Discrete-Continuous Optimization Framework for Simultaneous Clustering and Training in Mixture Models

Parth Sangani [*1] Arjun Kashettiwar [*1] Pritish Chakraborty [1] Bhuvan Reddy [1] Durga Sivasubramanian [1]
Ganesh Ramakrishnan [1] Rishabh Iyer [2] Abir De [1]

Abstract

We study a new framework of learning mixture models via automatic clustering called PRESTO, wherein we optimize a joint objective function on the model parameters and the partitioning, with each model tailored to perform well on its specific cluster. In contrast to prior work, we do not assume any generative model for the data. We convert our training problem to a joint parameter estimation and a subset selection problem, subject to a matroid span constraint. This allows us to reduce our problem into a constrained set function minimization problem, where the underlying objective is monotone and approximately submodular. We then propose a new joint discrete-continuous optimization algorithm which achieves a bounded approximation guarantee for our problem. We show that PRESTO outperforms several alternative methods. Finally, we study PRESTO in the context of resource efficient deep learning, where we train smaller resource constrained models on each partition and show that it outperforms existing data partitioning and model pruning/knowledge distillation approaches, which in contrast to PRESTO, require large initial (teacher) models.

1. Introduction

In the problem space of learning mixture models, our goal is to fit a given set of models implicitly to different clusters of the dataset. Mixture models are ubiquitous approaches for prediction tasks on heterogeneous data (Dasgupta, 1999; Achlioptas & McSherry, 2005; Kalai et al., 2010; Belkin & Sinha, 2010a; Pace & Barry, 1997; Belkin & Sinha, 2010b; Sanjeev & Kannan, 2001; Hopkins & Li, 2018; Fu & Robles-Kelly, 2008), and find use in a plethora of applications, *e.g.*, finance, genomics (Dias et al., 2009; Liesenfeld, 2001; Pan et al., 2003), *etc*. Existing literature on mixture models predominately focuses on the design of estimation algorithms and the analysis of sample complexity for these problems (Faria & Soromenho, 2010; Städler et al., 2010; Kwon et al., 2019; Yi et al., 2014), and analyzes them theoretically for specific and simple models such as Gaussians, linear regression, and SVMs. Additionally, erstwhile approaches operate on *realizable* settings — they assume specific generative models for the cluster membership of the instances. Such an assumption can be restrictive, especially when the choice of the underlying generative model differs significantly from the hidden data generative mechanism. Very recently, Pal et al. (2022) consider a linear regression problem in a *non-realizable* setting, where they do not assume any underlying generative model for the data. However, their algorithm and analysis is tailored towards the linear regression task.

1.1. Present Work

Responding to the above limitations, we design PRESTO, a novel data partitioning based framework for learning mixture models. In contrast to prior work, PRESTO is designed for generic deep learning problems including classification using nonlinear architectures, rather than only linear models (linear regression or SVMs). Moreover, we do not assume any generative model for the data. We summarize our contributions as follows.

Novel framework for training mixture models. At the outset, we aim to simultaneously partition the instances into different subsets and build a mixture of models across these subsets. Here, each model is tailored to perform well on a specific portion of the instance space. Formally, given a set of instances and the architectures of K models, we partition the instances into K disjoint subsets and train a family of K component models on these subsets, wherein, each model is assigned to one subset, implictly by the algorithm. Then, we seek to minimize the sum of losses yielded

[*]Equal contribution [1]Department of Computer Science and Engineering, IIT Bombay. [2]Department of Computer Science and Engineering, UT Dallas.. Correspondence to: Parth Sangani <parthsangani00@gmail.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

by the models on the respective subsets, jointly with respect to the model parameters and the candidate partitions of the underlying instance space.

Note that our proposed optimization method aims to attach each instance to one of the K models on which it incurs the least possible error. Such an approach requires that the loss function helps guide the choice of the model for an instance, thus rendering it incompatible for use at inference time. We build an additional classifier to tackle this problem; given an instance, the classifier takes the confidence from each of the K models as input and predicts the final label.

Design of approximation algorithm. Our training problem involves both continuous and combinatorial optimization variables. Due to the underlying combinatorial structure, the problem is NP-hard even when all the models are convex. To solve this problem, we first reduce our training problem to a parameter estimation problem in conjunction with a subset selection task, subject to a matroid span constraint (Iyer et al., 2014). Then, we further transform it into a constrained set function minimization problem and show that the underlying objective is a monotone and α-submodular function (El Halabi & Jegelka, 2020; Gatmiry & Gomez-Rodriguez, 2018) and has a bounded curvature. Finally, we design PRESTO, an approximation algorithm that solves our training problem, by building upon the majorization-minimization algorithms proposed in (Iyer & Bilmes, 2015; Iyer et al., 2013a; Durga et al., 2021). We provide the approximation bounds of PRESTO, even when the learning algorithm provides an imperfect estimate of the trained model. Moreover, it can be used to minimize any α-submodular function subject to a matroid span constraint and therefore, is of independent interest.

Application to resource-constrained settings. With the advent of deep learning, the complexity of machine learning (ML) models has grown rapidly in the last few years (Liu et al., 2020; Arora et al., 2018; Dar et al., 2021; Bubeck & Sellke, 2021; Devlin et al., 2018; Liu et al., 2019; Brown et al., 2020). The functioning of these models is strongly contingent on the availability of high performance computing infrastructures, *e.g.*, GPUs, large RAM, multicore processors, *etc*.

The key rationale behind the use of an expensive neural model is to capture the complex nonlinear relationship between the features and the labels across the entire dataset. Our data partitioning framework provides a new paradigm to achieve the same goal, while enabling multiple lightweight models to run on a low resource device. Specifically, we partition a dataset into smaller subslices and train multiple small models on each subslice— since each subslice is intuitively a smaller and simpler data subset, we can train a much simpler model on the subslice thereby significantly reducing the memory requirement. In contrast to approaches such as pruning (Wang et al., 2020a; Lee et al., 2019; Lin et al., 2020; Wang et al., 2020b; Jiang et al., 2019; Li et al., 2020; Lin et al., 2017) and knowledge distillation (Hinton et al., 2015; Son et al., 2021), we do not need teacher models (high compute models) with the additional benefit that we can also train these models on resource constrained devices.

Empirical evaluations. Our experiments reveal several insights, summarized as follows. (1) PRESTO yields significant accuracy boost over several baselines. (2) PRESTO is able to trade-off accuracy and memory consumed during training more effectively than several competitors, *e.g.*, pruning and knowledge distillation approaches. At the benefit of significantly lower memory usage, the performance of our framework is comparable to existing pruning and knowledge distillation approaches and much better than existing partitioning approaches and mixture models. Our code is available at `https://github.com/parthsangani00/PRESTO`.

1.2. Related Work

Mixture Models and Clustering. Mixture Models (Dempster et al., 1977; Jordan & Jacobs, 1994) and k-means Clustering (MacQueen, 1967; Lloyd, 1982) are two classical ML approaches, and have seen significant research investment over the years. Furthermore, the two problems are closely connected and the algorithms for both, *i.e.*, the k-means algorithm and the Expectation Maximization algorithm for mixture models are closely related – the EM algorithm is often called soft-clustering, wherein one assigns probabilities to each cluster. Mixture models have been studied for a number of problems including Gaussian Mixture Models (Xu & Jordan, 1996), Mixtures of SVMs (Collobert et al., 2001; Fu & Robles-Kelly, 2008), and linear regression (Faria & Soromenho, 2010; Städler et al., 2010; Kwon et al., 2019; Pal et al., 2022). Most prior work involving mixture models pertaining to regression is in the *realizable setting*, with the exception of (Pal et al., 2022). Pal et al. (2022) approach the problem of linear regression under the *non-realizable setting* by defining *prediction* and *loss* in the context of mixtures and formulating a novel version of the alternating maximisation (AM) algorithm. In recent years, there is an interest in mixture of experts models (Shazeer et al., 2017; Chen et al., 2022) which select a sparse combination from a set of expert models using a trainable gating mechanism to process each input. Such an approach focus on end-to-end differentiable training mechanism, whereas our approach optimizes for a combinatorial loss. We provide more connection with our approach and mixture of experts in Section 2.3.

Resource-constrained learning. In the pursuit of better performance, most state of the art deep learning models are often over-parameterized. This makes their deployment in the resource constrained devices nearly impossible. To mitigate the problems, several handcrafted architectures such as SqueezeNets (Iandola et al., 2016; Gholami et al., 2018), MobileNets (Howard et al., 2017; Sandler et al., 2018) and ShuffleNets (Zhang et al., 2018; Ma et al., 2018) were designed to work in mobile devices. Recently, EfficientNet (Tan & Le, 2019) was proposed, that employs neural architecture search. However, they are designed to work on the entire training set, and leave a heavy memory footprint.

Simultaneous model training and subset selection. Data subset selection approaches are predominately static, and most often, do not take into account the model's current state (Wei et al., 2014a;b; Kirchhoff & Bilmes, 2014; Kaushal et al., 2019; Liu et al., 2015; Bairi et al., 2015; Lucic et al., 2017; Campbell & Broderick, 2018; Boutsidis et al., 2013). Some recent approaches attempt to solve the problem of subset selection by performing joint training and subset selection (Mirzasoleiman et al., 2020a;b; Killamsetty et al., 2021b;a; Durga et al., 2021). On the other hand, some other approaches (Mirzasoleiman et al., 2020a; Killamsetty et al., 2021a) select subsets that approximate the full gradient. Further, some of these approaches (Killamsetty et al., 2021b;a) demonstrate improvement in the robustness of the trained model by selecting subsets using auxiliary or validation set.

2. Problem Formulation

In this section, we first present the notations and then formally state our problem. Finally, we connect our proposal with mixture of expert model.

2.1. Notations

We have N training instances $\{(\boldsymbol{x}_i, y_i) \,|\, 1 \le i \le N\}$, where $\boldsymbol{x}_i \in \mathcal{X}$ is the feature vector and $y_i \in \mathcal{Y}$ is the label of the i^{th} instance. In our work, we set $\mathcal{X} = \mathbb{R}^d$ and treat $\mathcal{Y}$ to be discrete. Given K, we denote $h_{\boldsymbol{\theta}_1}, .., h_{\boldsymbol{\theta}_K}$ as the component models that are going to be used for the K subsets resulting from a partition of $\mathcal{X}$. Here $\boldsymbol{\theta}_k$ is the trainable parameter vector of $h_{\boldsymbol{\theta}_k}$. These parameters are not shared across different models, *i.e.*, there is no overlap between $\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}_{k'}$ for $k \neq k'$. In fact, the models $h_{\boldsymbol{\theta}_k}$ and $h_{\boldsymbol{\theta}_{k'}}$ can even have different architectures. Moreover, the representation of the output $h_{\boldsymbol{\theta}_k}$ can vary across different settings. For example, given $\boldsymbol{x} \in \mathcal{X}$, $\text{sign}(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}))$ is a predictor of y for support vector machines with $\mathcal{Y} \in \{\pm 1\}$ whereas, for multiclass classification, $h_{\boldsymbol{\theta}_k}(\boldsymbol{x})$ provides a distribution over $\mathcal{Y}$. To this end, we use $\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}), y)$ to indicate the underlying loss function for any instance $(\boldsymbol{x}, y)$. We define $[A] = \{1, \ldots, A\}$ for an integer A.

