Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)

Accelerating Extreme Classification via Adaptive Feature Agglomeration

Ankit Jalan and **Purushottam Kar**
Department of CSE, IIT Kanpur, INDIA
aankitjalan@gmail.com, purushot@cse.iitk.ac.in

Abstract

Extreme classification seeks to assign each data point, the most relevant labels from a universe of a million or more labels. This task is faced with the dual challenge of high precision and scalability, with millisecond level prediction times being a benchmark. We propose DEFRAG, an adaptive feature agglomeration technique to accelerate extreme classification algorithms. Despite past works on feature clustering and selection, DEFRAG distinguishes itself in being able to scale to millions of features, and is especially beneficial when feature sets are sparse, which is typical of recommendation and multi-label datasets. The method comes with provable performance guarantees and performs efficient task-driven agglomeration to reduce feature dimensionalities by an order of magnitude or more. Experiments show that DEFRAG can not only reduce training and prediction times of several leading extreme classification algorithms by as much as 40%, but also be used for feature reconstruction to address the problem of missing features, as well as offer superior coverage on rare labels.

1 Introduction

The task of taking assigning data points, one or more labels from a vast universe of millions of labels is often referred to as the *extreme classification* problem. Although reminiscent of the classical multi-label learning problem, the emphasis on addressing extremely large label spaces distinguishes extreme classification. Recent advances in extreme classification have allowed problems such as ranking, recommendation and retrieval to be viewed and formulated as multi-label problems, indeed with millions of labels.

This focus on extremely large label sets has given us state-of-the-art methods for product recommendation [Jain *et al.*, 2016], search advertising [Prabhu *et al.*, 2018], and video recommendation [Weston *et al.*, 2013], as well as led to advances in our understanding of scalable optimization [Prabhu *et al.*, 2018], and distributed and parallel processing [Yen *et al.*, 2017; Babbar and Schölkopf, 2017]. Recent advances have utilized a variety of techniques – label embeddings, random forests, binary relevance, which we review in §3.

Nevertheless, extreme classification algorithms continue to face several challenges that we enumerate below.

1. **Precision**: data points often have very few labels relevant to them (e.g. of the millions of products on sale, very few products would interest any given customer). It is challenging to accurately identify these few relevant labels among the millions of irrelevant ones.
2. **Prediction**: for use in live recommendation systems, extremely rapid predictions are expected, typically within milliseconds. This often restricts the algorithmic techniques that can be used, to computationally frugal ones.
3. **Processing**: extreme classification datasets often contain millions of data points, each represented as a million-dimensional vector itself. It is challenging to offer scalable training on such large datasets.
4. **Parity**: huge label sets exhibit power-law behavior with most labels being *rare* i.e. relevant to very few data points. This causes algorithms to focus only on popular labels, neglecting the vast majority of rare ones. However, this is detrimental for recommendation outcomes.

1.1 Our Contributions

In this work, we develop the DEFRAG method and variants to address these specific challenges for a large family of algorithms. Our contributions are summarized below.

1. We propose the DEFRAG algorithm that accelerates extreme classification algorithms by performing efficient feature agglomeration on datasets with millions of features. DEFRAG performs agglomeration by constructing a balanced hierarchy which offers faster and better agglomerates than traditional clustering methods.
2. We show that DEFRAG *provably* preserves the performance of several extreme classification algorithms.
3. We exploit DEFRAG's agglomerates in a novel manner to develop the REFRAG algorithm to address the parity problem by performing efficient label re-ranking. This vastly improves the coverage of existing algorithms by accurately predicting extremely rare labels.
4. We develop the FIAT algorithm to perform scalable feature imputation which preserves prediction accuracy even when a large fraction of data features are removed.

5. We perform extensive experimentation on large-scale datasets to establish that DEFRAG not only offers significant reductions in training and prediction times, but that it does so with little or no reduction in precision.

2 Problem Formulation and Notation

The training data will be provided as n labeled data points $(\mathbf{x}^i, \mathbf{y}^i), i = 1, \ldots, n$ where $\mathbf{x}^i \in \mathbb{R}^d$ is the feature vector and $\mathbf{y}^i \in \{0,1\}^L$ is the label vector. There may be several (upto L) labels associated with each data point. Extreme classification datasets exhibit extreme *sparsity* in feature and label vectors. Let $\hat{d}$ denote the average number of non-zero features per data point and $\hat{L}$ denote the average number of active labels per data point. §6 shows that $\hat{d} \ll d$ and $\hat{L} \ll L$. We will denote the feature matrix using $X = \left[\mathbf{x}^1, \ldots, \mathbf{x}^n\right] \in \mathbb{R}^{d \times n}$ and the label matrix using $Y = \left[\mathbf{y}^1, \ldots, \mathbf{y}^n\right] \in \{0,1\}^{L \times n}$.

Let $\mathcal{F} = \{F_1, \ldots, F_K\}$ denote any K-partition of the feature set $[d]$ i.e. $F_i \cap F_j = \emptyset$ if $i \neq j$ and $\bigcup_{k=1}^{K} F_k = [d]$. Let $d_k := |F_k|$ denote the size of the k^{th} cluster. For any vector $\mathbf{z} \in \mathbb{R}^d$, let $\mathbf{z}_j$ denote its j^{th} coordinate. For any set $F_k \in \mathcal{F}$, let $\mathbf{z}_{F_k} := [\mathbf{z}_j]_{j \in F_k}^{\top} \in \mathbb{R}^{d_k}$ denote the (shorter) vector containing only coordinates from the set F_k.

