Shrinking the Size of Deep Extreme Multi-Label Classification

Marco Bornstein[*†] **Tahseen Rabbani**[*‡]

Brian Gravelle[§] **Furong Huang**[†]

Abstract

Training deep classifiers for Extreme Multi-Label Classification (XMC) is difficult due to the computational and memory costs caused by extremely large label sets. Traditionally, the final output layer of these deep classifiers scales linearly with the size of the label set, which is often in the millions for realistic settings. Reducing the size of deep classifiers for XMC is necessary to (i) train them more efficiently and (ii) deploy them within memory-constrained systems, such as mobile devices. We address the current limitations of deep classifiers by proposing a novel XMC method, DECLARE: Deep Extreme Compressed Labeling And Recovery Estimation. DECLARE compresses the labels into a smaller dimension, which reduces the training time and model-storage size. DECLARE retains enough information to recover the most-likely predicted labels in the original label space. Empirically, DECLARE compresses labels by up to 99.975% while outperforming uncompressed performance.

1 Introduction

Many online services use recommender systems to foster a personalized customer experience by assigning each product an individual label. These systems have grown to extreme sizes as online services have expanded the past two decades. For example, there are over 600 million products for sale on Amazon alone One [2024]. The output dimension of recommender systems is equivalent to the number of recommended products, so the computational and memory costs become extreme when these methods are applied to systems with millions of products. Fittingly, the name for this situation is Extreme Multi-Label Classification (XMC) Bhatia et al. [2016].

The goal of our work is to alleviate the computational- and memory-constraints of XMC when training recommender systems that leverage modern deep-learning methods. We propose DECLARE: Deep Extreme Compressed Labeling And Recovery Estimation, a novel deep learning XMC algorithm. The main contribution of DECLARE is that it reduces the computational and memory costs of training large-scale recommender systems. Namely, DECLARE: **(1)** compresses the label dimension up to 99.975% by capturing key information within the compressed space that other compression methods fail to utilize, **(2)** trains a small neural network to learn the compression mapping from the feature space to the compressed label space, **(3)** implements a recovery method to transform compressed predictions back to the uncompressed label space during inference time, without the need to train multiple networks, and **(4)** outperforms or closely matches uncompressed performance while reducing the label dimension by several orders of magnitude on five common XMC datasets.

[*]Equal Contribution. Correspondence to Marco Bornstein: marcob@umd.edu.

[†]Department of Computer Science, University of Maryland, College Park, MD, USA.

[‡]Department of Biomedical Informatics and Data Science, Yale University, CT, USA.

[§]Laboratory for Physical Sciences, University of Maryland

Preprint. Under review.

Table 1: DECLARE's Test Performance vs. (Un)compressed Baselines.

Dataset	DECLARE		Unstructured Pruning		Uncompressed	
	P@1	Output Dim.	P@1	Output Dim.	P@1	Output Dim.
BibTex	62.43%	100	62.73%	100	62.86%	159
EURLex-4.3K	83.24%	200	83.15%	200	86.65%	4,271
AmazonCat-13K	87.55%	200	82.52%	200	89.74%	13,330
Wiki10-31K	82.92%	100	51.68%	100	64.56%	30,938
Delicious-200K	46.12%	50	6.08%	50	40.84%	205,443

1.1 Related Work

Deep XMC. Similar to our work, existing literature is concerned with memory constraints Chen et al. [2020b,a], Rabbani et al. [2024]. In Chen et al. [2020b], a locality-sensitive hashing scheme (SLIDE) is used to prune the final classification layer of an XMC network. However, the full-scale weight structure must be maintained in memory and the extent of pruning will vary batch-to-batch. Chen et al. [2020a] considers a staggered update of hashes to avoid continuous batch-to-batch re-hashing, but the full weight structure must still be occasionally restored, resulting in memory blowup. Rabbani et al. [2024] designs a modified family of hashing functions to avoid the need for restoring or maintaining the full weight structure on a low-fidelity local device, but it requires the assistance of a centralized server to host the original weight. [Yan et al., 2022] proposes model splitting to alleviate the size of XMC layers across multiple nodes, though in the presence of few workers, the per-client share of the model is memory infeasible and thus inapplicable to standalone training.

