t. $x \in [d'], y \in [k]$ in the* `Updated Mapping` *obtained via Algorithm 1, probability that feature x is mapped to bin y is $1/k \pm O\left((d'-d)/d\right)$.*

Algorithm 1: `Feature deletion without bin shrinking.`

Input: `Original Mapping, Positions of the deleted features (list_del).`

Output: `Updated Mapping`.

for *feature i in the* `list_del` **do**
- Locate the mapped bin for i in the `Original Mapping`, denote it as t
- Randomly select a feature r from the remaining features which is not already mapped to the t^{th} bin.
- Assign the mapping between r^{th} and t^{th} bin.
- Delete the mapping between i^{th} and t^{th} bin.

return `Updated Mapping` from d' features to k bins.

3.1.2 Feature deletion with bin shrinking: Here, we use a smaller value of the reduced dimensions to accommodate the remaining features after feature deletion. We assume that an appropriate value of reduced dimension can be obtained from the corresponding feature hashing algorithms (see Theorems 1,2). Suppose the number of original features and the reduced dimension are d and k, respectively. The number of remaining features (post feature deletion) and the updated reduce dimension are d' and k', respectively, where $d' < d$ and $k' < k$. We handle this case in two steps: in the first step, we generate another random mapping from d' to k, using Algorithm 1. Then, we generate a random mapping from k bins to k' bins. Finally, the later mapping is taken into composition with the former one, which gives a mapping from d' features to k' bins. We require few notations for stating our algorithm which we discuss as follows:

- `Original Mapping` is the vector which stores the mapping from the uncompressed dimension d to the reduced dimension k.
- `Positions of Inserted features` consist of a list of features that are going to be deleted from the uncompressed data.
- `Temporary Mapping` is a vector which stores mapping from k (old reduced dimension) to k' (new reduced dimension).
- `Updated Mapping` is the vector which stores mapping from new uncompressed dimension d' (obtained after feature deletion) to the new reduced dimension k'.

We summarise our algorithm in Algorithm 2 and Figure 2 suggests a pictorial illustration of the same. We state a proof correctness of our algorithm in Theorem 4. We defer its proof to the appendix due to the space limit.

Theorem 4. *For every x, y s.t. $x \in [d'], y \in [k']$ in the* `Updated Mapping` *obtained via Algorithm 2, probability that the feature x is mapped to the bin y is $1/k' \pm O\left((d'-d)/d\right)$.*

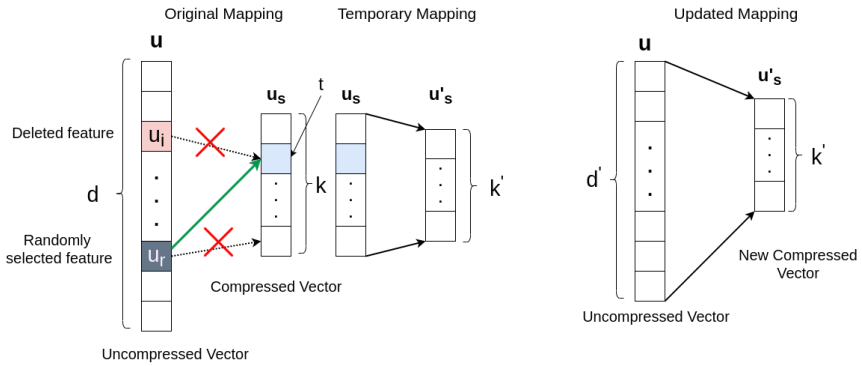

Fig. 2. Illustration of `feature deletion with bin shrinking`. First we generate a mapping from the remaining features d' (post feature deletion) with the original reduce dimension k using Algorithm 1. Then we generate a random mapping (Temporary Mapping) from k to k'. Finally, we take the composition of these two mapping to get the final updated mapping from d' to k'.

Algorithm 2: `Feature deletion with bin shrinking.`

Input: `Original Mapping, Positions of Deleted Features (list_del).`

Output: `Updated Mapping.`

for *feature i in the* `list_del` **do**
- Locate the mapped bin for i in the `Original Mapping`, denote it as t.
- Randomly select a feature r from the remaining features which is not already mapped to the t^{th} bin.
- Assign the mapping between r^{th} and t^{th} bin.
- Delete the mapping between i^{th} and t^{th} bin.

Generate a random mapping from k to k'. ▷ `Temporary Mapping.`

Take the composition of the `Original Mapping` from d' to k with the `Temporary mapping`, and output the `Updated Mapping` from d' to k'.