2.2. Problem Statement

High level objective. Our broad goal is to fit a mixture of models on a given dataset, without making any assumption concerning the generative process, the instances or features. Given a family of model architectures, our aim is to learn to partition the instance space $\mathcal{X}$ into a set of subsets, determine the appropriate model architecture to be assigned to each subset and to subsequently train the appropriate models on the respective subsets.

Problem statement. We are given the training instances D, the number of subsets K resulting from partitioning D and a set of model architectures $h_{\boldsymbol{\theta}_1}, \ldots, h_{\boldsymbol{\theta}_K}$. Our goal, then, is to partition the training set D into K subsets $S_1, ..S_K$ with $S_k \cap S_{k'} = \emptyset$ and $\cup_k S_k = D$ so that when the model $h_{\boldsymbol{\theta}_k}$ is trained on S_k for $k \in [K]$, the total loss is minimized. To this end, we define the following regularized loss, *i.e.*,

$$F(\{(S_k, \boldsymbol{\theta}_k) \,|\, k \in [K]\}) = \sum_{k=1}^{K} \sum_{i \in S_k} \ell\left(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i\right) + \lambda ||\boldsymbol{\theta}_k||^2. \tag{1}$$

Then, we seek to solve the following constrained optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K; S_1,\ldots S_K}{\text{minimize}} \quad F(\{(S_k, \boldsymbol{\theta}_k) \,|\, k \in [K]\}) \\ &\text{subject to, } \; S_k \cap S_{k'} = \emptyset, \; \forall k \in [K], \cup_{k \in [K]} S_k = D. \end{aligned} \tag{2}$$

Here, λ is the coefficient of the L_2 regularizer in Eq. (2). The constraint $S_k \cap S_{k'} = \emptyset$ ensures that that each example $i \in D$ belongs to exactly one subset and the constraint $\cup_{k \in [K]} S_k = D$ entails that the subsets $\{S_k \,|\, k \in [K]\}$ cover the entire dataset. The above optimization problem is a joint model parameter estimation and a partitioning problem. If we fix the partition $\{S_1, S_2, ..., S_K\}$, then the optimal parameter vector $\boldsymbol{\theta}_k$ depends only on S_k, the subset assigned to it. To this end, we denote the optimal value of $\boldsymbol{\theta}_k$ for the above partition as $\boldsymbol{\theta}_k^*(S_k)$ and transform the optimization (2) into the following equivalent problem:

$$\begin{aligned} &\underset{S_1, S_2, \ldots, S_K}{\text{minimize}} \; F(\{S_k, \boldsymbol{\theta}_k^*(S_k) \,|\, k \in [K]\}) \\ &\text{subject to, } S_k \cap S_{k'} = \emptyset, \; \forall k \in [K], \cup_{k \in [K]} S_k = D. \end{aligned} \tag{3}$$

Note that, computing $\boldsymbol{\theta}_k^*(S_k)$ has a polynomial time complexity for convex loss functions. However, even for such functions, minimizing F as defined above is NP-hard.

Test time prediction. Since computation of the optimal partitioning requires us to solve the optimization (3), it cannot be used to assign an instance $\boldsymbol{x}$ to a model $h_{\boldsymbol{\theta}_k}$ during the test time. To get past this blocker, we train an additional multiclass classifier π_ϕ, which combines $h_{\boldsymbol{\theta}_k}(\boldsymbol{x})$ for all k and predict the label for the unseen instance $\boldsymbol{x}$.

2.3. Connection and Difference with Mixture of Experts

Let us consider the following MoE model. The label y is computed using $\sum_{k=1}^{K} \pi_\phi(\boldsymbol{x})[k] h_{\boldsymbol{\theta}_k}(\boldsymbol{x})$. Here, K is the number of experts, π_ϕ is a gating network which is typically modelled as a probability distribution over the experts and $h_{\boldsymbol{\theta}_k}$ is the k-th expert model. Here, the predicted probability for y is computed by averaging over the gating probabilities. Thus, the corresponding training loss becomes $\sum_{i \in D} \ell\left(\sum_{k=1}^{K} \pi_\phi(\boldsymbol{x}_i)[k] h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i\right)$. This can also be written as $\sum_{i\in D} \ell(\mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i)], y_i)$.

Now, instead of computing average over the expert networks and then computing loss, we can first compute loss for a given expert and then compute average over experts as follows: $\sum_{i\in D} \mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)]$. By Jensen inequality, $\sum_{i\in D} \mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)] \geq \sum_{i\in D} \ell(\mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i)], y_i))$ and therefore, $\sum_{i\in D} \mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)]$ minimises the error more aggressively. Note that Jensen's inequality holds in this case because we need the loss function to be convex w.r.t $h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i)$ (not necessarily the model parameters θ). This will hold for loss functions such as SVM hinge loss, cross entropy loss, etc. In our setup, we minimize the second loss $\sum_{i\in D} \mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)]$ in the combinatorial sense. Here, we do not use a neural gating network. Instead, for each $\boldsymbol{x}_i$, we assume $\pi(\boldsymbol{x}_i)[k]$ to be the probability that the instance $\boldsymbol{x}_i$ belongs to expert k. Here, $\pi(\boldsymbol{x}_i)$ and $\pi(\boldsymbol{x}_j)$ do not share any parameter—they are different variables. Consider the following continuous optimization problem:

$$\min_{\phi, \{\boldsymbol{\theta}_k\}} \sum_{i\in D} \mathbb{E}_{k\sim\pi_\phi(\boldsymbol{x}_i)}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)] \tag{4}$$

It can be seen as an approximation of the following combinatorial optimization problem:

$$\min_{\{\pi(\boldsymbol{x}_i)\}_{i\in D}, \{\boldsymbol{\theta}_k\}_{k\in[K]}} \sum_{i\in D} \sum_{k\in[K]} \pi(\boldsymbol{x})[k][\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}), y)] \tag{5}$$

$$\text{subject to, } \sum_{k\in[K]} \pi(\boldsymbol{x}_i)[k] = 1, \tag{6}$$

$$\pi(\boldsymbol{x}_i)[k] \geq 0, k \in [K], i \in D \tag{7}$$

as long as the gating network π_ϕ is highly expressive, so that it can model all the independent variables $\pi(\boldsymbol{x}_i)$ via a shared neural network. Now, the above problem is a linear program w.r.t. $\pi(\boldsymbol{x}_i)$, the solution of $\pi(\boldsymbol{x}_i)$ lies in the vertices of the linear constraints and thus at optimal point, we have $\pi(\boldsymbol{x}_i)[k] = 1$ or 0. This effectively reduces to our problem as given below:

$$\min_{\{S_k, \boldsymbol{\theta}_k\}_{k\in[K]}} \sum_{i\in S_k} \sum_{k\in[K]} \ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i),$$

$$\text{where, } S_k = \{i : \pi(\boldsymbol{x}_i)[k] = 1\} \tag{8}$$

We believe that our combinatorial formulation offers two advantages over MoE setup. (1) It minimizes a more aggressive loss which is the average of losses on different subsets. (2) The gating network is rather a soft version of our proposed combinatorial subset assignment problem. Moreover, during end to end training, the trained models may be inaccurate, as the gating network π_ϕ only approximates π. Note that, our approach also involves π_ϕ, but the model training is decoupled from π_ϕ.

2.4. Application in the Resource-Constrained Setup

In general, the relationship between an instance $\boldsymbol{x}$ and the label y can be arbitrarily nonlinear. A complex deep neural network captures such relationship by looking into the entire dataset. However, such networks require high performance computing infrastructure for training and inference. The training problem (3) can be used to build K lightweight models $h_{\boldsymbol{\theta}_1}, \ldots, h_{\boldsymbol{\theta}_K}$, each of which is localized to a specific regime of the instance space $\mathcal{X}$. Thus, a model is required to capture the nonlinearity only from the region assigned to it and not the entire set $\mathcal{X}$. As a result, it can be lightweight and used with limited resources.

Consider a large model with number of parameters equal to the total number of parameters collectively across the K models, *i.e.*, $\sum_{k=1}^{K} \dim(\boldsymbol{\theta}_k)$. Such a model has to be loaded entirely into a GPU RAM for training or inference, and thus requires a larger GPU RAM. In contrast, our approach requires us to load at a time, only one model component $h_{\boldsymbol{\theta}_k}$ and the corresponding subset S_k during both training and test time. This is instead of having to load all the K model components and the entire dataset. While this can increase both training and inference time, it can substantially reduce the memory consumption by $1/K$ times in comparison to a large model having similar expressiveness.

3. PRESTO: Proposed Framework to Solve the Training Problem (3)

In this section, we first show that the optimization problem (3) is equivalent to minimizing a monotone set function subject to matroid span constraint (Iyer et al., 2014; Schrijver et al., 2003). Subsequently, we show that this set function is α-submodular and admits a bounded curvature. Finally, we use these results to design PRESTO, an approximation algorithm to solve the underlying constrained set function optimization problem. We next present these analyses, beginning with the necessary definitions about monotonicity, α-submodularity and different matroid related properties.

Definition 1. (1) Monotonicity, α-submodularity and generalized curvature: Given a ground set $\mathbb{V}$, let $G : 2^{\mathbb{V}} \to \mathbb{R}$ be a set function whose marginal gain is denoted by $G(e \,|\, \mathbb{S}) = G(\mathbb{S} \cup \{e\}) - G(\mathbb{S})$. The function G is monotone non-decreasing if $G(e\,|\,\mathbb{S}) \geq 0$ for all $\mathbb{S} \subset \mathbb{V}$ and $e \in \mathbb{V}\backslash\mathbb{S}$. G is α-submodular with $\alpha \in (0,1]$ if $G(e\,|\,\mathbb{S}) \geq \alpha G(e\,|\,\mathbb{T})$ for all $\mathbb{S} \subseteq \mathbb{T}$ and $e \in \mathbb{V}\backslash\mathbb{T}$ (Hashemi et al., 2019; El Halabi & Jegelka, 2020). The generalized curvature of $G(\mathbb{S})$ is defined as $\kappa_G(\mathbb{S}) = 1 - \min_{e\in\mathbb{S}} G(e\,|\,\mathbb{S}\backslash e)/G(e\,|\,\emptyset)$ (Iyer et al., 2013b; Zhang & Vorobeychik, 2016). **(2) Base, rank and span of a matroid:** Consider a matroid $\mathcal{M} = (\mathbb{V}, \mathcal{I})$ where $\mathcal{I}$ is the set of independent sets (Refer to Appendix A.1 for more details). A base of $\mathcal{M}$ is a maximal independent set. The rank function $r_{\mathcal{M}} : 2^{\mathbb{V}} \to \mathbb{N}$ is defined as: $r_{\mathcal{M}}(\mathbb{S}) = \max_{I\in\mathcal{I}: I\subset\mathbb{S}} |I|$. A set $\mathbb{S}$ is a spanning set if it is the superset of a base or equivalently, $r_{\mathcal{M}}(\mathbb{S}) = r_{\mathcal{M}}(\mathbb{V})$ (Schrijver et al., 2003; Iyer et al., 2014; Edmonds, 2003).