Compressed Sensing. A subset of existing XMC literature focuses on compression of a large label space into a smaller one, known as compressed sensing Bhatia et al. [2015], Zhou et al. [2012], Tai and Lin [2012], Dasgupta et al. [2023]. Zhou et al. [2012] and Tai and Lin [2012] utilize a random and SVD-based compression matrix to reduce the dimensionality of their label space. In Bhatia et al. [2015], a low-rank embedding is optimized to preserve nearest-neighbor distances and sparsity. Unlike DECLARE, the methods above are not deep learning methods. As a result, these methods require a large number of training instances, since each input-class pair needs its own classification model. Therefore, as the number of recommended products grows, so too do the number of training instances, resulting in larger training and storage costs for large-scale recommender systems.

2 Problem Setting

The goal of XMC is to train a classifier that maps text samples to labels within an extremely large label set. We denote $\mathcal{D} := \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ as the dataset, having n data samples. Let $\boldsymbol{x}_i \in \mathbb{R}^d$ be the input feature vector while $\boldsymbol{y}_i \in \{0,1\}^L$ be its corresponding label vector. The sparse feature and label sets are defined as $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^T \in \mathbb{R}^{n \times d}$ and $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n]^T \in \mathbb{R}^{n \times L}$ Denote f as our objective function, defined below, $\theta \in \mathbb{R}^f$ as the parameters of our deep classifier, $h_\theta(\cdot)$ as our deep classifier parameterized by θ, and ℓ as our loss function.

$$\min_\theta f(\theta; \boldsymbol{x}, \boldsymbol{y}) := \frac{1}{n} \sum_i^n \ell\big(h_\theta(\boldsymbol{x}_i), \boldsymbol{y}_i\big). \tag{1}$$

While Equation (1) can be solved through gradient descent approaches, the major issue at hand is that the label dimension, L, is large. As a result, the number of parameters, f, for the deep classifier is also large, resulting in expensive storage, inference $h_\theta(\boldsymbol{x})$, and gradient back-propagation $\nabla_\theta h_\theta(\boldsymbol{x})$.

3 Compressed Labeling & Recovery Estimation

DECLARE performs information-aware compression of the original labels to produce compressed labels $\boldsymbol{z} \in \mathbb{R}^c$, where $c << L$. Then, DECLARE trains a feed-forward neural network, h_θ, to learn the mapping from $\boldsymbol{x}$ to $\boldsymbol{z}$. Finally, DECLARE uses a recovery-estimation scheme, proposed in Section 3.2, that takes a compressed label $\boldsymbol{z}$ as input and outputs the most-likely predicted label classes from its corresponding original label $\boldsymbol{y}$ (which is unknown during inference).

Algorithm 1 Deep Extreme Compressed Labeling And Recovery Estimation

Require: Data (X_{trn}, Y_{trn}) & (X_{tst}, Y_{tst}), number of training samples n, original label dimension L, compressed label dimension c, q neighbors, top k, and T iterations

Compute the truncated SVD: $U_c^Y, \Sigma_c^Y, (V_c^Y)^T \leftarrow \text{TSVD}(Y_{trn}, c)$
Generate new compressed labels: $Z_{trn} \leftarrow U_c^Y * \Sigma_c^Y$
Train NN h_θ for T iterations (or convergence) on the compressed data (X_{trn}, Z_{trn})
for each test data sample $\boldsymbol{x}_i, \boldsymbol{y}_i \in (X_{tst}, Y_{tst})$ **do**
    Predict compressed label: $\boldsymbol{z}_i^p \leftarrow h_\theta(\boldsymbol{x}_i)$
    Compute similarity scores: $\boldsymbol{s}^{p,z} \leftarrow Z_{trn}\boldsymbol{z}_i^p$
    Find q-closest indices of Z_{trn}: $I^s \leftarrow \arg\max_{I \subseteq [n]:|I|=q} \sum_{i \in I} \boldsymbol{s}_i^{p,z}$
    Weight the true labels in I^s by their relative distances: $\boldsymbol{f}^p \leftarrow (\boldsymbol{s}_{I_s^p}^{p,z})^T Y_{I^s}$
    Select the top k largest label classes $I^y \leftarrow \arg\max_{I \subseteq [L]:|I|=k} \sum_{i \in I} \boldsymbol{f}_i^p$
    Compute and store the P@k accuracy
end for