Feature Agglomeration. This involves creating clusters of features and then summing up features within a cluster. If $\mathcal{F}$ is a partition of the features $[d]$, then for every cluster $F_k \in \mathcal{F}$, we create a a single "super"-feature. Thus, given a vector $\mathbf{z} \in \mathbb{R}^d$, we can create an agglomerated vector $\tilde{\mathbf{z}}^{[\mathcal{F}]} \in \mathbb{R}^K$ (abbreviated to just $\tilde{\mathbf{z}}$ for sake of notational simplicity) with just K features using the clustering $\mathcal{F}$. The k^{th} dimension of $\tilde{\mathbf{z}}$ will be $\tilde{\mathbf{z}}_k = \sum_{j \in F_k} \mathbf{z}_i$ for $k = 1, \ldots, K$. The DEFRAG algorithm will automatically learn relevant feature clusters $\mathcal{F}$.

3 Related Works

We discuss relevant works in extreme classification and scalable clustering and feature agglomeration techniques here.

Binary Relevance. Also known as *one-vs-all* methods, these techniques, for example DiSMEC [Babbar and Schölkopf, 2017], PPDSparse [Yen *et al.*, 2017], and ProXML [Babbar and Schölkopf, 2019], learn L binary classifiers: for each label $l \in [L]$, a binary classifier is learnt to distinguish data points that contain label l from those that do not. Binary relevance methods offer some of the highest precision values among extreme classification algorithms [Prabhu *et al.*, 2018]. However, despite advances in parallel training and active set methods, they still incur training and prediction times that are prohibitive for most applications.

Label/Feature Embedding. These techniques project feature and/or label vectors onto a low dimensional space i.e. $\mathbf{x}^i \mapsto \hat{\mathbf{x}}^i, \mathbf{y}^i \mapsto \hat{\mathbf{y}}^i$ where $\hat{\mathbf{x}}^i, \hat{\mathbf{y}}^i \in \mathbb{R}^p$, $p \ll \min\{d, L\}$ using random or learnt projections. Prediction and training is performed in the low dimensional space $\mathbb{R}^p$ for speed. These methods SLEEC [Bhatia *et al.*, 2015], AnnexML [Tagami, 2017] and LEML [Yu *et al.*, 2014] offer strong theoretical guarantees, but are usually forced to choose a moderate value of p to maintain scalability. This often results in low precision values and causes these methods to struggle on rare labels.

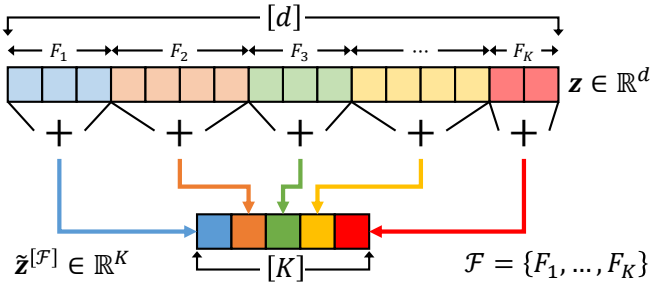

Figure 1: An illustration of the feature agglomeration process.

Data Partitioning. These techniques learn a decision tree over the data points which are hierarchically clustered into several leaves, with the hope that the similar data points, i.e. those with similar label vectors, end up in the same leaf. A simple classifier (usually constant) performs label prediction at a leaf. These methods PfastreXML [Jain *et al.*, 2016], FastXML [Prabhu and Varma, 2014] and CRAFTML [Siblini *et al.*, 2018] offer fast prediction times due to prediction being logarithmic in number of leaves in a balanced tree.

Label Partitioning. These methods instead learn to organize labels into (overlapping) clusters, using hierarchical partitioning techniques. Prediction is done by taking a data point to one or more of the leaves of the tree and using a simple method such as 1-vs-all among labels present at that leaf. These methods PLT [Jasinska *et al.*, 2016], Parabel [Prabhu *et al.*, 2018], LPSR [Weston *et al.*, 2013] offer fast prediction times due to the tree structure, as well as high precision by using a 1-vs-all classifier at the leaves, as Parabel does.

Large Scale (Feature) Clustering. Clustering, as well as feature clustering and agglomeration, are well-studied topics. Past works include techniques for scalable balanced k-means using alternating minimization techniques SCBC [Banerjee and Ghosh, 2006] and BCLS [Liu *et al.*, 2017], scalable spectral clustering using landmarking LSC [Chen and Cai, 2011], and scalable information-theoretic clustering ITDC [Dhillon *et al.*, 2003]. We do compare DEFRAG against all these algorithms. These algorithms were chosen since they were able to scale to at least the smallest datasets in our experiments.

4 DEFRAG: aDaptive Extreme FeatuRe AGglomeration

We now describe the DEFRAG method, discuss its key advantages and then develop the REFRAG method for rare label prediction and the FIAT method for feature imputation. Recall from §2 that given a K-partition $\mathcal{F}$ of the features $[d]$, feature agglomeration takes each cluster $F_k \in \mathcal{F}$ and agglomerates all features $j \in F_k$ by summing up their feature values.

Given a dataset with d-dimensional features, DEFRAG first clusters these features into balanced clusters, with each cluster containing, say no more than d_0 features. Suppose this process results in K clusters. DEFRAG then uses feature agglomeration (see §2 and Figure 1) to obtain K-dimensional features for all data points in the dataset which are then used for training and testing.