3.1. Representation of (3) as a Matroid Span Constrained Subset Selection Task

Transforming partitions into 2D configurations. Given the training instances D and the size of partition K, we first define the ground set $\mathbb{V}$ in the space of Cartesian products of D and $[K]$, *i.e.*, $\mathbb{V} = D \times [K]$. Thus $\mathbb{V}$ consists of all pairs $\{(i,k)\,|\, i \in D, k \in [K]\}$ which enumerates all possible assignments between the instances and model components. Moreover, we define $\mathbb{V}_{i\star} = \{(i,k)\,|\,k \in [K]\}$ and $\mathbb{V}_{\star k} = \{(i,k)\,|\, i \in D\}$. Here, $\mathbb{V}_{i\star} = \{i\} \times [K]$ enumerates all possible assignments of the i^{th} instance and $\mathbb{V}_{\star k} = D \times \{k\}$ enumerates all possible configurations specifically wherein S_k is assigned to an instance.

Optimization (3) as constrained subset selection. Having defined the ground set $\mathbb{V}$ and the partitions in 2D configuration space as above, we define $\mathbb{S} = \{(i,k)\,|\, i \in S_k, k \in [K]\}$ that encodes the set of assignments in space of $\mathbb{V}$ induced by the underlying partition. Then, the set $\widehat{\mathbb{S}}_k = \{(i,k)\,|\, i \in S_k\}$ specifies the set of instances attached to the subset k and elucidates the subset containing i. It can be observed that $\widehat{\mathbb{S}}_k = \mathbb{S} \cap \mathbb{V}_{\star k}$. Since every instance is assigned exactly to one subset $S_k \in \{S_1, \ldots, S_K\}$, we have that $|\mathbb{S} \cap \mathbb{V}_{i\star}| = 1$. To this end, we introduce the following set function, which is the sum of the fitted loss functions in Eq. (1), trained over individual subsets.

$$G(\mathbb{S}) = \sum_{k\in[K]} \sum_{(i,\bullet)\in\mathbb{S}\cap\mathbb{V}_{\star k}} \left[\ell\left(h_{\boldsymbol{\theta}^*_k(\mathbb{S}\cap\mathbb{V}_{\star k})}(\boldsymbol{x}_i), y_i\right) + \lambda||\boldsymbol{\theta}_k||^2\right]. \tag{9}$$

We rewrite our optimization problem (3) as follows:

$$\underset{\mathbb{S}\subset\mathbb{V}}{\text{minimize}}\; G(\mathbb{S}) \text{ subject to, } |\mathbb{S} \cap \mathbb{V}_{i\star}| = 1 \;\; \forall\, i \in D. \tag{10}$$

Theoretical characterization. Here, we show that our objective admits monotonicity, α-submodularity and bounded curvature in a wide variety of scenarios (Proven in Appendix A.2)

Theorem 2. *Given the set function $G(\mathbb{S})$ (9) and the individual regularized loss functions ℓ introduced in Eq. (1), we define: $\varepsilon_{\min} = \min_{k,i} Eigen_{\min}[\nabla^2_{\boldsymbol{\theta}_k} \ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)]$, $\ell_{\min,\min} = \min_{i\in D,\boldsymbol{\theta}_k}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i) + \lambda||\boldsymbol{\theta}_k||^2$ and $\ell_{\min,\max} = \max_{i\in D}\min_{\boldsymbol{\theta}_k}[\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i) + \lambda||\boldsymbol{\theta}_k||^2]$. Then, we have the following results:*

(1) Monotonicity: The function $G(\mathbb{S})$ defined in Eq. (9) is monotone non-decreasing in $\mathbb{S}$.

(2) α-submodularity: If $\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}), y)$ is L-Lipschitz for all $k \in \{1,..,K\}$ and the regularizing coefficient λ satisfies: $\lambda > -\varepsilon_{\min}$, function $G(\mathbb{S})$ is α-submodular with

$$\alpha \geq \alpha_G = \frac{\ell_{\min,\min}}{\ell_{\min,\min} + \frac{2L^2}{\lambda + 0.5\varepsilon_{\min}} + \frac{\lambda L^2}{(\lambda+0.5\varepsilon_{\min})^2}} \tag{11}$$

(3) Generalized curvature: The generalized curvature $\kappa_G(\mathbb{S})$ for any set $\mathbb{S}$ is given by: $\kappa_G(S) \leq \kappa^*_G = 1 - \ell_{\min,\min}/\ell_{\min,\max}$.

Solving the optimization (10) is difficult due to the equality constraint. However, as suggested by Theorem 2 (1), $G(\mathbb{S})$ is monotone in $\mathbb{S}$. Hence, even if we relax the equality constraints $|\mathbb{S} \cap \mathbb{V}_{i\star}| = 1$ to the inequality constraint $|\mathbb{S} \cap \mathbb{V}_{i\star}| \geq 1$, they achieve the equality at the optimal solution $\mathbb{S}^*$. Thus, the optimization (10) becomes

$$\underset{\mathbb{S}\subset\mathbb{V}}{\text{minimize}}\, G(\mathbb{S}) \text{ subject to, } |\mathbb{S} \cap \mathbb{V}_{i\star}| \geq 1 \;\; \forall\, i \in D. \tag{12}$$

As we formally state in the following proposition, the above constraint (set) can be seen as a matroid span constraint for a partition matroid. This would allow us to design an approximation algorithm to solve this problem.

Proposition 3. *Let the set $\mathbb{S}$ satisfies $|\mathbb{S} \cap \mathbb{V}_{i\star}| \geq 1$ for all $i \in D$. Then $\mathbb{S}$ is a spanning set of the partition matroid $\mathcal{M} = (\mathbb{V}, \mathcal{I})$ with $\mathcal{I} = \{I\,|\,|I \cap \mathbb{V}_{i\star}| \leq 1\}$. Moreover, if $\mathbb{S}$ satisfies $|\mathbb{S} \cap \mathbb{V}_{i\star}| = 1$ for all $i \in D$, then $\mathbb{S}$ is a base of $\mathcal{M}$.*

The above proposition suggests that the optimal solution $\mathbb{S}^*$ of the optimization (12) is a base of the partition matroid $\mathcal{M} = (\mathbb{V}, \mathcal{I})$ with $\mathcal{I} = \{I\,|\,|I \cap \mathbb{V}_{i\star}| \leq 1\}$.

3.2. PRESTO: Our Proposed Approximation Algorithm

Here, we aim to *minimize* a monotone approximate-submodular function G. In contrast to submodular or approximate submodular maximization where one can use greedy algorithms, minimizing the approximate submodular function G needs a completely different approach. Here, we resort to minimizing an upper bound m of G— reducing the value of this upper bound m will ensure the value of the underlying function G remains low. Now, we need to design m such that maximizing m is convenient. To this aim, we choose m to be modular. One can connect such a modular approximation of a set function to a simple linear approximation of a complex nonlinear function in the context of continuous optimization. Specifically, we build PRESTO upon Durga et al. (2021) which employs Majorization-Minimization approach (Iyer et al., 2013a;b). Toward that goal, we first develop the necessary ingredients

Algorithm 1 The PRESTO Algorithm

```
Require: Training data $D$, $\alpha_G$, $K$ model architectures, Iterations.
 1: Output: The learned parameters $\{\boldsymbol{\theta}_k \mid k \in [K]\}$, the partitioning of $D$: $D = \cup_{k\in[K]} \widehat{S}_k$
 2: $\widehat{S}_1 \leftarrow \emptyset, .., \widehat{S}_K \leftarrow \emptyset$
 3: for $k \in [K]$ do
 4:     $\hat{\boldsymbol{\theta}}_k \leftarrow$ InitParams( )
 5:     for all $i \in D$ do
 6:         $G(\{(i,k)\}) \leftarrow$ Train$(\{i\}\,; \hat{\boldsymbol{\theta}}_k)$
 7:     end for
 8: end for
 9: for $r \in$ [Iterations] do
10:     for $k \in [K]$ do
11:         $\widehat{\mathbb{S}}_k \leftarrow \left\{(i,k) \mid i \in \widehat{S}_k\right\}$
12:         $\hat{\boldsymbol{\theta}}_k, G(\widehat{\mathbb{S}}_k) \leftarrow$ Train$(\widehat{S}_k; \boldsymbol{\theta}_k)$
13:         for $i \in \widehat{S}_k$ do
14:             $G(\widehat{\mathbb{S}}_k \backslash (i,k)) \leftarrow$ Train$(\widehat{S}_k \backslash i; \boldsymbol{\theta}_k)$
15:             $\boldsymbol{M}[i][k] \leftarrow \alpha_G[G(\widehat{\mathbb{S}}_k) - G(\widehat{\mathbb{S}}_k \setminus \{(i,k)\})]$
16:         end for
17:         for all $i \notin \widehat{S}_k$, set $\boldsymbol{M}[i][k] = \dfrac{G(\{(i,k)\})}{\alpha_G}$
18:     end for
19:     for $p \in [N]$ do
20:         $(i^*, k^*) \leftarrow \text{argmin}_{i,k}(\boldsymbol{M}[i][k])$
21:         $\widehat{S}_{k^*} \leftarrow \widehat{S}_{k^*} \cup i^*$
22:         For all $k \neq k^*$ : $\widehat{S}_k \leftarrow \widehat{S}_k \backslash \{i^*\}$
23:         For all $k$ : $\boldsymbol{M}[i^*][k] \leftarrow \infty$
24:     end for
25:     $\widehat{\mathbb{S}} \leftarrow \left\{(i,k) \mid i \in \widehat{S}_k, k \in [K]\right\}$
26: end for
27: Return $\hat{\boldsymbol{\theta}}_1, ..., \hat{\boldsymbol{\theta}}_K, \widehat{\mathbb{S}}$
```

as follows.

Computation of modular upper bound. First, we present a modular upper bound for $G(\mathbb{S})$ (Iyer et al., 2013a). Given a fixed set $\widehat{\mathbb{S}}$ and the set function G which is α-submodular and monotone, let the modular function $m^G_{\widehat{\mathbb{S}}}[\mathbb{S}]$ be defined as follows:

$$m^G_{\widehat{\mathbb{S}}}[\mathbb{S}] = G(\widehat{\mathbb{S}}) - \sum_{(i,k)\in\widehat{\mathbb{S}}} \alpha_G G((i,k) \mid \widehat{\mathbb{S}} \setminus \{(i,k)\}) + \sum_{(i,k)\in\widehat{\mathbb{S}}\cap\mathbb{S}} \alpha_G G((i,k) \mid \widehat{\mathbb{S}} \setminus \{(i,k)\}) + \sum_{(i,k)\in\mathbb{S}\setminus\widehat{\mathbb{S}}} \frac{G((i,k) \mid \emptyset)}{\alpha_G}. \tag{13}$$

If G is α-submodular and monotone, it holds that $G(\mathbb{S}) \leq m^G_{\widehat{\mathbb{S}}}[\mathbb{S}]$ for all $\mathbb{S} \subseteq \mathcal{D}$ (Durga et al., 2021). Note that when G is submodular, *i.e.*, when $\alpha = 1$, the expression $m^G_{\widehat{\mathbb{S}}}[\mathbb{S}]$ gives us the existing modular upper bounds for submodular functions (Nemhauser et al., 1978; Iyer et al., 2013a; Iyer & Bilmes, 2012).