3.1 Label Compression via SVD Approximation

Since the labels, $\boldsymbol{y}$, are sparse, we believe that the information stored in $\boldsymbol{y}$ can be distilled down into a much smaller, dense vector $\boldsymbol{z}$. Namely, we aim to preserve the similarity between each label, $\boldsymbol{y}_i$, and all labels in the label set $\boldsymbol{y}_j, \forall j \in [n]$. In matrix form, this is denoted as YY^T. We solve the following optimization problem in order to determine the best low-rank approximation $Z \in \mathbb{R}^{n \times c}$ to Y,

$$\underset{\text{rank}(Z) \leq c}{\arg\min} \left\| YY^T - ZZ^T \right\|_F. \tag{2}$$

The best rank-c approximation of YY^T are the first c singular vectors and values (via the Eckart-Young-Mirsky Theorem Eckart and Young [1936]),

$$\underset{\text{rank}(Z) \leq c}{\arg\min} \left\| YY^T - ZZ^T \right\|_F = \underset{\text{rank}(Z) \leq c}{\arg\min} \left\| U\Sigma^2 U^T - ZZ^T \right\|_F = U_c \Sigma_c. \tag{3}$$

From Equation (3), the optimal method for label compression is to compute the truncated SVD of Y, select the top c left singular vectors and singular values, and then perform the matrix product to obtain the compressed labels Z. While effective, approximating a large and sparse Y to extremely low dimensions via the SVD can cause issues in datasets where there are many tail labels: labels that have few training samples from which the model can learn. As a result, important tail-label information within Y is lost when using a low-rank SVD approximation. We provide discussion of future methods to combat this issue and further refine our SVD label compression method above within Appendix A. In DECLARE, we compress our original labels via Equation (3) to receive compressed labels Z, and train a neural network to learn the mapping between the feature and compressed label spaces $h_\theta(\boldsymbol{x}_i) = \boldsymbol{z}_i$.

3.2 Label Recovery

As the output of our neural network $h_\theta(\cdot)$ is within the compressed space, a label-recovery mechanism is necessary to recover the most-likely true predicted label classes in the original label space. As detailed in Section 3.1, the compressed labels Z are optimized to best approximate the similarity between each label in Y. Thus, by computing the inner product between a predicted compressed label $\boldsymbol{z}^p$ and all existing compressed labels Z, we approximate the similarity $\boldsymbol{s}$ between the original, and unknown, uncompressed prediction $\boldsymbol{y}^p$ with all labels Y,

$$\boldsymbol{s}^{p,z} = Z\boldsymbol{z}^p \approx Y\boldsymbol{y}^p = \boldsymbol{s}^{p,y}. \tag{4}$$

Equation (4) not only defines a relationship between $\boldsymbol{z}_p$ and $\boldsymbol{y}_p$, it directly relates the true and low-rank similarity vectors $\boldsymbol{s}^{p,y}, \boldsymbol{s}^{p,z} \in \mathbb{R}^n$. The indices I^s of the q largest values in $\boldsymbol{s}^{p,z}$ approximate the indices of the q most similar labels $Y_{I^s} := Y[I^s, :]$ within the dataset to the unknown true prediction $\boldsymbol{y}^p$,

$$I^s := \underset{I \subseteq [n]:|I|=q}{\arg\max} \sum_{i \in I} \boldsymbol{s}_i^{p,z}. \tag{5}$$