Algorithm 1 DEFRAG: Make-Tree

Input: Feature set $S \subseteq [d]$, representative vectors $\mathbf{z}^i \in \mathbb{R}^p$ for each feature $i \in S$, maximum leaf size d_0
Output: A tree with each leaf having upto d_0 features

```
if |S| ≤ d_0 then
    n ← Make-Leaf(S)                    // No need to split
else
    n ← Make-Internal-Node()
    {S_+, S_-} ← Balanced-Split(S, {z^i, i ∈ S})
                    // Balanced k-means or nDCG split
    n_+ ← Make-Tree(S_+, {z^i, i ∈ S_+}, d_0)
    n_- ← Make-Tree(S_+, {z^i, i ∈ S_-}, d_0)
    n.Left-Child ← n_+
    n.Right-Child ← n_-
end if
return Root node of this tree n
```

DEFRAG first creates a representative vector for each feature $j \in [d]$ and then performs *hierarchical clustering* on them (see Algorithm 1) to obtain feature clusters. At each internal node of the hierarchy, features at that node are split into two children nodes of equal sizes by solving either a balanced spherical 2-means problem or else by minimizing a *ranking loss* like nDCG [Prabhu and Varma, 2014] which we call DEFRAG-N (please see the full version for details). This process is continued till we are left with less than d_0 features at a node, in which case the node is made a leaf. We now discuss two methods to construct these representative vectors.

DEFRAG-X. This variant clusters together *co-occurrent* features e.g. $j, j' \in [d]$ where data points with a non-zero (or high) value for feature j also have a non-zero (or high) value for feature j'. DEFRAG-X represents each feature $j \in [d]$ as an n-dimensional vector $\mathbf{p}^j = [\mathbf{x}_j^1, \ldots, \mathbf{x}_j^n]^\top \in \mathbb{R}^n$, i.e. as a list of values that feature takes in all data points.

DEFRAG-XY. This variant clusters together *co-predictive* features e.g. $j, j' \in [d]$ where data points where feature j is non-zero have similar labels as data points where feature j' is non-zero. To do so, DEFRAG-XY represents each feature $j \in [d]$ as an L-dimensional vector $\mathbf{q}^j = \sum_{i=1}^n \mathbf{x}_j^i \mathbf{y}^i \in \mathbb{R}^L$, essentially as a weighted aggregate of the label vectors of data points where the feature j is non-zero.

4.1 Advantages of DEFRAG

DEFRAG suits high-dim., sparse data better than dimensionality reduction techniques like PCA or random projection.

1. Applying feature agglomeration to a vector involves summing up the coordinates of that vector and is much cheaper than performing PCA or a random projection.
2. Linear projections densify vectors and so methods like LEML and SLEEC are compelled to use a small embedding dimension ($\approx$ 500) for sake of scalability which leads to information loss. Feature agglomeration, however, does not densify vectors: if a vector $\mathbf{x} \in \mathbb{R}^d$ has only s non-zero coordinates, then for any feature K-clustering $\mathcal{F}$, the vector $\tilde{\mathbf{x}}^{[\mathcal{F}]} \in \mathbb{R}^K$ cannot have more than s non-zero coordinates. This allows DEFRAG to operate with relatively large values of K (e.g. $K = d/8$ is default in our experiments as we set $d_0 = 8$) without worrying about memory or time issues. DEFRAG thus offers *mildly* agglomerated vectors which preserve much of the information of the original vector, yet offer speedups due to the reduced dimensionality.
3. Feature agglomeration has an implicit *weight-tying* effect: a linear model learnt over the agglomerated features effectively places the same weight over all features belonging to a cluster $F_k \in \mathcal{F}$. This reduces the capacity of the model and can improve generalization error.

§5 shows that feature agglomeration using a feature clustering $\mathcal{F}$ with small clustering error *provably* preserves the performance of *all* linear models. Specifically, for every model $\mathbf{w} \in \mathbb{R}^d$ over the original vectors, there must exist a model $\tilde{\mathbf{w}} \in \mathbb{R}^K$ over the agglomerated vectors such that for *any vector* $\mathbf{x} \in \mathbb{R}^d$, we have $\mathbf{w}^\top \mathbf{x} \approx \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}^{[\mathcal{F}]}$. Thus, similar 1-vs-all models, trees and label partitions can be learnt over $\tilde{\mathbf{x}}^{[\mathcal{F}]}$. We note that *all* algorithms discussed in §3 ultimately use just linear models as components (e.g. binary relevance methods learn L linear classifiers – one per label, embedding methods learn linear projections, data and label partitioning methods learn linear models to split internal tree nodes, etc.).

Task adaptivity. DEFRAG-XY takes into account labels in its feature representation which makes it task-adaptive as compared to dimensionality reduction or clustering methods like k-means, PCA which do not take consider labels. Indeed, we will see that on many datasets, DEFRAG-XY outperforms DEFRAG-X which also does not take labels into account.

Novelty, Speed and Scalability. Hierarchical *feature* agglomeration is novel in the context of extreme classification although hierarchical *data* partitioning (Parabel) and hierarchical *label* partitioning (PfastreXML) have been successfully attempted before. The representative vectors created by DEFRAG are themselves sparse and hierarchical feature agglomeration offers speedy feature clustering. DEFRAG's overhead on the training process is thus, very small.

Time Complexity. Let $\text{nnz}(X) = n \cdot \hat{d}$ be the number of non-zero elements in the feature matrix X. Computing the feature representations $\mathbf{p}^j, j \in [d]$ takes $\mathcal{O}(\text{nnz}(X))$ time. The total time taken to perform balanced spherical 2-means clustering for all nodes at a certain level in the tree is $\mathcal{O}(\text{nnz}(X))$ as well. Since DEFRAG performs balanced splits, there can be at most $\mathcal{O}(\log d)$ levels in the tree, thus giving us a total time complexity of $\mathcal{O}(\text{nnz}(X) \log d)$.