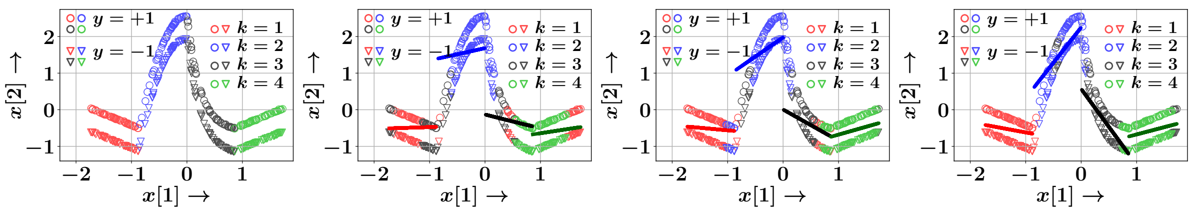
Outline of PRESTO (Alg. 1). Minimizing $m^G_{\widehat{\mathbb{S}}}[\mathbb{S}]$ is the same as finding the lowest values of $\alpha_G G((i,k)|\hat{\mathbb{S}} \setminus \{(i,k)\})$ or $G((i,k)|\emptyset)/\alpha_G$ depending on whether $(i,k) \in \hat{\mathbb{S}}$ (third term in Eq (13)) or $(i,k) \notin \hat{\mathbb{S}}$ (fourth term in Eq (13)). We compute these two quantities in line nos. 15 and 17 in Algorithm 1 and store the values of m at different pairs (i,k) in the matrix $\boldsymbol{M}$. The complexity of this operation is $O(|D|K)$. Finally, we take the minimum of $\boldsymbol{M}$ to find (i^*, k^*) (line 20) which indicates that the partition k^* should contain i^*. Therefore, we include i^* to $\hat{S}_{k^*}$ (line 21). Now, since any instance can belong to exactly one partition, we remove i^* from all other partition $k \neq k^*$ (line 22). Finally, we update $\hat{\mathbb{S}}$ (line 25). Finally, note that to evaluate G, we need to train the algorithm on the specific datapoints and partition, as encapsulated in the Train( ) routine in Algorithm 1. Here, Train(S, θ_k) trains the model for partition k on the subset $\widehat{S}_k$ for a few iterations and, returns the estimated parameters $\hat{\boldsymbol{\theta}}_k$ and the objective $G(\widehat{\mathbb{S}}_k)$.

Approximation guarantee. The following theorem shows that if G is α-submodular with $\alpha > \alpha_G$, and curvature $\kappa > \kappa_G$, Algorithm 1 enjoys an approximation guarantee (Proven in Appendix A.3).

Theorem 4. *Given the set function $G(\mathbb{S})$ defined in Eq. (9), let OPT be an optimal solution of the optimization problem (10) (or equivalently (12)). If Algorithm 1 returns $\widehat{\mathbb{S}}$ as the solution, then $G(\widehat{\mathbb{S}}) \leq G(OPT)/[(1-\kappa_G)\alpha_G^2]$, where κ_G and α_G are computed using Theorem 2.*

Discussion on quality of approximation guarantee: Note that, for classification task, we will always have $\ell_{\min,\min} > 0$. Consequently, our algorithm will always have a non-trivial approximation guarantee. While the quality of the guarantee may differ depending on the setting, we would like to highlight some cases where the quality of approximation guarantee will be high. Note that the approximation guarantee has a stronger dependence on α than κ since Theorem 4 suggests that the approximation factor is $1/((1-\kappa)\alpha^2)$. Therefore, to obtain a high quality approximation guarantee, it is crucial to ensure α to be as much close to 1 as possible. By looking into the expression of α, we note that $\alpha \to 1$ is possible when the loss function is heavily convex with respect to θ, so that the minimum eigenvalue of $\nabla^2_\theta \ell(h_{\theta_k}(x_i), y_i)$ is high. This leads to the high values of $\varepsilon_{\min}$. Consequently the second and third term of the above expression become small and α becomes high. Apart from that, even when λ is high or the loss function is stable during training, we obtain α to be close to 1 and the resulting function is close to submodular. Therefore, in these cases, we will have a better approximation guarantee.

The approximation factor also depends on κ which is the curvature. We would like κ to be close to 0 whereby our algorithm has an approximation factor close to 1. If we observe the expression in Theorem 2 (point number 3), this is true if $\ell_{\min,\min} \approx \ell_{\min,\max}$ which can happen if a) the regularization factor λ is slightly high, and b) the differ-



(a) True Partitions (b) Epoch $r = 10$ (c) Epoch $r = 20$ (d) Epoch $r = 50$

Figure 1. Snapshot of the true partition (panel (a)); and, the snapshots of the trained models $\{h_{\boldsymbol{\theta}_k} \,|\, k \in [K]\}$ and the partitions $\{S_k \,|\, k \in [K]\}$ predicted by PRESTO (panels (b)–(d)) on the synthetic dataset during progression of Algorithm 1. The dataset is generated using a degenerate mixture model with $K = 4$ components. Here, an instance $(\boldsymbol{x}_i, y_i)$ belongs to exactly one of the four sets $\{S^*_k \,|\, k \in [4]\}$ with probability 1, where S^*_k are defined in Eqs. (14)— (17). Each model component is an SVM, *i.e.*, $h_{\boldsymbol{\theta}_k}(\boldsymbol{x}) = \boldsymbol{w}_k^\top \boldsymbol{x} + b_k$ and $\ell(\hat{y}, y) = (1 - \hat{y}y)_+$. We observe that as r increases, PRESTO becomes more and more accurate in assigning an instance to the correct mixture component and finally, at $r = 50$, it is almost able to recover the true mixture component— the ground truth assignments of the instances (panel (a)) is extremely close to the final assignments (panel (d)).

ence between the losses across different points are not too high, which means that the obtained clusters have similar hardness. We will add these details in the revised version of the paper. Note that there exists a slightly tighter bound in Appendix A.3 (Eq. (41)), which shows that even when $\kappa = 1$ in the worst case, we will have a bounded approximation guarantee with $|D|/\alpha_G$. Note that Theorem 4 in Appendix A.3 indicates a data dependent approximation bound. However, for brevity, we preferred to mention a data independent approximation bound in the main paper.

4. Experiments on Synthetic Data

In this section, we experiment with synthetically generated instances from a latent degenerate mixture distribution and show that, Algorithm 1 can accurately recover the partitions which correspond to the true mixture components.

4.1. Experimental Setup

Dataset generation. We generate $|D| = 20000$ examples $\{(\boldsymbol{x}_i, y_i)\}_{i \in D}$ where $\mathcal{X} = [-1.73, 1.73] \times [-1.17, 2.59]$ and $\mathcal{Y} = \{-1, +1\}$. The instances $(\boldsymbol{x}, y)$ belong to one of the $K = 4$ sets, *viz.*, $S^*_1, S^*_2, S^*_3, S^*_4$ defined as follows:

$$S^*_1 = \{(\boldsymbol{x}, y) \,|\, 7x[1] + x[2] - 2y - 27 = 0 \\ \wedge \boldsymbol{x} \in [-1.73, -0.86] \times [-1.16, 0.04]\} \quad (14)$$

$$S^*_2 = \big\{(\boldsymbol{x}, y) \,|\, 20(x[1]^2) - 200x[1] + 414 + x[2] - 2y \\ \wedge \boldsymbol{x} \in [-0.87, 0.01] \times [-1.15, 2.59]\big\} \quad (15)$$

$$S^*_3 = \{(\boldsymbol{x}, y) \,|\, x[2] + 1 + \exp(-x[1]) + 2y \\ \wedge\, \boldsymbol{x} \in [-0.01, 0.87] \times [-1.17, 2.56]\} \quad (16)$$

$$S^*_4 = \{(\boldsymbol{x}, y) \,|\, 7x[1] + x[2] - 2y + 43 = 0 \\ \wedge \boldsymbol{x} \in [0.86, 1.73] \times [-1.17, 0.04]\} \quad (17)$$

Panel (a) of Figure 1 shows a scatter plot of $(\boldsymbol{x}, y)$. The instances are homogeneously distributed across all components, *i.e.*, $|S^*_k| = 5000$.

Choice of $f_{\boldsymbol{\theta}_k}$ and ℓ. We consider $K = 4$ linear support vector machines $h_{\boldsymbol{\theta}_k}(x) = \boldsymbol{w}_k^\top \boldsymbol{x} + b_k$ with $\boldsymbol{\theta}_k = \{\boldsymbol{w}_k, b_k\}$ set ℓ as the margin based hinge loss, *i.e.*, $\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}), y) = (1 - y(\boldsymbol{w}_k \boldsymbol{x}_k + b_k))_+$. Then we apply PRESTO (Algorithm 1) to simultaneously learn the parameters $\{(\boldsymbol{w}_k, b_k) \,|\, 1 \le k \le 4\}$ and the partitions S_k such that $D = \cup_{k=1}^4 S_k$.

4.2. Results

In Figure 1, we plot different snapshots of the models $h_{\boldsymbol{\theta}_k}$ and the partitions $D = \cup_{k=1}^4 S_k$, as PRESTO progresses during different iterations r (line 9 in Algorithm 1). We make the following observations: (1) As r increases, the classification accuracy increases. For $r = 50$, we observe that PRESTO is able to recover the true partitions, *i.e.*, $S^*_k \approx S_k$ (2) We observe that PRESTO finds it relatively difficult to correctly assign the instances on S^*_2 and S^*_3 to the right mixture components. This is due to two reasons: first, the instances on S^*_2 and S^*_3 fit naturally to a nonlinear SVMs, whereas we attempt to fit a linear SVM; second, the instances in S^*_2 and S^*_3, which are around $(0, 0)$, are close to each other— this poses difficulty in demarcating their model components.

5. Experiments with Real Data

In this section, we provide a comprehensive evaluation of our method on three real world classification datasets and several baselines. Next, we evaluate PRESTO in the context of resource constrained deep learning methods. Appendix E contains additional experimental results.

5.1. Experimental Setup

Datasets. We experiment with three real world classification datasets, *viz.*, CIFAR10 (Krizhevsky, 2009), CIFAR100 (Krizhevsky, 2009) and STL (Coates et al., 2011). Following previous work (Chen et al., 2022), we rotate each image by 30 degrees and include the rotated images

	Accuracy $\mathbb{P}(\hat{y}=y)$			Macro F1-score		
Method	CIFAR10	CIFAR100	STL	CIFAR10	CIFAR100	STL
Kmeans++	87.31	65.78	74.95	87.28	65.08	74.29
Agglomerative	81.19	69.68	78.68	81.12	69.02	78.43
MoE	90.02	85.19	79.73	90.06	85.13	79.67
GMM	87.31	64.51	73.42	87.28	62.61	72.82
Learn-MLR	87.95	81.82	81.93	87.94	81.81	81.91
PRESTO	95.49	94.86	90.07	95.48	94.85	90.06

Table 2. Comparison of classification accuracy $\mathbb{P}(\hat{y} = y)$ and Macro-F1 score of PRESTO against two unsupervised partitioning methods (Kmeans++ (Arthur & Vassilvitskii, 2006), Agglomerative (Müllner, 2011)); and three mixture models Mixture of Experts (MoE) (Shazeer et al., 2017) (GMM (Bishop & Nasrabadi, 2006) and Learn-MLR (Pal et al., 2022)) for all datasets. In all cases, we used exactly the same model architecture for each component, which is a ResNet18 network with reduced capacity (described in Section 5.1) The best and second best results are highlighted in green and yellow respectively.

with the original one. Finally, the task is to predict if the image is rotated, which is a binary classification problem. As argued in Chen et al. (2022), such rotated dataset provide some clusters around the object labels, which leads to a natural source of heterogeneity. Note that, traditional (unrotated) CIFAR10 dataset does not contain any natural mixture of diverse models and may not be a suitable dataset for our purpose, as also suggested by the marginal improvement of MoE model from single model (Chen et al., 2022).