3.2 Feature Insertion

Whenever new features are inserted in data, the existing mapping needs to be updated such that each feature in the original dimension is allocated to one bin uniformly at random. If the number of inserted features is such that the size of the reduced dimension is not required to get increased, then this case can be trivially handled by randomly assigning the newly inserted features to one of the existing bins. However, if the size of the reduced dimension needs to get increased, then this case requires careful thought. Suppose the original and updated dimensions are d and d', respectively, and original and updated reduced dimensions are k and k', respectively, where $d < d'$ and $k < k'$. Thus, we need to generate a mapping that randomly assigns each feature from $[d']$ to one of the k' bins.

3.2.1 `Feature insertion with bin expansion:` We generate our mapping in the following two steps: 1) we first need to update the existing mapping from d features to k' bins such that each feature is randomly assigned to one of k' bins. We do this by randomly selecting $(k'-k)(d/k')$ features, assigning them randomly to the extra $(k'-k)$ bins, and deleting their earlier mapping. This gives a random mapping from d features to k' bins. 2) In the second step, we randomly assign the inserted $(d'-d)$ features to one of the k' bins.

- `Original Mapping` is the vector which stores the mapping from the original dimension d to the original reduced dimension k.
- `Positions of Inserted features` consists of a list of features that are going to be inserted in the original dimension.
- `Updated Mapping` is a vector which stores mapping from updated dimension d' to the updated reduced dimension k'.

We summarise our algorithm in Algorithm 3 and Figure 3 suggests a pictorial illustration of the same. We state a proof correctness of our algorithm in Theorem 5. We defer its proof to the appendix due to the space limit.

Algorithm 3: `Feature insertion with bin expansion.`

Input: `Original Mapping, Positions of Inserted Features (list_insert).`
Output: `Updated Mapping.`

Randomly select $(k'-k)(d/k')$ features from d. ▷ Denote it by `temp_list`.
for *feature j in* `temp_list` **do**
  Randomly assign j to one of the $(k'-k)$ bins.
for *feature i in* `list_insert` **do**
  Assign i to randomly one of the k' bins.
return `Updated Mapping` from d' features to k' bins.

Theorem 5. *For all x, y such that $x \in [d'], y \in [k']$ in the* `Updated Mapping` *obtained via Algorithm 3, probability that feature x is mapped to bin y is $1/k'$.*

4 Experimental Evaluation

Machine Configuration: CPU: Intel(R) Xeon(R) E5-2640 v4 @ 2.40GHz, Memory: 125 GB, Operating System: CentOS Linux 7.
Datasets: The experiments have been performed on the following datasets: NYTimes news articles (number of points = 300000, dimension = 102660), KOS blog entries (number of points = 3430, dimension = 6960), NIPS paper publications (number of points = 1500, dimension = 12419) [6]. These datasets are *"Bag-of-Words"* representations of text documents. We used the same dataset for both real-valued and binary experiments. In real-valued representation, for each word in the dictionary, we consider its frequency in the document. Whereas in binary representation, we consider only their presence/absence in the document. We considered a random sample of 1000 points of the NYTimes dataset, whereas for the remaining we considered their entire corpus.

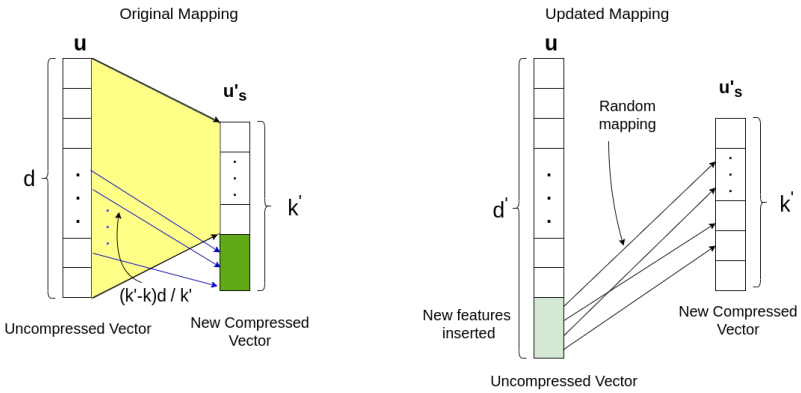

Fig. 3. Illustration of `Feature insertion with bin expansion`. We first randomly select $(k'-k)(d/k')$ features from d, then for each of the selected features we randomly assign them to one of $k'-k$ bins. Finally, we add the inserted $d'-d$ features randomly to one of the k' bins.