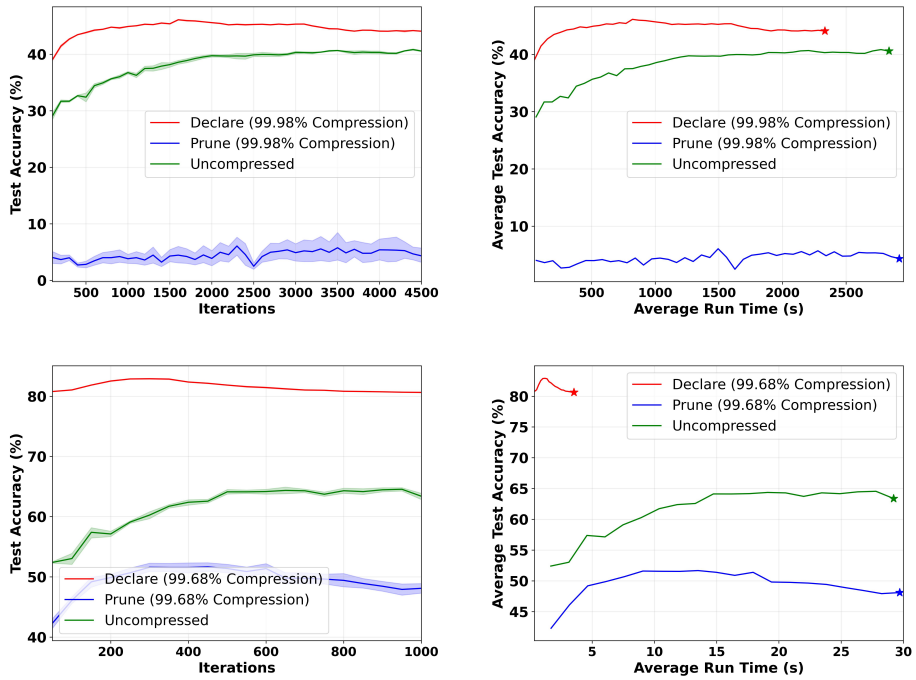

Figure 1: **Simultaneous Performance and Efficiency Improvement**. DECLARE outperforms its uncompressed counterpart by 5.28% and 18.36% on Delicious-200K (top) and Wiki10-31K (bottom) while reducing the final output dimension by 99.975% and 99.677% respectively. DECLARE outperforms the baselines in terms of test accuracy (left) and convergence (right).

In summary, we compute the top-q indices of $\boldsymbol{s}^{p,z}$ and use their corresponding labels within Y as the closest approximate labels Y_{I^s} to $\boldsymbol{y}^p$. Once the closest labels are found, they are scaled element-wise by their similarity scores and summed to determine the most frequent label classes,

$$\boldsymbol{f}^p := (\boldsymbol{s}^{p,z}_{I^p_s})^T Y_{I^s}, \quad I^y := \underset{I \subseteq [L]:|I|=k}{\arg\max} \sum_{i \in I} \boldsymbol{f}^p_i. \tag{6}$$

The indices I^y are the top-k largest label classes and are thus selected in DECLARE as the k most-likely true label classes. We use I^y to compute the Precision@k accuracy metric widely used within XMC settings.

4 Experiments

We compare DECLARE with the original, uncompressed neural network as well as a neural network trained with unstructured pruning, one of the best deep-learning compression techniques. We test DECLARE on five common XMC datasets Bhatia et al. [2016].

Datasets. Our experiments use five publicly available datasets: BibTex Katakis et al. [2008] (159 labels), EURLex-4.3K Loza Mencía and Fürnkranz [2010] (4,271 labels), AmazonCat-13K McAuley and Leskovec [2013] (13,330 labels), Wiki10-31K Zubiaga [2012] (30,938), and Delicious-200K Wetzker et al. [2008] (205,443 labels). Success across this variety of datasets (see Table 1) indicates that DECLARE will scale to real-world applications.

Unequal Baselines. DECLARE is at a disadvantage when compared to the uncompressed neural network or a neural network pruned via unstructured pruning. The uncompressed network is much larger and contains many more parameters than DECLARE. As a result, we expect the uncompressed network to be an upper bound on how well DECLARE can perform. However, we observe below that this is not always true; DECLARE significantly outperforms the uncompressed network on our largest XMC dataset, Delicious-200K. Unstructured pruning, like DECLARE, trains a reduced (pruned)

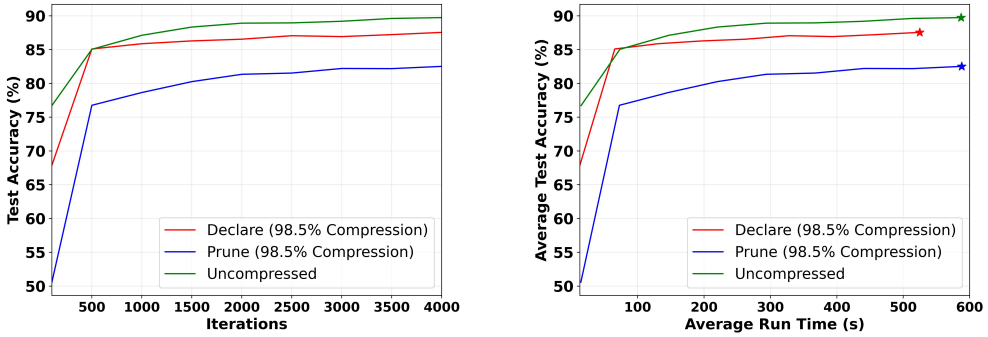

Figure 2: DECLARE outperforms Unstructured Pruning while keeping pace with the uncompressed network under 98.5% compression (left). Furthermore, DECLARE trains faster than uncompressed and compressed baselines (right) for AmazonCat-13K.