Implementation details. For all models, we extract features $\boldsymbol{x}$ from sixth layer of a pre-trained ResNet18 (He et al., 2016). We design $h_{\boldsymbol{\theta}_k}$ using a neural architecture similar to ResNet18 with reduced capacity– specifically, starting with the seventh layer and until the eighth layer, we reduce the number of convolutional filters to 8 instead of 256 and 16 instead of 512, respectively. In each case, we set the number of model components K using cross validation. Specifically, for our model, we set $K = 6$ for CIFAR10 and $K = 4$ for each of CIFAR100 and STL. We use a fully connected single layer neural network to model the additional classifier π_ϕ that is used to decide the model component to be assigned to an instance during inference. Further details are in Appendix D.

5.2. Results

Comparison with partitioning methods. We first compare our method against several unsupervised partitioning methods and mixture models. In the case of unsupervised partitioning, we use clustering methods to slice the data into K partitions before the start of the training and then use these data partitions to train the K models. Specifically, we consider two clustering methods, *viz.*, (1) Kmeans++ (Arthur & Vassilvitskii, 2006), a variant of K-means method that uses a smarter initialisation of cluster centers. (2) Agglomerative clustering (Müllner, 2011), where the clusters are built hierarchically, and three mixture models, *viz.*, (3) Mixture of Experts (MoE) (Chen et al., 2022), a Sparsely-Gated Mixture-of-Experts layer (MoE), which selects a sparse combination from a set of expert models using a trainable gating mechanism to process each input, and two mixture models, *viz.*, (4) Gaussian mixture models for classification (GMM) (Bishop & Nasrabadi, 2006) and (5) Learn-MLR (Pal et al., 2022) which presents a version of the AM algorithm for learning mixture of linear regressions. We adapt this algorithm for classification as a baseline for PRESTO. Across all baselines, we employed exactly same set of model architectures. Details of these methods is given in Appendix D.

In Table 2, we summarize the results in terms of classification accuracy $\mathbb{P}(\hat{y} = y)$ and Macro-F1 score. We make the following observations. (i) PRESTO demonstrates significantly better predictive accuracy than all the baselines for all three datasets. MoE model is the second best method, which outperforms all other baselines in CIFAR10 and CIFAR100 datasets. Note that the objective of the MoE model is a surrogate of our objective (2), as explained in Section 2.3 (Eq. (4) vis-a-vis Eq. (8)). The quality of approximation provided by this MoE surrogate depends strongly how well the gating network π_ϕ can generalize the cluster assignment of each instance $\boldsymbol{x}_i$ (Eq. (7) vs Eq. (8)). Moroever, MoE performs end-to-end training. Thus, the inaccuracy of the gating network affects the quality of the trained model. In contrast, we first optimize the true objective directly using Algorithm 1 and then train π_ϕ using the direct supervision from the partitions $\{S_k \mid k \in [K]\}$ we learned. (ii) Learn-MLR is the second best performer in the STL dataset and is the third best performer in CIFAR10 and CIFAR100 datasets. (iii) Among loss oblivious clustering baselines, Kmeans++ shows extremely poor performance. But Agglomerative outperforms GMM, in CIFAR100 and STL.

In Table 3, we compare the performance of PRESTO with the partitioning methods in terms of clustering metrics such as Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), Inter-cluster similarity (ICS) and Intra-cluster dissimilarity (ICD). Note that NMI and ARI require ground truth clusters for a fair comparison. We use the original ground truth object labels as surrogate ground truth cluster labels. We expect that in CIFAR10 (100), we will have 10 (100) clusters each containing similar objects (car, air-

	CIFAR10		CIFAR100	
Method	NMI	ARI	NMI	ARI
Kmeans++	2e-4	1e-4	1e-3	9e-5
MoE	3e-4	-3e-4	14e-4	-3e-4
Learn-MLR	2e-4	1e-3	2e-3	-4e-3
PRESTO	1e-1	6e-2	15e-2	16e-2

	CIFAR10		CIFAR100	
Method	ICS	ICD	ICS	ICD
Kmeans++	0.326	0.676	0.326	0.676
MoE	0.325	0.675	0.256	0.743
Learn-MLR	0.326	0.676	0.326	0.676
PRESTO	0.399	0.690	0.436	0.712

Table 3. Comparison of Normalized Mutual Information (NMI), Adjusted Rand Index (ARI) (top half); and, Intra-cluster similarity (ICS) and Inter-cluster dissimilarity (ICD) (bottom half) of PRESTO against the most competitive methods for CIFAR10 and CIFAR100. Note that for all metrics, a higher number ensures better clustering quality. The best results are highlighted in green.

Method	Training time GPU Usage (MiB)			Test time GPU Usage (MiB)		
	CIFAR10	CIFAR100	STL	CIFAR10	CIFAR100	STL
GraSP	1104.23	1104.23	1413.99	814.26	814.26	1028.93
SNIP	863.66	863.66	1068.53	815.2	815.2	1027.84
KD	274.27	274.27	270.02	27.61	27.61	25.03
DGKD	274.27	274.28	270.02	27.61	27.61	25.03
PRESTO	58.12	84.05	81.55	12.07	12.07	12.07

Table 4. Maximum GPU memory used during training and test (in mebibytes, MiB) for PRESTO, two model pruning methods, *viz.*, GraSP (Wang et al., 2020a) and SNIP (Lee et al., 2019) and two knowledge distillation methods, *viz.*, KD (Hinton et al., 2015) and DGKD (Son et al., 2021) to reach 90% accuracy in all datasets. The best and second best results are highlighted in green and yellow respectively. Numbers in green and yellow indicate the best and the second best method.

plane, etc.). We report NMI and ARI under these ground truth cluster assignments. Intra-cluster similarity (ICS) is the ratio of the number of pairs that have the same cluster and same original image label (from CIFAR10 (100)) to the number of pairs that have the same cluster. Inter-cluster dissimilarity (ICD) is the ratio of the number of pairs that have different clusters and different image labels to the number of pairs that have different clusters. Note that in all these metrics, a higher value indicates better clustering quality. PRESTO outperforms all the baselines in almost all cases, except in CIFAR100 in terms of ICD.

Application on resource constrained learning. Given K, the number of partitions, PRESTO works with K lightweight models, which can be trained on different partitions of a dataset, instead of training a very large model on the entire dataset. To evaluate the efficacy of PRESTO on a resource constrained learning setup, we compare it against two model pruning methods, *viz.*, (1) GraSP (Wang et al., 2020a), (2) SNIP (Lee et al., 2019) and two methods for knowledge distillation, *viz.*, (3) KD (Hinton et al., 2015) (4) DGKD (Son et al., 2021) in terms of GPU memory used during training and inference. Pruning methods mask connections of a larger model that are relatively less important for training, whereas knowledge distillation methods start with a high capacity model (teacher model) and then design a lightweight model (student) that can mimic the output of the large model with fewer parameters. For a fair comparison, we maintain roughly similar (with 5% tolerance) number of parameters of each of the final lightweight models obtained by the student models given by the knowledge distillation methods; the total number parameters used in our approach is $\sum_{k=1}^{K} \dim(\boldsymbol{\theta}_k) + \dim(\phi)$.

First, we set K of PRESTO same as in Table 2. This gives the number of parameters of PRESTO as 122630 for $K = 6$, and 81754 for $K = 4$. In Table 4, we summarize the results for this setup. We make the following observations. (i) PRESTO consumes significantly lesser GPU memory than any other method across all datasets. During training, PRESTO consumes 78% lesser GPU memory than the knowledge distillation methods, *i.e.*, the second best method, on CIFAR10, whereas, for CIFAR100 and STL it consumes 70% lesser memory. During inference, it consumes 56% lesser GPU memory on CIFAR10 and CIFAR100, whereas, for STL it consumes 52% lesser memory. (ii) The pruning models take 3–4x memory as compared to the knowledge distillation models.

6. Conclusion

We present PRESTO, a novel framework of learning mixture models via data partitioning. Individual models specialising on a data partition, present a good alternative to learning one complex model and help achieve better generalisation. We present a joint discrete-continuous optimization algorithm to form the partitions with good approximation guarantees. With our experiments we demonstrate that PRESTO achieves best performance across different datasets when compared with several mixture-models and unsupervised partitioning methods. We also present that PRESTO achieves best accuracy *vs.* memory utilisation trade-off when compared with knowledge distillation and pruning methods. Future work involves extending our work into different distributed learning setups including federated learning. It would also be interesting to extend our work in a multi class learning setup, where different clusters are aligned with different tasks.

7. Acknowledgment

Abir De acknowledges a gift from Google Research and Inspire Faculty Fellowship. Rishabh Iyer would like to acknowledge support from NSF Grant Number IIS-2106937, a gift from Google Research, and the Adobe Data Science Research Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the NSF, Google Research, Adobe Data Science or the U.S. Government.

References

Achlioptas, D. and McSherry, F. On spectral learning of mixtures of distributions. In *International Conference on Computational Learning Theory*, pp. 458–469. Springer, 2005.

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pp. 244–253. PMLR, 2018.

Arthur, D. and Vassilvitskii, S. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

Bairi, R., Iyer, R., Ramakrishnan, G., and Bilmes, J. Summarization of multi-document topic hierarchies using submodular mixtures. In *ACL*, pp. 553–563, 2015.

Belkin, M. and Sinha, K. Polynomial learning of distribution families. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pp. 103–112. IEEE, 2010a.

Belkin, M. and Sinha, K. Polynomial learning of distribution families. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pp. 103–112. IEEE, 2010b.

Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Boutsidis, C., Drineas, P., and Magdon-Ismail, M. Near-optimal coresets for least-squares regression. *IEEE transactions on information theory*, 59(10):6880–6892, 2013.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Bubeck, S. and Sellke, M. A universal law of robustness via isoperimetry. *Advances in Neural Information Processing Systems*, 34, 2021.

Campbell, T. and Broderick, T. Bayesian coreset construction via greedy iterative geodesic ascent. In *International Conference on Machine Learning*, pp. 698–706, 2018.

Chen, Z., Deng, Y., Wu, Y., Gu, Q., and Li, Y. Towards understanding mixture of experts in deep learning. *arXiv preprint arXiv:2208.02813*, 2022.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Collobert, R., Bengio, S., and Bengio, Y. A parallel mixture of svms for very large scale problems. *Advances in Neural Information Processing Systems*, 14, 2001.

Dar, Y., Muthukumar, V., and Baraniuk, R. G. A farewell to the bias-variance tradeoff? an overview of the theory of overparameterized machine learning. *arXiv preprint arXiv:2109.02355*, 2021.