4.2 FIAT: Feature Imputation via AgglomeraTion

Co-occurrence based feature imputation is popularly used to overcome the problem of missing features. However, this becomes prohibitive for extreme classification settings since the standard co-occurrence matrix $C = XX^\top$ is too dense to store and operate. We exploit the feature clusters offered by DEFRAG to create a scalable co-occurrence based feature imputation algorithm FIAT. For any feature cluster $F_k \in \mathcal{F}$ let $X_{F_k} \in \mathbb{R}^{d \times n}$ denote the matrix with only those rows that belong to the cluster F_k. Given this, we compute a pseudo co-occurrence matrix $C^{\mathcal{F}} = \sum_{k=1}^K X_{F_k} X_{F_k}^\top \in \mathbb{R}^{d \times d}$.

Note that $C^{\mathcal{F}}$ has a block-diagonal structure and has only upto $\frac{d^2}{K}$ non-zero entries where K is the number of clusters. Thus, it is much cheaper to store and operate. Given a feature vector $\mathbf{x} \in \mathbb{R}^d$ that we suspect has missing features, we perform feature imputation on it by simply calculating $C^{\mathcal{F}}\mathbf{x}$.

4.3 REFRAG: REranking via FeatuRe AGglomeration

Algorithms frequently neglect rare labels (that occur in very few data points) in favor of popular ones [Wei and Li, 2018]. To address this, we propose an efficient re-ranking solution based on the pseudo co-occurrence matrix $C^{\mathcal{F}}$ described earlier. First compute the matrix product $C^{\mathcal{F}}XY^\top \in \mathbb{R}^{d\times L}$. The l^{th} column of this matrix $l \in [L]$ can be interpreted as giving us a *prototype data point* $\boldsymbol{\xi}^l \in \mathbb{R}^d$ for the label l.

These prototypes can be used to get the affinity score of a test data point $\mathbf{x}^t$ to a label $l \in [L]$ as $e^{-\frac{\gamma}{2}\cdot\left\|\mathbf{x}^t-\boldsymbol{\xi}^l\right\|_2^2}$. Once a base classification algorithm such as Parabel or DiSMEC has given scores for the test point $\mathbf{x}^t$ with respect to various labels, instead of predicting the labels with the highest scores right-away, we combine the classifier scores with these affinity scores and then make the predictions. We note that a similar approach was proposed by [Jain *et al.*, 2016] who did achieve enhanced performance on rare labels.

However, whereas their method requires an optimization problem to be solved to obtain the prototypes, we have a closed form expression for prototypes in our model given the efficiently computable pseudo co-occurrence matrix $C^{\mathcal{F}}$.

Due to lack of space, further algorithmic details as well as proofs of theorems in §5 are presented in the full version of the paper available at the URL given below.

Full Version Link: http://arxiv.org/abs/1905.11769

5 Performance Guarantees

In this section we establish that DEFRAG provably preserves the performance of extreme classification algorithms. For any vector $\mathbf{v} \in \mathbb{R}^p$ we will utilize the orthogonal decomposition $\mathbf{v} = \mathbf{v}^{\|} + \mathbf{v}^{\perp}$ where $\mathbf{v}^{\|}$ is the component of $\mathbf{v}$ along the all-ones vector $\mathbf{1}_p = (1,\ldots,1) \in \mathbb{R}^p$ and $\mathbf{v}^{\perp}$ is the component orthogonal to it i.e. $\mathbf{1}_p^\top \mathbf{v}^{\perp} = 0$. At the core of our results is the following lemma. Given a real valued matrix $Z \in \mathbb{R}^{d\times p}$ for some $p > 0$ and a K-partition $\mathcal{F}$ of the feature set $[d]$, we will let $Z_k \in \mathbb{R}^{d_k\times p}$ denote the matrix formed out of the rows of the matrix that correspond to the partition F_k.

Lemma 1. *Given any matrix $Z \in \mathbb{R}^{d\times p}$ and any K-partition $\mathcal{F} = \{F_1,\ldots,F_K\}$ of $[d]$, suppose there exist vectors $\boldsymbol{\mu}^1,\ldots,\boldsymbol{\mu}^K \in \mathbb{R}^p$ such that $Z_k = \mathbf{1}_{d_k}(\boldsymbol{\mu}^k)^\top + \Delta_k$ where $\mathbf{1}_{d_k} := (1,\ldots,1)^\top \in \mathbb{R}^{d_k}$, then for* every *$\mathbf{w} \in \mathbb{R}^d$ and every $k \in [K]$, there must exist a real value $c_{\mathbf{w},k} \in \mathbb{R}$ such that*

$$(\mathbf{w}_{F_k} - c_{\mathbf{w},k}\cdot\mathbf{1}_{d_k})^\top Z_k Z_k^\top (\mathbf{w}_{F_k} - c_{\mathbf{w},k}\cdot\mathbf{1}_{d_k}) \le \left\|\Delta_k^\top \mathbf{w}_{F_k}^{\perp}\right\|_2^2.$$

Lemma 1 will be used to show below that, if a group of features is "well-clustered", then it is possible to tie together weights corresponding to those features in *every* linear model.