4.1 Experiments for feature deletion

One obvious approach for feature deletion is to randomly sample some features, delete them and the associated mapping. However, it is easy to see that after deleting the features randomly, the mapping from the remaining set of features to the reduced dimension is still close to random, with high probability. Therefore, we need a strategy to delete the features such that the mapping from the remaining set of features to the reduced dimension becomes skewed. We discuss one such strategy as follows.

Adversarial feature deletion: Here, we adversarially select some features and delete them and the associated mapping such that the mapping from the remaining set of features to the reduced dimension becomes skewed. Suppose we wish to delete p number of features. Then we sample a bin randomly from the reduced dimension, collect all the features which are mapped to that bin, and delete them. We repeat this process – of picking another random bin from the remaining bins in the reduced dimension, and deleting the mapped features – until we delete the desired number of features p.

4.1.1 Experiments for feature deletion without bin shrinking

Baselines: We compare our proposal with the following two candidates.

- `No Compensation`: In this method for each feature belonging to the list of deleted features, we delete that feature along with the associated mapping.
- `Remap`: In this method, we first delete the features belonging to the list of deleted features, and then we fully allocate the new random mapping with the remaining set of features and the reduced dimension.

Evaluation Metric: We use MSE as an evaluation criteria for comparison among baselines. We discuss it as follows: for every pair of data points, we

first calculate the ground truth inner product in the original data. Then we estimate the inner product between those pairs of points from their compressed representation. We compute the difference of these two inner products, add these values for all such pairs and calculate its mean. This we report as the MSE. We repeat it for all the baselines for various values of the number of deleted features. We follow adversarial feature deletion strategy stated above. We also note the corresponding compression time for all the baselines. We summarise our results in the Figure 4, 5 for real-valued, and binary data, respectively.

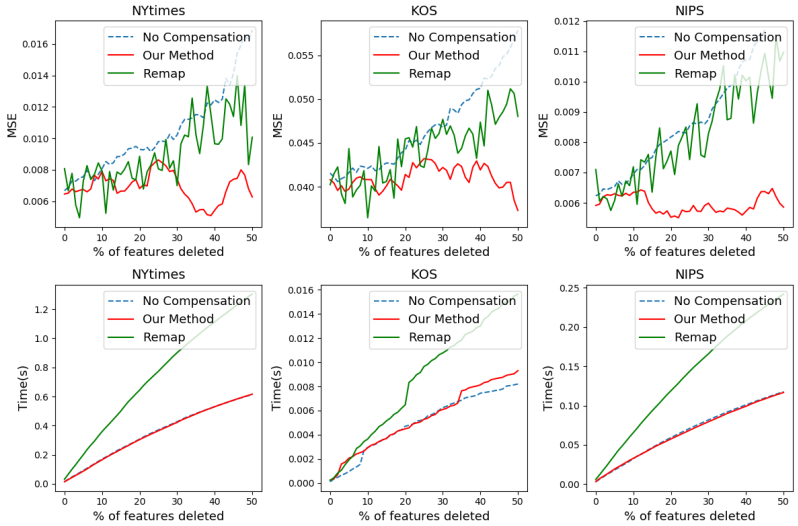

Fig. 4. Comparison on the MSE (for Inner Product) and the running time among the baselines for "`feature deletion without bin shrinking`" on NYTimes, KOS and NIPS datasets, considering them as real-valued datasets. A lower MSE is an indication of better performance.

Insights: We notice that our method offers significantly better performance with respect to both `No Compensation` and `Remap` approach. The running time of our approach is significantly faster than `Remap`, and it is comparable to the `No Compensation`. Therefore by paying almost no extra time as compared to `No Compensation`, we offer a performance which is even better than the `Remap` approach. This we notice for both real-valued as well as binary datasets.

4.1.2 Experiments for feature deletion with bin shrinking

Baselines: We compare our proposal with the following two candidates.

– `Remap`: In this method, we first delete the features belonging to the list of deleted features, and then we fully allocate the new random mapping with the remaining set of features and the updated reduced dimension.