Dasgupta, S. Learning mixtures of gaussians. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pp. 634–644. IEEE, 1999.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dias, J. G., Vermunt, J. K., and Ramos, S. Mixture hidden markov models in finance research. In *Advances in data analysis, data handling and business intelligence*, pp. 451–459. Springer, 2009.

Durga, S., Iyer, R., Ramakrishnan, G., and De, A. Training data subset selection for regression with controlled generalization error. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9202–9212. PMLR, 18–24 Jul 2021.

Edmonds, J. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization—Eureka, You Shrink!*, pp. 11–26. Springer, 2003.

El Halabi, M. and Jegelka, S. Optimal approximation for unconstrained non-submodular minimization. In *International Conference on Machine Learning*, pp. 3961–3972. PMLR, 2020.

Faria, S. and Soromenho, G. Fitting mixtures of linear regressions. *Journal of Statistical Computation and Simulation*, 80(2):201–225, 2010.

Fu, Z. and Robles-Kelly, A. On mixtures of linear svms for nonlinear classification. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pp. 489–499. Springer, 2008.

Gatmiry, K. and Gomez-Rodriguez, M. Non-submodular function maximization subject to a matroid constraint, with applications. *arXiv preprint arXiv:1811.07863*, 2018.

Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., and Keutzer, K. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

Hashemi, A., Ghasemi, M., Vikalo, H., and Topcu, U. Submodular observation selection and information gathering for quadratic models. *arXiv preprint arXiv:1905.09919*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network, 2015.

Hopkins, S. B. and Li, J. Mixture models, robustness, and sum of squares proofs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1021–1034, 2018.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. URL https://arxiv.org/abs/1704.04861.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5mb model size, 2016. URL https://arxiv.org/abs/1602.07360.

Iyer, R. and Bilmes, J. Algorithms for approximate minimization of the difference between submodular functions, with applications. *arXiv preprint arXiv:1207.0560*, 2012.

Iyer, R. and Bilmes, J. Polyhedral aspects of submodularity, convexity and concavity. *arXiv preprint arXiv:1506.07329*, 2015.

Iyer, R., Jegelka, S., and Bilmes, J. Fast semidifferential-based submodular function optimization: Extended version. In *ICML*, 2013a.

Iyer, R., Jegelka, S., and Bilmes, J. Monotone closure of relaxed constraints in submodular optimization: Connections between minimization and maximization: Extended version. In *UAI*. Citeseer, 2014.

Iyer, R. K., Jegelka, S., and Bilmes, J. A. Curvature and optimal algorithms for learning and minimizing submodular functions. In *NeurIPS*, 2013b.

Jiang, Y., Wang, S., Valls, V., Ko, B. J., Lee, W.-H., Leung, K. K., and Tassiulas, L. Model pruning enables efficient federated learning on edge devices. *arXiv preprint arXiv:1909.12326*, 2019.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2): 181–214, 1994.

Kalai, A. T., Moitra, A., and Valiant, G. Efficiently learning mixtures of two gaussians. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pp. 553–562, 2010.

Kaushal, V., Iyer, R., Kothawade, S., Mahadev, R., Doctor, K., and Ramakrishnan, G. Learning from less data: A unified data subset selection and active learning framework for computer vision. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1289–1299. IEEE, 2019.

Killamsetty, K., Sivasubramanian, D., Mirzasoleiman, B., Ramakrishnan, G., De, A., and Iyer, R. Grad-match: A gradient matching based data subset selection for efficient learning. *arXiv preprint arXiv:2103.00123*, 2021a.

Killamsetty, K., Subramanian, D., Ramakrishnan, G., and Iyer, R. Glister: A generalization based data selection framework for efficient and robust learning. *In AAAI*, 2021b.

Kirchhoff, K. and Bilmes, J. Submodularity for data selection in machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 131–141, 2014.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Kwon, J., Qian, W., Caramanis, C., Chen, Y., and Davis, D. Global convergence of the em algorithm for mixtures of two component linear regression. In *Conference on Learning Theory*, pp. 2055–2110. PMLR, 2019.

Lee, N., Ajanthan, T., and Torr, P. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Li, B., Wu, B., Su, J., and Wang, G. Eagleeye: Fast subnet evaluation for efficient neural network pruning. In *European conference on computer vision*, pp. 639–654. Springer, 2020.

Liesenfeld, R. A generalized bivariate mixture model for stock price volatility and trading volume. *Journal of econometrics*, 104(1):141–178, 2001.

Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. *Advances in neural information processing systems*, 30, 2017.

Lin, T., Stich, S. U., Barba, L., Dmitriev, D., and Jaggi, M. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253*, 2020.

Liu, C., Zhu, L., and Belkin, M. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307*, 2020.

Liu, Y., Iyer, R., Kirchhoff, K., and Bilmes, J. Svitchboard ii and fisver i: High-quality limited-complexity corpora of conversational english speech. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Lloyd, S. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Lucic, M., Faulkner, M., Krause, A., and Feldman, D. Training gaussian mixture models at scale via coresets. *The Journal of Machine Learning Research*, 18(1): 5885–5909, 2017.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

MacQueen, J. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pp. 281–297, 1967.

Mirzasoleiman, B., Bilmes, J., and Leskovec, J. Coresets for data-efficient training of machine learning models, 2020a.

Mirzasoleiman, B., Cao, K., and Leskovec, J. Coresets for robust training of neural networks against noisy labels. *arXiv preprint arXiv:2011.07451*, 2020b.

Müllner, D. Modern hierarchical, agglomerative clustering algorithms. *CoRR*, abs/1109.2378, 2011. URL http://dblp.uni-trier.de/db/journals/corr/corr1109.html#abs-1109-2378.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1): 265–294, 1978.

Pace, R. K. and Barry, R. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

Pal, S., Mazumdar, A., Sen, R., and Ghosh, A. On learning mixture of linear regressions in the non-realizable setting. In *International Conference on Machine Learning*, pp. 17202–17220. PMLR, 2022.

Pan, W., Lin, J., and Le, C. T. A mixture model approach to detecting differentially expressed genes with microarray data. *Functional & integrative genomics*, 3(3):117–124, 2003.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rau, D. Sparsely-gated mixture-of-experts pytorch implementation, 2019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Sanjeev, A. and Kannan, R. Learning mixtures of arbitrary gaussians. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pp. 247–257, 2001.

Schrijver, A. et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Son, W., Na, J., Choi, J., and Hwang, W. Densely guided knowledge distillation using multiple teacher assistants, 2021.

Städler, N., Bühlmann, P., and Van De Geer, S. l1-penalization for mixture regression models. *Test*, 19(2): 209–256, 2010.

Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., and Lee, J. D. Sanity-checking pruning methods: Random tickets can win the jackpot. *arXiv preprint arXiv:2009.11094*, 2020.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=SkgsACVKPH.

Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., and Hu, X. Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 12273–12280, 2020b.

Wei, K., Iyer, R., and Bilmes, J. Fast multi-stage submodular maximization. In *International conference on machine learning*, pp. 1494–1502. PMLR, 2014a.

Wei, K., Liu, Y., Kirchhoff, K., and Bilmes, J. Unsupervised submodular subset selection for speech data. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4107–4111. IEEE, 2014b.

Xu, L. and Jordan, M. I. On convergence properties of the em algorithm for gaussian mixtures. *Neural computation*, 8(1):129–151, 1996.

Yi, X., Caramanis, C., and Sanghavi, S. Alternating minimization for mixed linear regression. In *International Conference on Machine Learning*, pp. 613–621. PMLR, 2014.

Zhang, H. and Vorobeychik, Y. Submodular optimization with routing constraints. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018. doi: 10.1109/CVPR.2018.00716.

Appendix
(Discrete-Continuous Optimization Framework for Simultaneous Clustering and Training in Mixture Models)

A. Proofs of the Technical Results in Section 3

A.1. Formal Discussion on Matroid

Definition 5. *A matroid is a combinatorial structure $M := (\mathbb{V}, \mathcal{I})$ defined on a ground set $\mathbb{V}$ and a family of independent sets $\mathcal{I} \subseteq 2^{\mathbb{V}}$, which satisfies two conditions.*

(1) If $\mathbb{S} \subseteq \mathbb{T}$ and $\mathbb{T} \in \mathcal{I}$, then $\mathbb{S} \in \mathcal{I}$.
(2) If $\mathbb{S} \in \mathcal{I}$ and $\mathbb{T} \in \mathcal{I}$ and $|\mathbb{T}| > |\mathbb{S}|$, then there exists a $e \in \mathbb{T} \backslash \mathbb{S}$ so that $\mathbb{S} \cup \{e\} \in \mathcal{I}$.

From the above definition, it is clear the all the maximal independent sets have same cardinality. A maximum independent set is called the base of the matroid.

A.2. Monotonicity and α-submodularity of G

Here we prove the claims of Theorem 2. We repeat the theorem for convenience

Theorem (2). *Given the set function $G(\mathbb{S})$ defined in Eq. (9) and the individual regularized loss functions ℓ introduced in Eq. (1), we define: $\varepsilon_{\min} = \min_{k,i} \mathit{Eigen}_{\min}[\nabla^2_{\boldsymbol{\theta}_k} \ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i)]$, $\ell_{\min,\min} = \min_{i\in D} \min_{\boldsymbol{\theta}_k} [\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i) + \lambda||\boldsymbol{\theta}_k||^2]$ and $\ell_{\min,\max} = \max_{i\in D} \min_{\boldsymbol{\theta}_k} [\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}_i), y_i) + \lambda||\boldsymbol{\theta}_k||^2]$. Then, we have the following results:*

(1) Monotonicity: The function $G(\mathbb{S})$ defined in Eq. (9) is monotone non-decreasing in $\mathbb{S}$.
(2) α-submodularity: If $\ell(h_{\boldsymbol{\theta}_k}(\boldsymbol{x}), y)$ is L-Lipschitz for all $k \in \{1, .., K\}$ and that the regularizing coefficient λ satisfies: $\lambda > -\varepsilon_{\min}$, function $G(\mathbb{S})$ is α-submodular with

$$\alpha \geq \alpha^* = \frac{\ell_{\min,\min}}{\ell_{\min,\min} + \frac{2L^2}{\lambda+0.5\varepsilon_{\min}} + \frac{\lambda L^2}{(\lambda+0.5\varepsilon_{\min})^2}} \tag{18}$$

(3) Generalized curvature: The generalized curvature $\kappa_G(\mathbb{S})$ for any set $\mathbb{S}$ is given by: $\kappa_G(\mathbb{S}) \leq \kappa^*_G = 1 - \ell_{\min,\min}/\ell_{\min,\max}$.