Theorem 2. *Upon executing DEFRAG-X with a feature matrix $X = [\mathbf{x}^1,\ldots,\mathbf{x}^n]$ and label matrix $Y = [\mathbf{y}^1,\ldots,\mathbf{y}^n]$, suppose we obtain a feature K-partition $\mathcal{F} = [F_1,\ldots,F_K]$ with err_k denoting the Euclidean clustering error within the k^{th} cluster, then for* any *loss function $\ell(\cdot)$ that is L-Lipschitz and for* every *linear model $\mathbf{w} \in \mathbb{R}^d$, there must exist a model $\tilde{\mathbf{w}} \in \mathbb{R}^K$ such that for all subsets of data points $S \subseteq [n]$,*

$$\sqrt{\sum_{i\in S}\left(\ell(\mathbf{w}^\top\mathbf{x}^i;\mathbf{y}^i) - \ell(\tilde{\mathbf{w}}^\top\tilde{\mathbf{x}}^i;\mathbf{y}^i)\right)^2} \le L\cdot\sum_{k=1}^{K}\left\|\mathbf{w}_{F_k}^{\perp}\right\|_2 \cdot err_k.$$

To simplify this result, let $w_0 = \max_{k\in[K]}\left\|\mathbf{w}_{F_k}^{\perp}\right\|_2^2 \le \max_{k\in[K]}\left\|\mathbf{w}_{F_k}\right\|_2^2$ (since cluster sizes d_k are typically small, w_0 is small too) and use the fact that $\text{err}_k \ge 0$ to get

$$\sqrt{\sum_{i\in S}\left(\ell(\mathbf{w}^\top\mathbf{x}^i;\mathbf{y}^i) - \ell(\tilde{\mathbf{w}}^\top\tilde{\mathbf{x}}^i;\mathbf{y}^i)\right)^2} \le L\cdot w_0\cdot\sum_{k=1}^{K}\text{err}_k.$$

In the full version, we show that DEFRAG-XY preserves the performance of label clustering methods such as Parabel. A few points are notable about the above results.

Uniform Model Preservation. Theorem 2 guarantees that if the clustering error is small (and DEFRAG does minimize clustering error), then for *every* possible linear model $\mathbf{w} \in \mathbb{R}^d$ over the original features $\mathbf{x}^i$, we can learn a model $\tilde{\mathbf{w}} \in \mathbb{R}^d$ over the agglomerated features $\tilde{\mathbf{x}}^i$ such that both models behave similarly with respect to any Lipschitz loss function. Note that Theorem 2 holds *simultaneously* for all linear models $\mathbf{w}$, and is thus, agnostic to the algorithm used to learn $\mathbf{w}$.

Classifier Preservation. Most leading algorithms (Parabel, DiSMEC, PfastreXML, PPDSparse, SLEEC) construct classifiers by learning several linear models using hinge loss, exponential loss etc. which are Lipschitz. By preserving the performance of all such individual linear models, DEFRAG preserves the overall performance of these algorithms too. Note that Theorem 2 holds uniformly over *all* subsets $S \subseteq [n]$ of data points, which is useful since these algorithms often learn several linear models on various subsets of the data.

Graceful Adaptivity. Suppose for a model $\mathbf{w}$, the weights within a cluster F_k are similar i.e. $\mathbf{w}_{F_k} \approx w\cdot\mathbf{1}_{d_k}, w \in \mathbb{R}$. Then this implies $\mathbf{w}_{F_k}^{\perp} \approx \mathbf{0}$ and the contribution of this cluster to the total error will be very small. This indicates that if some of the original weights are anyway tied together, DEFRAG automatically offers extremely accurate reconstructions.

6 Experimental Results

We studied the effects of using DEFRAG variants with several extreme classification algorithms, as well as compared DEFRAG with other clustering algorithms. Our implementation of DEFRAG is available at the URL given below.

Code Link: https://github.com/purushottamkar/defrag/

Datasets and benchmark implementations. All datasets, train-test splits, and implementations of extreme classification algorithms were sourced from the Extreme Classification Repository [Bhatia *et al.*, 2019]. Implementations of clustering algorithms were sourced from the respective authors whenever possible. For SCBC, LSC and ITDC, public implementations were not available and scalable implementations were created in the Python language.

Method	LMI	Bal.	Ent.	Time (min)	P1(%)	P3(%)	P5(%)
EURLex-4K							
ITDC	0.47	Inf	0.87	0.7	73.85	59.63	49.19
SCBC	0.39	321	0.75	0.55	71.96	59.50	49.41
LSC	0.60	130	0.93	5.7	71.44	58.68	48.71
BCLS	0.50	Inf	0.91	2.5	74.14	60.96	50.53
DEFRAG-X	**0.37**	**1.11**	**0.99**	**0.03**	**78.97**	**65.68**	**54.46**
Wiki10-31K							
ITDC	0.52	Inf	0.88	23	82.03	69.72	60.07
DEFRAG-G	0.46	**1.08**	**0.99**	0.48*	82.72	69.62	60.23
DEFRAG-X	**0.36**	**1.08**	**0.99**	**0.45**	**84.99**	**73.47**	**63.91**

Table 1: A comparison of DEFRAG with other clustering algorithms on clustering quality (definitions of clustering metrics in full version), as measured by loss of mutual information (LMI), balance factor, normalized entropy, clustering time, and classification performance when the Parabel algorithm was executed upon agglomerated features given by the clustering algorithms. BCLS, LSC and SCBC could not scale to Wiki10. A balance factor of Inf indicates the presence of an empty cluster. DEFRAG not only outperforms other clustering algorithms in terms of clustering quality and classification accuracy, but offers clustering times that can be an order of magnitude smaller. DEFRAG-G denotes the DEFRAG-X algorithm executed on word features learnt by the GloVe algorithm [Pennington *et al.*, 2014]. DEFRAG could not be outperformed by carefully crafted word vector representations like GloVe either.
*The clustering time for DEFRAG-G does not include time taken to extract (dense) GloVe word features from raw text.