version of a model. Unstructured pruning serves as an upper bound for how well pruning-based compression performs. Unlike its structured counterpart, unstructured pruning sets a given number of parameters to zero (irregular sparsity) and generally performs better Wang et al. [2019], Yao et al. [2019]. As a result, the size of an unstructured pruning model is the same as the uncompressed network albeit with many more parameters set to zero. This incurs larger memory costs than DECLARE. Furthermore, due to its irregular sparsity, unstructured pruning is more difficult to exploit for efficient computation Anwar et al. [2017]. Many modern deep learning packages, like TensorFlow and PyTorch, cannot handle irregular sparsity to speed up tensor computations. As a result, unstructured pruning does not provide much computational speed-up, as shown in Figure 1.

Hyper-parameters & Evaluation. We train a feed-forward neural network with one hidden layer having 256 neurons (following previous deep-learning XMC papers Chen et al. [2020b,a], Rabbani et al. [2024]). For BibTex, EURLex-4.3K, Wiki10-31K, and Delicious-200K we use a batch size of 128 while we use a batch size of 256 for AmazonCat-13K. The recovery scheme uses 25 nearest neighbors for all experiments, and we train using ADAM with a standard learning rate of 5e-4. Each experiment is run three times, with the average results being shown in Table 1 as well as the dark lines on all our Figures. The test accuracies listed are Precision@k (we use $k = 1$), a metric widely used in XMC Bhatia et al. [2015, 2016], Chen et al. [2020b,a], Rabbani et al. [2024]. This metric gauges the percentage of k predicted label classes that are indeed found within a target label $\boldsymbol{y}_i$.

Computation and Memory Reduction Without Sacrificing Performance. The main takeaway from our experimental results is that DECLARE can dramatically reduce memory and computational costs while reducing performance by no more than 3% across all datasets. Furthermore, DECLARE can *improve* performance while simultaneously reducing memory and computational costs by up to 99.975% versus an uncompressed neural network. On Delicious-200K, DECLARE achieves 46.12% P@1 accuracy compared to 40.84% for its uncompressed counterpart, while only requiring an output dimension of 50 (compared to the original dimension of 205,443). DECLARE also performs better as the output dimension grows (see Table 1). While unstructured pruning is competitive on smaller datasets, DECLARE outpaces it on the larger AmazonCat-13K and Delicious-200K datasets. DECLARE outperforms unstructured pruning on the larger datasets due to its distance-preserving compression scheme.

Improved Convergence Speed. Another advantage that DECLARE has over its peers is that it trains faster and converges quicker to peak performance. Figure 1 shows DECLARE outperforming its baselines at the beginning of training. DECLARE trains faster than the uncompressed network due its greatly reduced size (its final layer weight is reduced by upwards of four magnitudes). While unstructured pruning compresses the original network by a similar amount as DECLARE, its irregular sparsity requires a full forward pass since PyTorch does not support unstructured sparse matrix operations (as detailed above).

5 Conclusion

Our novel XMC algorithm, DECLARE: Deep Extreme Compressed Labeling And Recovery Estimation, addresses the ever-increasing size of recommender systems through intelligent compression and recovery of large and sparse labels. By compressing labels in a manner that preserves label-wise similarity, DECLARE is able to reduce memory footprint and computational costs of recommender systems by up to 99.975%. This is accomplished without sacrificing predictive performance: DECLARE not only maintains near state-of-the-art performance of uncompressed recommender systems, it actually *improves* performance compared to uncompressed systems (by nearly 6%) on large XMC datasets such as Delicious-200K. In summary, DECLARE is an XMC algorithm that efficiently trains and stores large-scale recommender systems without sacrificing performance.

References

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL `http://manikvarma.org/downloads/XC/XMLRepository.html`.

Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. *Advances in neural information processing systems*, 28, 2015.

Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Christopher Re. Mongoose: A learnable lsh framework for efficient neural network training. In *International Conference on Learning Representations*, 2020a.

Beidi Chen, Tharun Medini, James Farwell, Charlie Tai, Anshumali Shrivastava, et al. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *Proceedings of Machine Learning and Systems*, 2:291–306, 2020b.