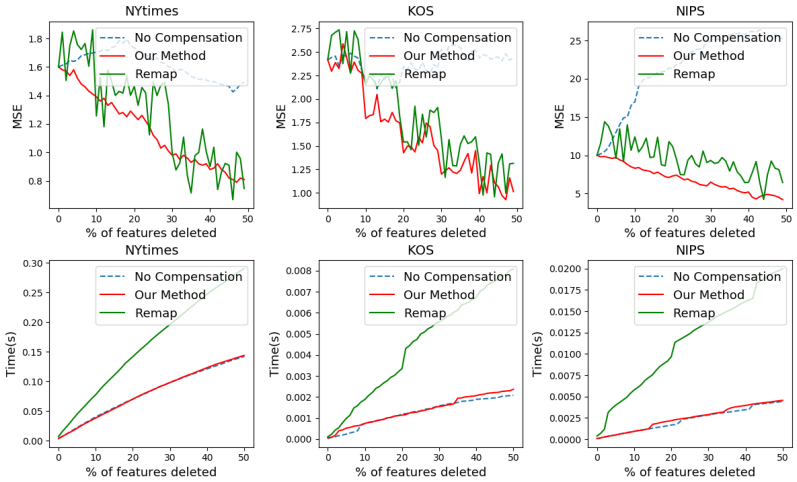

Fig. 5. Comparison on the MSE (for Inner Product) and the running time among the baselines for "`feature deletion without bin shrinking`" on NYTimes, KOS and NIPS datasets, considering them as binary datasets.

– `No Compensation`: In this method, for each feature belonging to the list of deleted features, we delete that feature along with the associated mapping.

Evaluation Metric: We evaluate the performance of our approach *w.r.t.* the baseline algorithms using MSE over Inner product as an evaluation criteria. We do it for all the baseline algorithms for various values of the number of deleted features. We also note the corresponding time required for the compression for all the baseline algorithms. In our experiments, for every thousand features deleted, we decrease the size of the reduced dimension by one. We summarise our results in the Figure 6, 7 for real-valued and binary datasets, respectively.
Insight: We notice that our method offers significantly better performance with respect to the `Remap` method. Moreover, the running time of our approach is significantly faster than `Remap`. We notice the same pattern for both the real-valued as well as the binary datasets.

4.2 Experiments for feature insertion

Baselines: We compare our proposal with the following two candidates.

– `No Compensation`: In this method, we update the features with the list of inserted features in the original dimension, and assign them randomly to one of the k bins.
– `Remap`: In this method, we update the features in the original dimension with the list of inserted features, and then we generate a random mapping from the updated dimension d' to the updated reduced dimension k'.

Evaluation Metric: We evaluate the performance of our approach *w.r.t.* the baselines using (MSE) for Inner product as an evaluation criteria. We do it for

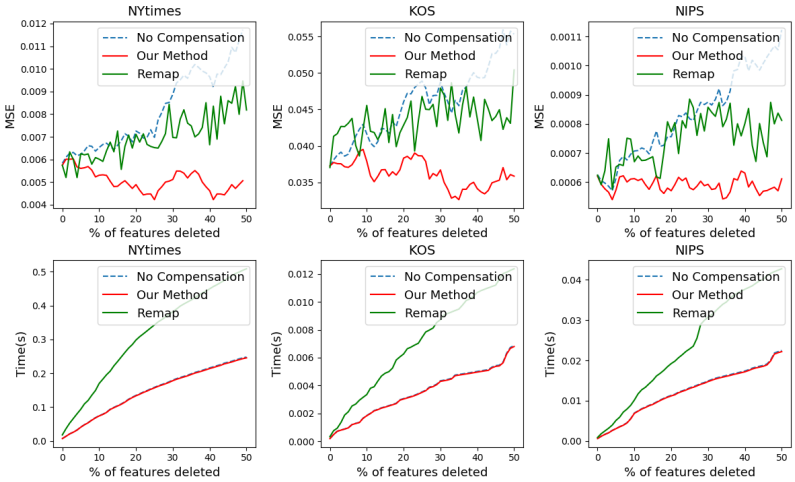

Fig. 6. Comparison on the MSE (for Inner Product) and the running time among the baselines for "`feature deletion with bin shrinking`" on NYTimes, KOS and NIPS datasets, considering them as real-valued datasets.

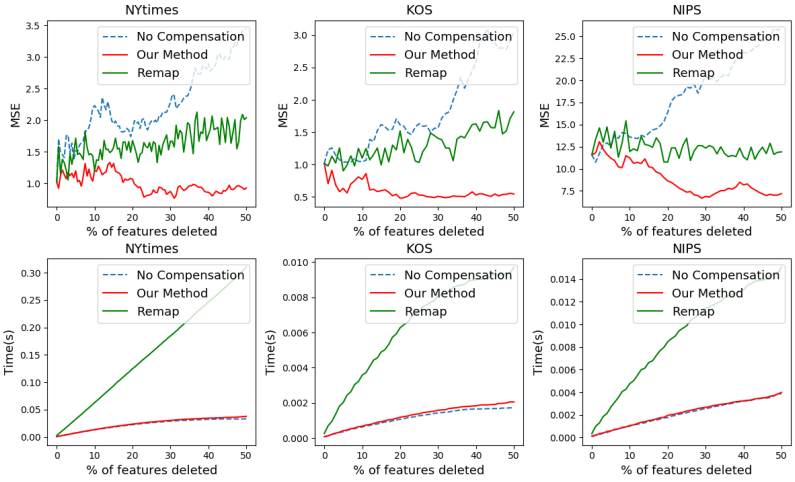

Fig. 7. Comparison on the MSE (for Inner Product) and the running time among the baselines for "`feature deletion with bin shrinking`" on NYTimes, KOS and NIPS datasets, considering them as binary datasets.