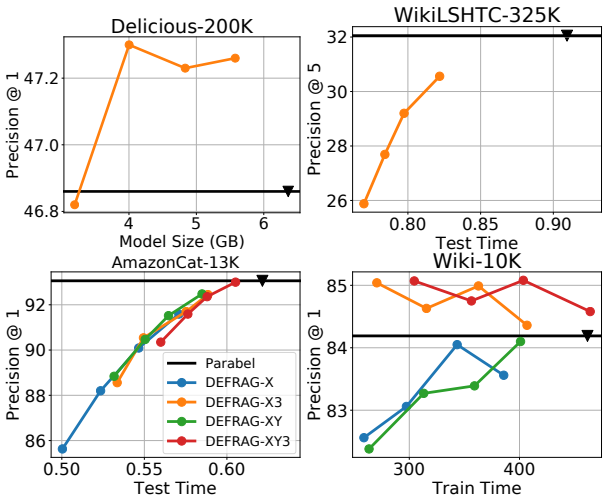

Figure 2: Trade-offs offered by DEFRAG variants. The maximum size of clusters in DEFRAG variants d_0 was changed among the values $32, 16, 8$ (default), and 4. The plots show how this affects prediction accuracy, training/test time and model sizes. The black line shows Parabel's default performance with the black triangle marking its model size, test time etc. Aggressive clustering (e.g. $d_0 = 32$) gives faster training/prediction times and smaller model sizes but also some drop in accuracy. DEFRAG-X3 and DEFRAG-XY3 refer to an ensemble of 3 independent realizations of DEFRAG (please see the full version for more details and additional trade-off plots.

Hyperparameters. If available, hyperparameter settings recommended by authors were used for all methods. If unavailable, a fine grid search was performed over a reasonable range to offer adequately tuned hyperparameters to the methods. DEFRAG had its only hyperparameter, the max size of a feature cluster d_0 (see Algorithm 1), fixed to 8.

Comparison with other clustering methods. Table 1 compares DEFRAG with other clustering algorithms on clustering quality, execution time and classification performance (please see the full version for definitions of clustering metrics). Features were agglomerated according to feature clusters given by all algorithms and Parabel was executed on them. DEFRAG handily outperforms all other methods.

Dataset-wise and method-wise performance. Table 3 presents the outcome of using DEFRAG with several leading algorithms on 8 datasets. On Wiki10 and Delicious, DEFRAG-XY+Parabel offers the best overall performance across all methods. More generally, the table shows 21 instances, across the 8 datasets, of how DEFRAG performs with various algorithms. In 3 of these instances, DEFRAG outperforms the base method (EURLex-PPDSparse, Wiki10-Parabel and Delicious-Parabel), in 7 others, DEFRAG lags by less than 2.5%, in 8 others, it lags by less than 5%. Only in 3 cases is the lag $> 5\%$. DEFRAG variants do seem to work best with the Parabel method.

Trade-offs offered by DEFRAG. It is easy to see that if we create a small number of clusters K, by setting d_0 to be a large value, then the agglomerated vectors will be lower dimensional and as such, offer faster training/prediction and smaller model sizes. However this may cause a dip in prediction accuracy. Figure 2 shows that DEFRAG variants offer attractive trade-offs in this respect.

Method	P1	P3	P5	N1	N3	N5
Wiki-10K						
PfastReXML‖	19.02	18.34	18.43	19.02	18.49	18.52
REFRAG	**20.56**	**19.51**	**19.26**	**20.56**	**19.74**	**19.54**
Delicious-200K						
PfastReXML‖	3.15	3.87	4.43	3.15	3.68	4.06
REFRAG	**7.34**	**8.05**	**8.66**	**7.34**	**7.86**	**8.27**

Table 2: REFRAG with propensity scored metrics. N1,3,5 refer to propensity weighted nDGG@k. ‖Values from [Bhatia *et al.*, 2019].

Rare-label prediction with REFRAG. Table 2 shows that REFRAG offers much better propensity-weighted metrics [Jain *et al.*, 2016] (which down-weigh popular and emphasize rare labels) than PfastreXML which also attempts label re-ranking. Figure 3 shows that REFRAG achieves much better coverage (3.85) than Parabel (1.23) on Delicious and in general, predicts rare labels far more accurately. Figure 3 also shows that FIAT offers resilience to feature erasures.

Acknowledgments

The authors thank the reviewers for comments on improving the presentation of the paper. The authors are also thankful to the lab-team members at the CSE department, IIT Kanpur esp. M. Bagga, S. Malhotra, N. Yadav, B. K. Mishra for their support in running the experiments. P. K. thanks Microsoft Research India and Tower Research for research grants.