Arpan Dasgupta, Siddhant Katyan, Shrutimoy Das, and Pawan Kumar. Review of extreme multilabel classification. *arXiv preprint arXiv:2302.05971*, 2023.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multilabel text classification for automated tag suggestion. *ECML PKDD discovery challenge*, 75:2008, 2008.

Eneldo Loza Mencía and Johannes Fürnkranz. *Efficient multilabel classification algorithms for large-scale problems in the legal domain*. Springer, 2010.

Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172, 2013.

Capital One. Number of products on amazon. Technical report, Capital One Shopping Research, Jul 2024. URL `https://capitaloneshopping.com/research/number-of-products-on-amazon/`.

Tahseen Rabbani, Marco Bornstein, and Furong Huang. Large-scale distributed learning via private on-device lsh. *Advances in Neural Information Processing Systems*, 36, 2024.

Farbound Tai and Hsuan-Tien Lin. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*, 2019.

Robert Wetzker, Carsten Zimmermann, and Christian Bauckhage. Analyzing social bookmarking systems: A del. icio. us cookbook. In *Proceedings of the ECAI 2008 Mining Social Data Workshop*, pages 26–30, 2008.

Minghao Yan, Nicholas Meisburger, Tharun Medini, and Anshumali Shrivastava. Distributed slide: Enabling training large neural networks on low bandwidth and simple cpu-clusters via model parallelism and sparsity. *arXiv preprint arXiv:2201.12667*, 2022.

Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced sparsity for efficient dnn inference on gpu. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5676–5683, 2019.

Tianyi Zhou, Dacheng Tao, and Xindong Wu. Compressed labeling on distilled labelsets for multi-label learning. *Machine Learning*, 88:69–126, 2012.

Arkaitz Zubiaga. Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469*, 2012.

A Future Work

The compression method we utilize is fast and effective, but is prone to possibly over-weighting the zero values within an extremely large and not dealing with tail labels in Y (as detailed in Section 3.1). As a result, we are looking into variations of our SVD approach that can alleviate over-weighting of zero values that can possibly cause issues when predicting datasets with heavy-tailed labels.

Nearest-Neighbor Preserving Compression. Within the non-deep learning approach of SLEEC Bhatia et al. [2015], the authors propose preserving the similarity between only a few nearest neighbors of each label. They utilize a masking function P_Ω to only penalize nearest-neighbor similarities between YY^T and ZZ^T. The goal is to ensure that the compressed labels maintain nearest-neighbor information that could be washed away by the non-nearest-neighbor terms when a normal SVD is computed. Mathematically, they solve for Z such that,

$$\underset{\text{rank}(Z)\leq c}{\arg\min} \left\| P_\Omega(YY^T) - P_\Omega(ZZ^T) \right\|_F^2, \tag{7}$$

where the masking function is defined as,

$$[P_\Omega(YY^T)]_{ij} = \begin{cases} \langle \boldsymbol{y}_i, \boldsymbol{y}_j \rangle & \text{if } j \text{ is a neighbor of } i, \\ 0 & \text{else.} \end{cases} \tag{8}$$

In order to solve Equation (7), a projected gradient descent method, with step-size η, is used to solve for the matrix $M = ZZ^T$,

$$M_{t+1} = P_c\bigg(M_t + \eta P_\Omega\big(YY^T - M_t\big) \bigg), \tag{9}$$

where P_c is the projection of M onto the set of rank-c matrices. The projection P_c can be computed via an eigenvalue decomposition or SVD of M, $P_c(M) = U_c^M \Sigma_c^M (U_C^M)^T$. Once M has converged, the compressed labels are determined as $Z = U_c^M (\Sigma_c^M)^{1/2}$ A major issue with this method, however, is that storing the projection P_c is expensive when there are many data samples, as $P_c(M) \in \mathbb{R}^{n \times n}$. To circumvent this issue, Bhatia et al. [2015] cluster the data X, and compute compressed labels Z_k via Equation (9) for the corresponding labels in each k clusters of data.

Our future goal is to implement a nearest-neighbor preserving compression method that is memory-efficient (does not require full storage of P_c) while incorporating SVD approximation. In Bhatia et al. [2015], the initial matrix when performing projected gradient descent is the zero matrix $M_0 = 0$. We hope to initialize M_0 as the SVD approximation to Y or YY^T and preserve nearest-neighbor distances via projected gradient descent while simultaneously penalizing deviation away from the initial SVD at non-nearest neighbor values.