Proof. (1) (**Proof of monotonicity**) We define the regularized $g(\boldsymbol{\theta}_k, S) = \sum_{i\in S} \ell(h_{\boldsymbol{\theta}_k}(x_i), y_i) + \lambda\|\boldsymbol{\theta}_k\|^2$. First, we show that $g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}^*_k(S), S) > 0$. To that aim, note that,

$$\begin{aligned} & g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}^*_k(S), S) \\ & \quad = g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S) + g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S) - g(\boldsymbol{\theta}^*_k(S), S) \end{aligned} \tag{19}$$

Now, since $\boldsymbol{\theta}^*_k(S) = \text{argmin}_{\boldsymbol{\theta}_k} g(\boldsymbol{\theta}_k, S)$, we have $g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S) \geq g(\boldsymbol{\theta}^*_k(S), S)$. From Eq. (19), this leads to the following inequality:

$$g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}^*_k(S), S) \geq g(\boldsymbol{\theta}^*_k(S \cup \{j\}), \{j\}) \geq \underline{\ell}_{\min} > 0 \tag{20}$$

Now, if we include a new element (j, t) into $\mathbb{S}$, then it will only change the loss $g(\boldsymbol{\theta}^*_t(S), S)$ among all model components. Thus we have,

$$G(\mathbb{S} \cup \{(j, t)\}) - G(\mathbb{S}) \geq g(\boldsymbol{\theta}^*_t(S \cup \{j\}), (S \cup \{j\})) - g(\boldsymbol{\theta}^*_t(S), S) > 0$$

(2) (**Proof of α-submodularity**) First, we try to show α-submodularity of $g(\boldsymbol{\theta}^*_k(S), S)$. Hence, we first bound the following ratio:

$$\frac{g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}^*_k(S), S)}{g(\boldsymbol{\theta}^*_k(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}^*_k(T), T)} \tag{21}$$

Eq. (20) directly provides bound on the numerator of this ratio:

$$g(\boldsymbol{\theta}^*_k(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}^*_k(S), S) \geq \underline{\ell}_{\min} \tag{22}$$

Next, we try to upper bound $g(\boldsymbol{\theta}^*_k(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}^*_k(T), T)$. We note that:

$$\begin{aligned} & g(\boldsymbol{\theta}^*_k(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}^*_k(T), T) \\ & \quad = g(\boldsymbol{\theta}^*_k(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}^*_k(T), T \cup \{j\}) + g(\boldsymbol{\theta}^*_k(T), T \cup \{j\}) - g(\boldsymbol{\theta}^*_k(T), T). \end{aligned} \tag{23}$$

Now, since $\boldsymbol{\theta}_k^*(T \cup \{j\}) = \operatorname{argmin}_{\boldsymbol{\theta}_k} g(\boldsymbol{\theta}_k, T \cup \{j\})$, we have:

$$g(\boldsymbol{\theta}_k^*(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}_k^*(T), T \cup \{j\}) \leq 0 \tag{24}$$

Next, we note that $g(\boldsymbol{\theta}_k^*(T), T \cup \{j\}) - g(\boldsymbol{\theta}_k^*(T), T) = g(\boldsymbol{\theta}_k^*(T), \{j\})$ which gives us:

$$g(\boldsymbol{\theta}_k^*(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}_k^*(T), T) = g(\boldsymbol{\theta}_k^*(T), \{j\}) \tag{25}$$

This leads us to bound the ratio in Eq. (21) as follows:

$$\begin{aligned} &\frac{g(\boldsymbol{\theta}_k^*(S \cup \{j\}), S \cup \{j\}) - g(\boldsymbol{\theta}_k^*(S), S)}{g(\boldsymbol{\theta}_k^*(T \cup \{j\}), T \cup \{j\}) - g(\boldsymbol{\theta}_k^*(T), T)} \qquad (26)\\ &\geq \frac{g(\boldsymbol{\theta}_k^*(S \cup \{j\}), \{j\})}{g(\boldsymbol{\theta}_k^*(T), \{j\})} \\ &\geq \frac{g(\boldsymbol{\theta}_k^*(S \cup \{j\}), \{j\})}{g(\boldsymbol{\theta}_k^*(S \cup \{j\}), \{j\}) + g(\boldsymbol{\theta}_k^*(T), \{j\}) - g(\boldsymbol{\theta}_k^*(S \cup \{j\}), \{j\})}. \qquad (27) \end{aligned}$$

To bound the denominator, we note that:

$$g(\boldsymbol{\theta}_k^*(T), \{j\}) - g(\boldsymbol{\theta}_k^*(S \cup \{j\}), \{j\}) \leq \ell(h_{\boldsymbol{\theta}_k^*(T)}(x_j), y_j) - \ell(h_{\boldsymbol{\theta}_k^*(S \cup \{j\})}(x_j), y_j) + \lambda \|\boldsymbol{\theta}_k^*(T)\|^2$$

Now, we note that

$$\begin{aligned} 0 \geq g(\boldsymbol{\theta}_k^*(T), T) - g(\mathbf{0}, T) &= \lambda \|\boldsymbol{\theta}_k^*(T)\|^2 + \left(\nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(x_i), y_i)\right)_{\boldsymbol{\theta}=\mathbf{0}}^{\top} \boldsymbol{\theta}_k^*(T) \\ &\quad + \frac{1}{2} [\boldsymbol{\theta}_k^*(T) \nabla^2 \ell(h_{\boldsymbol{\theta}}(x_i), y_i)]_{\boldsymbol{\theta} \in (0, \boldsymbol{\theta}_k^*(T))} \boldsymbol{\theta}_k^*(T) \\ \implies \frac{1}{2} \|\boldsymbol{\theta}_k^*(T)\|^2 \varepsilon_{\min} + \lambda \|\boldsymbol{\theta}_k\|^2 - L \|\boldsymbol{\theta}_k\| < 0 \qquad (28) \end{aligned}$$

Thus, we have: $\|\boldsymbol{\theta}_k\| \leq \frac{L}{\lambda + \frac{\varepsilon_{\min}}{2}}$. Putting this in Eq. (28), we have:

$$\begin{aligned} g(\boldsymbol{\theta}_k^*(T), \{j\}) - g(\boldsymbol{\theta}_k^*(S \cup \{j\}), \{j\}) &\leq L \|\boldsymbol{\theta}_k^*(T) - \boldsymbol{\theta}_k^*(S \cup \{j\})\| + \lambda \|\boldsymbol{\theta}_k^*(T)\|^2 \qquad (29) \\ &\leq \frac{2L^2}{\lambda + \frac{\varepsilon_{\min}}{2}} + \frac{\lambda L^2}{(\lambda + \frac{\varepsilon_{\min}}{2})^2} \qquad (30) \end{aligned}$$

Thus, replacing the above quantity in Eq. (27), we have:

$$\alpha \geq \alpha^* = \frac{\ell_{\min,\min}}{\ell_{\min,\min} + \frac{2L^2}{\lambda + 0.5\varepsilon_{\min}} + \frac{\lambda L^2}{(\lambda + 0.5\varepsilon_{\min})^2}} \tag{31}$$

(3) (**Proof of the bound on Generalized curvature**) From the definition of curvature,

$$1 - \kappa_g(S) = \min_{a \in V} \frac{g(\boldsymbol{\theta}_k^*(S), S) - g(\boldsymbol{\theta}_k^*(S \setminus \{a\}), S \setminus \{a\})}{g(\boldsymbol{\theta}_k^*(\{a\}), \{a\})} \tag{32}$$

From Eq. (20),

$$g(\boldsymbol{\theta}_k^*(S), S) - g(\boldsymbol{\theta}_k^*(S \setminus \{a\}), S \setminus \{a\}) \geq \ell_{\min,\min} \tag{33}$$

$$g(\boldsymbol{\theta}_k^*(\{a\}), \{a\}) \leq \max_a \left[\lambda \|\boldsymbol{\theta}_k\|^2 + l(f_{\boldsymbol{\theta}_k}(x_a), y_a)\right] = \ell_{\min,\max} \tag{34}$$

Thus,

$$\kappa_g(S) \leq 1 - \frac{\ell_{\min,\min}}{\ell_{\min,\max}} \tag{35}$$

If we add an element $a = (j, t)$ to $\mathbb{S}$ only $g(\boldsymbol{\theta}_t, S)$ will be changed among the component models. Thus,

$$\begin{aligned} \kappa_G(\mathbb{S}) &= 1 - \min_{a \in V} \frac{G(\mathbb{S}) - G(\mathbb{S} \setminus \{a\})}{G(a)} \qquad (36) \\ &= 1 - \min_{a \in V} \frac{g(\boldsymbol{\theta}_t^*(S), S) - g(\boldsymbol{\theta}_t^*(S \setminus \{a\}), S \setminus \{a\})}{g(\boldsymbol{\theta}_t^*(\{a\}), \{a\})} \\ &\geq 1 - \frac{\ell_{\min,\min}}{\ell_{\min,\max}} \end{aligned}$$

□

A.3. Approximation Guarantees

We next prove the approximation bound for Theorem 4.

Theorem (4). *If the function G is α_G-submodular and has a curvature κ_G, Algorithm 1 obtains an approximation guarantee of $\frac{|D|}{\alpha_G(1+(|D|-1)(1-\kappa_G)\alpha_G)} \leq \frac{1}{\alpha_G^2(1-\kappa_G)}$ assuming there exists a perfect training oracle in Lines (6,12,14).*

Proof. From the definition of α-submodularity, note that $\alpha_G G(\mathbb{S}) \leq \sum_{i \in \mathbb{S}} G(i)$. Next, we can obtain the following inequality for any $k \in \mathbb{S}$ using weak submodularity:

$$G(\mathbb{S}) - G(k) \geq \alpha_G \sum_{j \in \mathbb{S} \backslash k} (G(j|\mathbb{S} \backslash j) \tag{37}$$

We can add this up for all $k \in \mathbb{S}$ and obtain:

$$\begin{aligned} |\mathbb{S}|G(\mathbb{S}) - \sum_{k \in \mathbb{S}} G(k) &\geq \alpha_G \sum_{k \in \mathbb{S}} \sum_{j \in \mathbb{S} \backslash k} (G(j|\mathbb{S} \backslash j) \\ &\geq \alpha_G(|\mathbb{S}| - 1) \sum_{k \in \mathbb{S}} G(k|\mathbb{S} \backslash k) \end{aligned} \tag{38}$$

Finally, from the definition of curvature, note that $G(k|\mathbb{S} \backslash k) \leq (1 - \kappa_f)G(k)$. Combining all this together, we obtain:

$$|\mathbb{S}|G(\mathbb{S}) \geq (1 + \alpha_G(1 - \kappa_f)(|\mathbb{S}| - 1)) \sum_{j \in \mathbb{S}} G(j) \tag{39}$$

which implies:

$$\sum_{j \in \mathbb{S}} G(j) \leq \frac{|\mathbb{S}|}{1 + \alpha_G(1 - \kappa_G)(|S| - 1)} G(\mathbb{S}) \tag{40}$$

Combining this with the fact that $\alpha_G G(\mathbb{S}) \leq \sum_{i \in \mathbb{S}} G(i)$, we obtain that:

$$G(\mathbb{S}) \leq \frac{1}{\alpha_G} \sum_{i \in \mathbb{S}} G(i) \leq \frac{|\mathbb{S}|}{\alpha_G(1 + \alpha_G(1 - \kappa_G)(|S| - 1))} G(\mathbb{S}) \tag{41}$$

Note that $|\mathbb{S}|/(1 + \alpha_G(1 - \kappa_G)(|\mathbb{S}| - 1) \leq 1/\alpha_G^2(1 - \kappa_G)$ so we just use this factor in the approximation bound. The approximation guarantee then follows from some simple observations. In particular, given an approximation

$$m^G(\mathbb{S}) = \frac{1}{\alpha_G} \sum_{i \in \mathbb{S}} G(i) \tag{42}$$

which satisfies $G(\mathbb{S}) \leq m^G(\mathbb{S}) \leq \beta_G G(\mathbb{S})$, we claim that optimizing m^G essentially gives a β_G approximation factor. To prove this, let $\mathbb{S}^*$ be the optimal subset, and $\hat{\mathbb{S}}$ be the subset obtained after optimizing m^G. The following chain of inequalities holds:

$$G(\hat{\mathbb{S}}) \leq m^G(\hat{\mathbb{S}}) \leq m^G(\mathbb{S}^*) \leq \beta_G G(\mathbb{S}^*) \tag{43}$$

This shows that $\hat{\mathbb{S}}$ is a β_G approximation of $\mathbb{S}^*$. Finally, note that this is just the first iteration of PRESTO, and with subsequent iterations, PRESTO is guaranteed to reduce the objective value since we only proceed if there is a reduction in objective value. □

B. Comparison with EM Algorithm

General EM algorithm usually considers a prior and computes the membership probability of each datapoint at each iteration. It makes assumption about the generative mechanism of the data as well as the prior about the cluster probabilities. Such a probabilistic approach naturally leads one to develop EM algorithm.