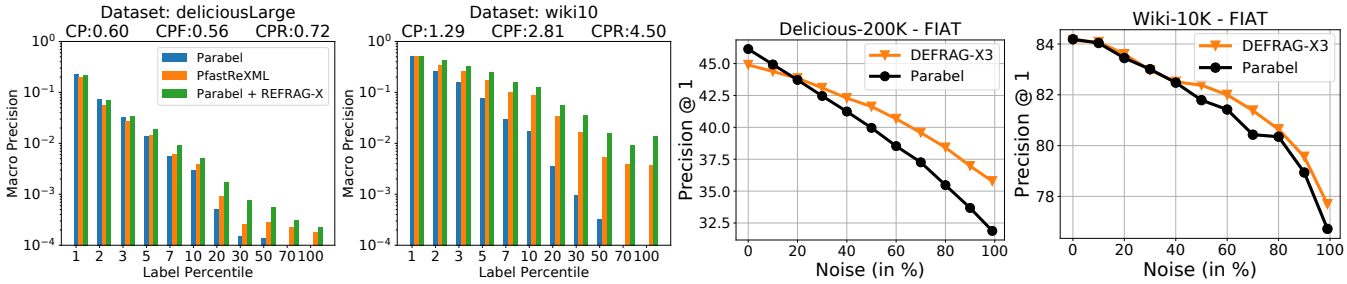

Figure 3: REFRAG offers far superior coverage of rare labels than Parabel on Wiki10 and Delicious datasets. Please see the full version for definitions of coverage and other details. In the last two plots, a fraction of features was randomly erased from the feature vectors of all test data points. The FIAT algorithm is more robust to such erasures than the default Parabel algorithm. The gap between FIAT and Parabel widens as erasures become more common. Please see the full version for more details and plots.

Method	P1(%)	P3(%)	P5(%)	Total Time (hr)	Train Time (hr)	Test Time (ms)	Model Size (GB)
EURLex-4K							
PfastReXML	70.41	59.22	50.56	0.07	0.07	1.26	0.26
DEFRAG-X	68.50	56.57	47.78	0.05	0.05	1.55	0.19
SLEEC	72.96	56.03	45.49	0.06	0.06	1.62	0.70
DEFRAG-X	67.89	51.55	42.04	0.03	0.03	1.05	0.31
Dismec	82.85	70.37	58.69	0.04	0.04	1.1	0.08
DEFRAG-X	79.12	66.39	54.97	0.02	0.02	0.5	0.02
PPDSparse	72.90	57.1	45.8	0.010	0.010	0.03	0.01
DEFRAG-X	71.40	**57.9**	**47.4**	0.006	0.006	0.05	0.01
Parabel	82.28	68.81	57.58	0.010	0.010	0.73	0.03
DEFRAG-X	78.97	65.68	54.46	0.009	0.008	0.59	0.01
DEFRAG-XY	79.23	65.77	54.65	0.009	0.008	0.59	0.01
ProXML‡	83.40	70.90	59.10	-	-	-	-
Wiki10-31K							
PfastReXML	75.67	64.55	57.35	0.27	0.27	11.94	1.12
DEFRAG-X	69.79	58.54	52.52	0.14	0.13	11.74	0.80
SLEEC	84.28	72.05	61.80	0.38	0.38	6.00	3.9
DEFRAG-X	83.87	70.35	59.76	0.17	0.17	3.50	2.0
Dismec	84.12	74.71	65.94	1.48	1.48	42	7.1
DEFRAG-X	82.30	72.14	63.78	0.66	0.65	15	1.5
PPDSparse	74.68	60.03	49.12	0.59	0.59	2.2	0.04
DEFRAG-X	63.55	50.42	41.20	0.50	0.50	3.7	0.03
Parabel	84.19	72.46	63.37	0.13	0.13	2.04	0.18
DEFRAG-X	84.99	73.47	63.91	0.10	0.09	1.46	0.14
DEFRAG-XY	**85.08**	**73.76**	**64.06**	0.11	0.09	1.47	0.14
Amazon-670K							
PfastReXML	36.90	34.22	32.10	5.70	5.70	6.10	10.98
DEFRAG-X	32.67	30.27	28.40	2.70	2.69	7.21	9.40
SLEEC	32.48	28.87	26.31	2.22	2.22	1.43	8.0
DEFRAG-X	31.40	28.04	25.69	1.63	1.63	1.62	4.2
Parabel	44.92	39.77	35.98	0.24	0.24	0.81	1.94
DEFRAG-X	42.71	37.71	33.93	0.23	0.21	0.76	1.68
DEFRAG-XY	42.62	37.72	33.94	0.23	0.21	0.77	1.66
DiSMEC §	44.70	39.70	36.10	-	-	-	-
ProXML‡	43.50	38.70	35.30	-	-	-	-

Method	P1(%)	P3(%)	P5(%)	Total Time (hr)	Train Time (hr)	Test Time (ms)	Model Size (GB)
AmazonCat-13K							
PfastReXML	85.56	75.19	62.84	11.66	11.66	0.54	19.02
DEFRAG-X	84.71	73.48	61.19	7.01	7.00	0.53	16.17
Dismec†	93.80	79.07	64.05	6.68	6.68	1.45	6.0
DEFRAG-X	89.39	74.90	60.67	4.19	4.18	0.89	1.1
Parabel	93.06	79.15	64.51	0.43	0.43	0.62	0.61
DEFRAG-X	91.70	77.25	62.79	0.44	0.40	0.57	0.39
DEFRAG-XY	92.36	78.20	63.55	0.43	0.40	0.58	0.38
Delicious-200K							
Parabel	46.86	40.08	36.70	5.33	5.33	2.22	6.36
DEFRAG-X	47.23	40.53	37.19	3.23	3.16	1.05	4.83
DEFRAG-XY	**47.61**	**40.90**	**37.66**	3.34	3.12	1.06	4.76
PfastReXML‖	41.72	37.83	35.58	-	-	-	-
DiSMEC §	45.50	38.70	35.50	-	-	-	-
WikiLSHTC-325K							
PfastReXML	58.47	37.70	27.57	11.23	11.23	2.66	14.20
DEFRAG-X	50.86	32.08	23.40	6.03	6.03	2.14	12.82
Parabel	65.04	43.24	32.05	0.58	0.58	0.91	3.09
DEFRAG-X	59.49	39.25	29.20	0.56	0.50	0.79	2.50
DEFRAG-XY	61.38	40.42	29.99	0.54	0.50	0.78	2.44
PPDSparse‖	64.08	41.26	30.12	-	-	-	-
DiSMEC §	64.40	42.50	31.50	-	-	-	-
ProXML‡	63.60	41.50	30.80	-	-	-	-
Wikipedia-500K*							
Parabel	68.70	49.57	38.64	5.13	5.13	3.11	5.68
DEFRAG-X	65.15	44.96	34.85	3.27	3.14	1.62	5.25
DEFRAG-XY	64.73	44.79	34.76	3.31	3.20	1.62	5.22
DiSMEC §	70.20	50.60	39.70	-	-	-	-
ProXML‡	69.00	49.10	38.80	-	-	-	-
Amazon-3M*							
Parabel	47.42	44.66	42.55	3.14	3.14	0.73	31.43
DEFRAG-X	45.68	42.85	40.76	2.93	2.83	0.66	25.34
DEFRAG-XY	45.11	42.36	40.30	2.87	2.80	0.63	25.22
PfastReXML‖	43.83	41.81	40.09	-	-	-	-