all the baselines for various values of the number of inserted features. We also note the corresponding time required for the compression for all the baselines. In our experiments, for every thousand features inserted, we increase the size of the reduced dimension by one. We summarise our results in the Figures 8, 9 for real-valued and binary data, respectively.

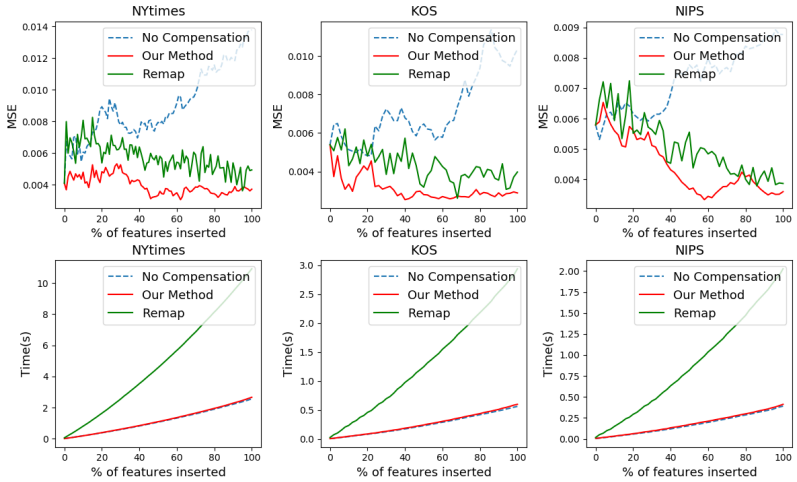

Fig. 8. Comparison on the MSE (for Inner Product) and the running time among the baselines for "`feature insertion with bin expansion`" on NYTimes, KOS and NIPS datasets, considering them as real-valued datasets.

Insights: We notice that our method offers significantly better performance with respect to both the baselines. The running time of our approach is significantly faster than `Remap` while it is being comparable to the `No Compensation` approach. We notice the same pattern for both real-valued and binary datasets.

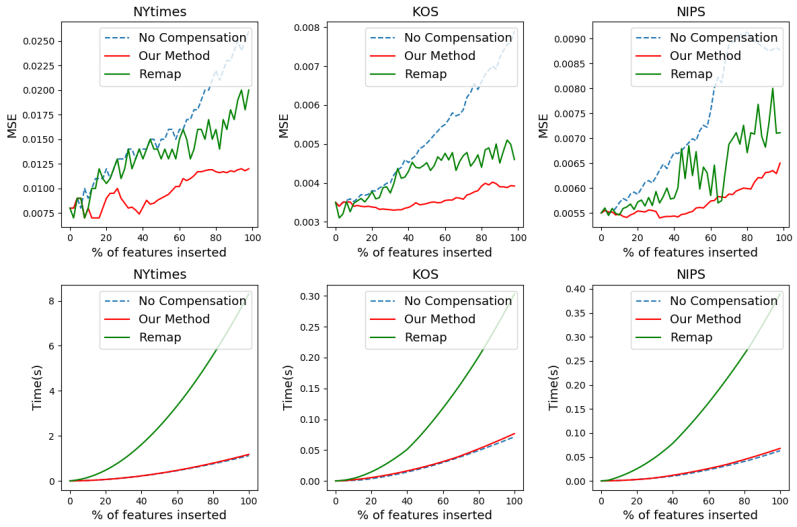

Fig. 9. Comparison on the MSE (for Inner Product) and the running time among the baselines for "`feature insertion with bin expansion`" on NYTimes, KOS and NIPS datasets, considering them as binary datasets.

5 Concluding remarks and open questions

In this work, we propose algorithms which make existing feature hashing algorithms adaptable to dynamic feature insertions and deletions. Our proposed algorithms fit in the framework of both real-valued [10] as well as binary [8] feature hashing algorithms. We show a theoretical analysis of our proposal, and complement it with extensive experiment on several real world datasets. Our proposal offers a significant speed up in dimensionality reduction time, and simultaneously offers better/comparable performance with respect to baselines. Our proposal is easy to implement in practice. Our work leaves the possibility of several open questions stated below:– a) improving the guarantee of Algorithms 1, 2 (stated in Theorems 3, 4, respectively) *via* tighter analysis, b) improving our proposal in terms of performance and dimensionality reduction time, c) suggesting a similar approach for other class of feature hashing/sketching algorithms.

References

1. Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. volume 15, pages 1111–1133, 2014.
2. 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 the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 327–336, 1998.
3. Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 380–388, 2002.
4. Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A sparse johnson: Lindenstrauss transform. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 341–350, 2010.
5. W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Conference in modern analysis and probability (New Haven, Conn., 1982), Amer. Math. Soc., Providence, R.I.*, pages 189–206, 1983.
6. M. Lichman. UCI machine learning repository, 2013.
7. M. M. A. Patwary, S. Byna, N. R. Satish, N. Sundaram, Z. Lukić, V. Roytershteyn, M. J. Anderson, Y. Yao, Prabhat, and P. Dubey. Bd-cats: big data clustering at trillion particle scale. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.
8. Rameshwar Pratap, Debajyoti Bera, and Karthik Revanuru. Efficient sketching algorithm for sparse binary data. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 508–517, 2019.
9. Michael Steinbach, Levent Ertöz, and Vipin Kumar. *The Challenges of Clustering High Dimensional Data*, pages 273–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
10. Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 1113–1120, 2009.

A Appendix

A.1 Proofs of Theorems

Theorem 3. *For every x, y such that $x \in [d]/\{i\}, y \in [k]$ in the* `Updated Mapping` *obtained via Algorithm 1, probability that the feature x is mapped to the bin y is $\frac{1}{k} \pm O\left(\frac{1}{d}\right)$, where i is the index of the deleted feature.*

Proof. Suppose after deletion of i^{th} feature we have $x \in [d]/\{i\}, y \in [k]$. Let r be the feature chosen *u.a.r.* from $[d]/\{i\}$, and t be the feature where feature i was mapped before. We now calculate the $\Pr(x \mapsto y)$ under the following two cases: **Case 1:** when $x = r$, and **Case 2:** when $x \neq r$.

$$\begin{aligned}\Pr(x \mapsto y) &= \Pr(x \mapsto y|\text{Case 1}) \times \Pr(\text{Case 1}) + \Pr(x \mapsto y|\text{Case 2}) \times \Pr(\text{Case 2}) \\ &= \mathbb{1}_{\{y=t\}} \times \frac{1}{d-1} + \frac{1}{k} \times \left(1 - \frac{1}{d-1}\right) = \frac{1}{k} \pm O\left(\frac{1}{d}\right).\end{aligned}$$

Theorem 4. *For every x, y s.t. $x \in [d'], y \in [k']$ in the* `Updated Mapping` *obtained via Algorithm 2, probability that the feature x is mapped to the bin y is $1/k' \pm O\left((d'-d)/d\right)$.*

Proof. Let $z \in [k]$ denote the bins in the original reduced dimension. For every x, y such that $x \in [d'], y \in [k']$ in the `Updated Mapping`, we have

$$\begin{aligned}\Pr(x \mapsto y) &= \sum_{z \in [k]} \Pr(x \mapsto y | x \mapsto z) \times \Pr(x \mapsto z) \\ &= \left(\frac{1}{k} \pm O\left(\frac{d'-d}{d}\right)\right) \sum_{z \in [k]} \Pr(x \mapsto y | x \mapsto z) \qquad (1) \\ &= \left(\frac{1}{k} \pm O\left(\frac{d'-d}{d}\right)\right) \sum_{z \in [k]} \Pr(z \mapsto y) \\ &= \left(\frac{1}{k} \pm O\left(\frac{d'-d}{d}\right)\right) \frac{k}{k'} = \frac{1}{k'} \pm O\left(\frac{d'-d}{d}\right).\end{aligned}$$

Equality 1 follows from Corollary 1.