Table 3: DEFRAG's performance when used with various extreme classification algorithms. "Total time" for DEFRAG includes clustering=train time. On EURLex, Wiki10 and Delicious, DEFRAG actually achieves better classification accuracy than the base classifier itself (bold items). DEFRAG allows expensive 1-vs-all methods like DiSMEC and PPDSparse to be executed in a scalable manner with training time reductions of upto 40% on AmazonCat and model size reductions of upto 20% on WikiLSHTC. Precision values on certain datasets are being reported for sake of easy comparison. These were not obtained in our experiments and are being sourced from original publications: §[Babbar and Schölkopf, 2017], ‡[Babbar and Schölkopf, 2019], ‖[Bhatia *et al.*, 2019].
* For all but Wiki-500K and Amazon-3M, DEFRAG was executed in an ensemble of 3 independent realizations (details in the full version).
† Except for DiSMEC on AmazonCat (which required 12 cores to execute scalably), all times are reported on a single core.

References

[Babbar and Schölkopf, 2017] Rohit Babbar and Bernhard Schölkopf. DiSMEC - Distributed Sparse Machines for Extreme Multi-label Classification. In *10th ACM International Conference on Web Search and Data Mining (WSDM)*, 2017.

[Babbar and Schölkopf, 2019] Rohit Babbar and Bernhard Schölkopf. Data Scarcity, Robustness and Extreme Multi-label Classification. *Machine Learning*, 2019. https://doi.org/10.1007/s10994-019-05791-5.

[Banerjee and Ghosh, 2006] Arindam Banerjee and Joydeep Ghosh. Scalable Clustering Algorithms with Balancing Constraints. *Data Mining and Knowledge Discovery*, 13(3):365–395, 2006.

[Bhatia *et al.*, 2015] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse Local Embeddings for Extreme Multi-label Classification. In *29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[Bhatia *et al.*, 2019] Kush Bhatia, Kunal Dahiya, Himanshu Jain, Yashoteja Prabhu, and Manik Varma. The Extreme Classification Repository. http://manikvarma.org/downloads/XC/XMLRepository.html, 2019. Retrieved on February 15, 2019.

[Chen and Cai, 2011] Xinlei Chen and Deng Cai. Large Scale Spectral Clustering with Landmark-Based Representation. In *25th AAAI Conference on Artificial Intelligence (AAAI)*, 2011.

[Dhillon *et al.*, 2003] Inderjit S. Dhillon, Subramanyam Mallela, and Rahul Kumar. A Divisive Information-Theoretic Feature Clustering Algorithm for Text Classification. *Journal of Machine Learning Research*, 3:1265–1287, 2003.

[Jain *et al.*, 2016] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications. In *22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[Jasinska *et al.*, 2016] Kalina Jasinska, Krzysztof Dembczyński, Róbert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hüllermeier. Extreme F-Measure Maximization using Sparse Probability Estimates. In *33rd International Conference on Machine Learning (ICML)*, 2016.

[Liu *et al.*, 2017] Hanyang Liu, Junwei Han, Feiping Nie, and Xuelong Li. Balanced Clustering with Least Square Regression. In *31st AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[Prabhu and Varma, 2014] Yashoteja Prabhu and Manik Varma. FastXML: A Fast, Accurate and Stable Tree-classifier for eXtreme Multi-label Learning. In *20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2014.

[Prabhu *et al.*, 2018] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising. In *27th International World Wide Web Conference (WWW)*, 2018.

[Siblini *et al.*, 2018] Wissam Siblini, Pascale Kuntz, and Frank Meyer. CRAFTML, an Efficient Clustering-based Random Forest for Extreme Multi-label Learning. In *35th International Conference on Machine Learning (ICML)*, 2018.

[Tagami, 2017] Yukihiro Tagami. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *23rd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

[Wei and Li, 2018] Tong Wei and Yu-Feng Li. Does Tail Label Help for Large-Scale Multi-Label Learning. In *27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[Weston *et al.*, 2013] Jason Weston, Ameesh Makadia, and Hector Yee. Label Partitioning For Sublinear Ranking. In *30th International Conference on Machine Learning (ICML)*, 2013.

[Yen *et al.*, 2017] Ian E. H. Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit S. Dhillon, and Eric Xing. PPDSparse: A Parallel Primal-Dual Sparse Method for Extreme Classification. In *23rd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

[Yu *et al.*, 2014] Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit S. Dhillon. Large-scale Multi-label Learning with Missing Labels. In *31st International Conference on Machine Learning (ICML)*, 2014.