Theorem 5. *For all x, y such that $x \in [d'], y \in [k']$ in the* `Updated Mapping` *obtained via Algorithm 3, probability that feature x is mapped to bin y is $1/k'$.*

Proof. Algorithm 3 works in the following two steps:
Step 1: We randomly pick $d(k'-k)/k'$ features from d features, and for each sampled feature, we randomly assign them to one of the $(k'-k)$ bins.
Step 2: We map each of the $(d'-d)$ features to one of the k' bins chosen *u.a.r.*.
We give a proof of the theorem considering the following cases:
Case 1: When $x \in [d], y \in [k]$.

$$\Pr(x \mapsto y) = \Pr(x \text{ is selected in Step 1}) \times \Pr(x \mapsto y) +$$

$$+ \Pr(x \text{ is not selected in Step 1}) \times \Pr(x \mapsto y)$$
$$= \frac{(k'-k)}{k'} \times 0 + \left(1 - \frac{(k'-k)}{k'}\right) \times \frac{1}{k} = \frac{1}{k'}.$$

Case 2: When $x \in [d], y \in [k'-k]$.

$$\Pr(x \mapsto y) = \Pr(x \text{ is selected in Step 1}) \times \Pr(x \mapsto y) +$$
$$+ \Pr(x \text{ is not selected in Step 1}) \times \Pr(x \mapsto y)$$
$$= \frac{(k'-k)}{k'} \times \frac{1}{(k'-k)} + \left(1 - \frac{(k'-k)}{k'}\right) \times 0 = \frac{1}{k'}.$$

Case 3: When $x \in [d'-d], y \in [k']$. In this case trivially $\Pr(x \mapsto y) = 1/k'$.
Case 1, 2, and 3 completes a proof of the Theorem.

A.2 Extended experimental results for binary data

Binary feature hashing algorithm stated in Definition 2 also approximates other similarity measures such as Jaccard Similarity, Cosine Similarity, Hamming Distance. We defer the readers to [8] for further details. We therefore evaluate the performance of our algorithms for Jaccard and Cosine Similarity as well. We summarise our empirical findings in Figures 10, 11 for feature deletion, and in Figure 12 for feature insertion.

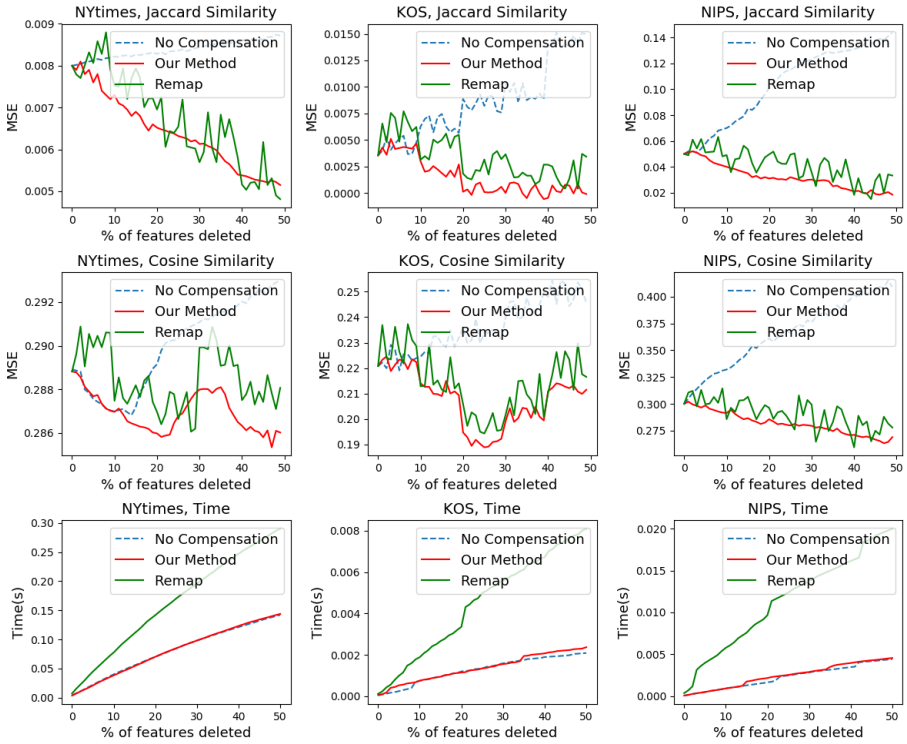

Fig. 10. Comparison of the MSE for Jaccard and Cosine Similarity and the running time among the baselines for "`feature deletion without bin shrinking`" on NYTimes, KOS and NIPS datasets, considering them as binary datasets.

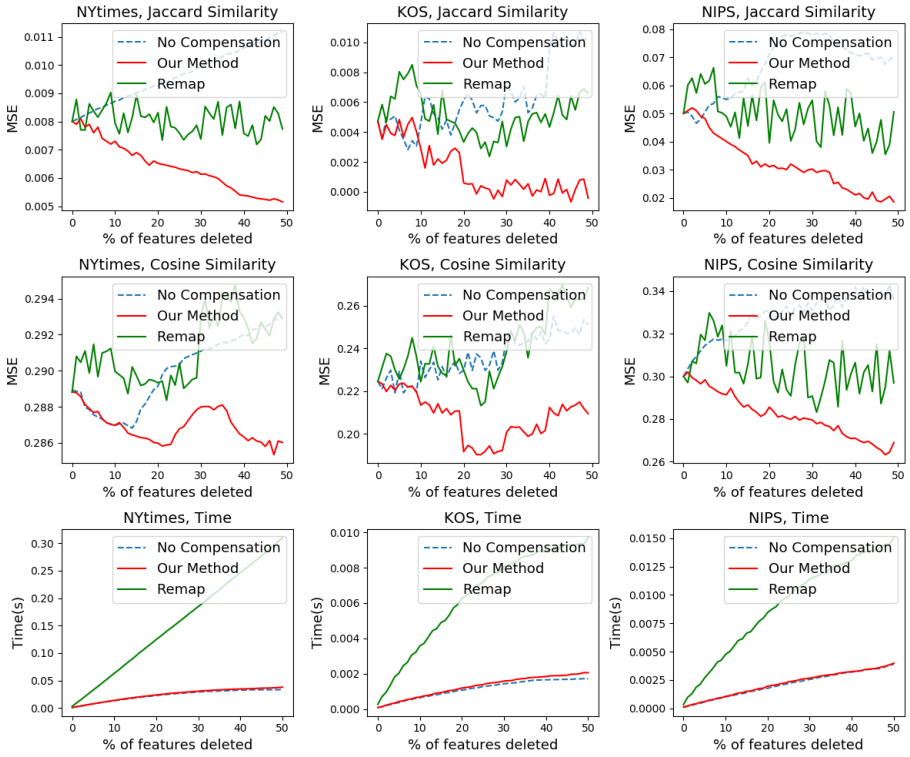

Fig. 11. Comparison of the MSE for Jaccard and Cosine Similarity and the running time among the baselines for "`feature deletion with bin shrinking`" on NYTimes, KOS and NIPS datasets, considering them as binary datasets.

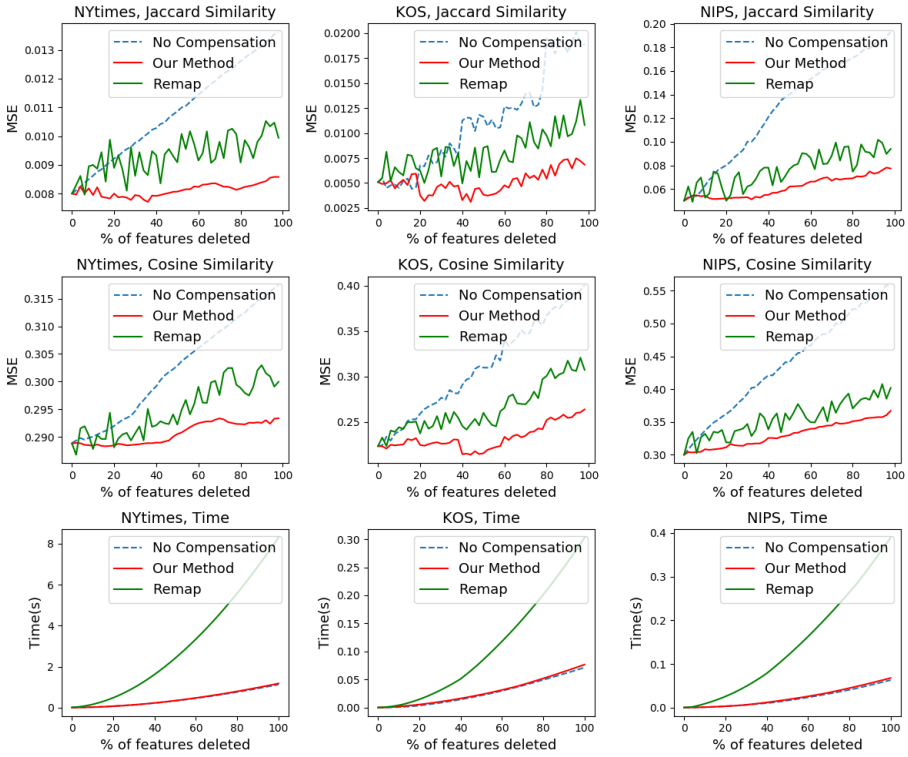

Fig. 12. Comparison of the MSE for Jaccard and Cosine Similarity and the running time among the baselines for "`feature insertion with bin expansion`" on NYTimes, KOS and NIPS datasets, considering them as binary datasets.