On the other hand, our method makes no assumption about the data as well as cluster membership. We do not make any probabilistic assumption which naturally led us to develop a set optimization problem— we do not make use of any "continuous" or "soft" scores on cluster membership. Thus, our method is an MM algorithm which is an iterative set optimization algorithm, which is functionally very different from EM algorithm.

C. Difference with Mixture of Linear Regression

All our experiments consider classification tasks. Therefore, we modify Learn-MLR (Pal et al., 2022) to a mixture of classifiers. However, we note that the optimization *problem formulation* of the mixture of linear regression (or classification) can be viewed as special case of the *problem formulation* of our method.

In most cases, the existing methods make assumptions about the generative mechanism of the cluster membership (e.g., some definite prior) and resort to an algorithm which makes an involved use of that assumption and is significantly tailored to a particular task (mixture of regression or classification).

In contrast, our framework operates on a generic framework and does not make any specific assumption about the data or the cluster membership. Moreover, the algorithm is very different from what is used by the usual mixture model learning algorithms.

D. Additional Details about Experimental Setup

D.1. Dataset details

We experiment with three real world classification datasets in our experiments,

- **CIFAR10** (Krizhevsky, 2009) has images of 10 different real-world objects.
- **CIFAR100** (Krizhevsky, 2009) has images of 100 different real-world objects.
- **STL** (Coates et al., 2011) images of 10 different real-world objects.

For all datasets we extract embeddings from the sixth layer of a pretrained ResNet18. We rotate each of the examples by 30 degrees and post-processing, pick them in such a way that half of the examples are rotated in the train and test sets for each dataset.

D.2. Model Architecture used for our method, the Unsupervised Partitioning methods and the Mixture Models

In case of our method, the unsupervised partitioning methods and the mixture models (all methods in Table 2 and PRESTO in Table 4), we use a relatively light model. This model is exactly same in terms of architecture design to the final four layers of ResNet18. This model design is chosen because we use extracted embeddings for training. The model reduces the convolution filters at various junctures. The filter at fourth last layer is changed from 256 to 8, filter at third last layer is changed from 512 to 16. We train an additional classifier. This classifier is a single layer fully connected feed forward network that aids in predicting the final class for each instance.

D.3. Implementation Details for PRESTO

Number of partitions K. Number of partitions (K) is found for various datasets by performing an analysis of classification accuracy v/s K in the validation. Specifically, for our model, we set $K = 6$ for CIFAR10 and $K = 4$ for each of CIFAR100 and STL using cross-validation. Similarly, we obtain the values of $K = 5$ for CIFAR10 and CIFAR100, and $K = 4$ for STL for the clustering baselines.

Initialization. PRESTO is initialised using one of the clustering techniques depending on the dataset. We also try to ensure that the clusters are of approximately the same size. We impose $\frac{|N|}{|K|}$ size constraints for each of the bin with some leeway. The leeway granted for every dataset is also a tunable hyperparameter.

Final selection of (i, k) from the matrix $\boldsymbol{M}$. The values of the matrix $\boldsymbol{M}$ computed in line 19 of Algorithm 1 are supplemented with euclidean distance ($\boldsymbol{D}$) of a point to the existing bin center. The final criterion for binning becomes $\boldsymbol{M} + \epsilon\boldsymbol{D}$. Here, we have $\epsilon = 6e - 3$.

Optimizer. We use an Adam optimizer with the Cross-Entropy Loss to train the PRESTO models and the classifier. The learning rate is set to 0.01. We set the value of the regularizer $\lambda = 1e - 4$. In Algorithm 1, Train$(\cdot)$ train the models on the respective data partitions for 10 epochs. We set `Iterations` = 30. The additional classifier π_ϕ is also trained for 10 epochs.

Table 2 mentions various baselines. Details about each baseline is covered in the following points.

D.4. Implementation details about the Unsupervised Partitioning methods and Mixture Models

Kmeans++. We use the scikit-learn (Pedregosa et al., 2011) implementation of Kmeans++ with the following parameters: The number of clusters is set as K. A Kmeans++ initialization is used for selecting the initial centroids. We keep the number of times the algorithm will be run with different centroid seeds as n_init = 10. The final result is the best output of n_init consecutive runs in terms of inertia. The algorithm runs either a max of 300 iterations or convergence, whichever occurs earlier. The convergence criterion is defined as the Frobenius norm of the difference in the cluster centers of two consecutive iterations becoming less than $1e-4$.

Agglomerative. It is a bottom-up hierarchical clustering approach. We use the scikit-learn (Pedregosa et al., 2011) implementation of Agglomerative We use the *Euclidean* distance metric to calculate distance between data points. Initially each data point is in its own cluster. The clusters are subsequently merged based on a linkage criterion. We use the ward linkage criterion, which minimizes the sum of square differences within each cluster. The number of clusters is set as K.

GMM. We use the scikit-learn (Pedregosa et al., 2011) implementation of GMM with the number of components $= K$. The GMM weights are initialized according to a KMeans initialization. The convergence threshold for the EM iterations is set as $1e-3$.

Learn-MLR. We adapt the original paper (Pal et al., 2022) for the purpose of classification. The initial partitioning is random and the we do not have a leeway when the data is re-partitioned. The ϵ parameter which gives a contribution to the euclidean distance (D) is set to 0. An Adam optimizer is used with Cross-Entropy loss. The learning rate is set to 0.01 and the regularizer $\lambda = 1e-4$.

Mixture of Experts. We use the Sparse-MoE layer described in (Shazeer et al., 2017) for classification. The expert networks have the architecture described in Section D.2. We adapt the (Rau, 2019) implementation for this purpose. We use Noisy Top-K Gating as described in (Shazeer et al., 2017), which balances the number of training examples each expert receives. We set the number of experts used for each batch element, k = 4.

D.5. Implementation Details for Resource-Constrained Experiments

We evaluate PRESTO in a resource-constrained learning setting by comparing with model pruning and knowledge distillation methods. The specifications for them are mentioned in the following points.

SNIP (Lee et al., 2019). SNIP is a pruning at initialization method, that given a large reference network prunes network connections to a desired sparsity level (i.e. number of non-zero weights). Pruning is done *prior to training*, in a data-dependent way based on the loss function at a variance scaling initialization, to create a sparse subnetwork which is then used for inference. We adapt the implementation from (Su et al., 2020) to run experiments using SNIP. We train a large reference model, with the same architecture as the PRESTO models described in Section D.2. The reference model has the convolution filter size at the fourth last layer as 512 and at the third last layer as 1024. The models are trained on using an Adam optimizer and the Cross-Entropy loss with a learning rate of 0.1. The reference models are trained for 150 epochs, and the subsequent pruned models are trained for a total of 20 epochs till convergence.

GraSP (Wang et al., 2020a). GraSP is a pruning at initialization method similar to SNIP and also learns a smaller subnetwork given a reference network, which can subsequently be trained independently. The gradient norm *after pruning* is the pruning criterion in GraSP, and those weights whose removal will result in least decrease in the gradient norm after pruning are pruned. We adapt the implementation from (Su et al., 2020) to run experiments using SNIP. We train a large reference model, with the same architecture as the PRESTO models described in Section D.2. The reference model has the convolution filter size at the fourth last layer as 512 and at the third last layer as 1024. The models are trained on using an Adam optimizer and the Cross-Entropy loss with a learning rate of 0.1. The reference models are trained for 150 epochs, and the subsequent pruned models are trained for a total of 20 epochs till convergence.

KD (Hinton et al., 2015). We train two models, a teacher model and a student model. The architecture of these models is same as the PRESTO models described in Section D.2. The teacher model is heavier in size, the convolution filter size at the fourth last layer is 256 and at the third last layer is 512. On the other hand, the lighter student model has convolution filter size set to 24 and 32. The models are trained on using an Adam optimizer and the Cross-Entropy loss. The KD Temperature hyperparameter is set to 5. The teacher model is trained and the loss of the student model is modified to include a contribution from the teacher model. The multiplicative hyperparameter controlling the contribution of the teacher model is set to 0.9.

DGKD (Son et al., 2021). This method requires us to train 3 models, a teacher model, a TA model and a student model. The teacher model is the heaviest, having the convolution filters set to 256 and 512. The TA model is slightly lighter with filter size of 64 and 128. The student model is the lightest with filter size of 24 and 32. The temperature hyperparameter is 5 and the factor controlling the contribution of the teacher and TA model is set at 0.9. We train the teacher and the TA models independently. As in KD, the loss function of the student model is tweaked to consider a contribution from the teacher and the TA models.

Machine configuration. We performed our experiments on a computer system with Ubuntu 16.04.6 LTS, an i-7 with 8 cores CPU and a total RAM of 528 GBs. The system had a single Titan RTX GPU which was employed in our experiments.

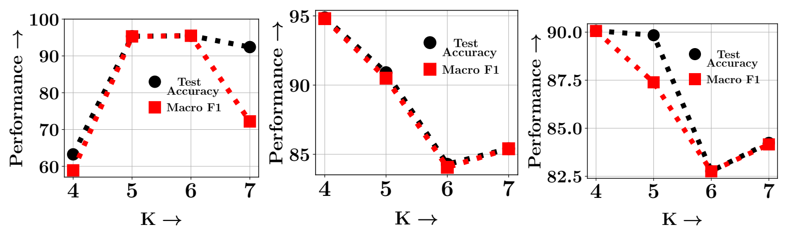
E. Additional Experiments

NMI and ARI for synthetic data. Here, we measure the performance of PRESTO against competitive partitioning and mixture model techniques on clustering metrics such as Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI). These results are presented in Table 5.

	NMI	ARI
Kmeans++	0.26	0.09
Agglomerative	0.71	0.79
MoE	0.28	0.21
Learn-MLR	0.59	0.63
PRESTO	0.86	0.92

Table 5. Comparison of Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI) of PRESTO against the unsupervised partitioning and mixture model methods for synthetic data. The best results are highlighted in green.

Accuracy vs K. Here, we aim to assess the impact of number of partitions (K) on the accuracy. K is varied from 4 to 7. Figure 6 shows the variation of K with the test accuracy and the Macro-F1 score. Note that the Macro-F1 score is the harmonic mean of the precision and recall.



(a) CIFAR10 (b) CIFAR100 (c) STL

Figure 6. Accuracy vs. the number of partitions K

At the outset, we expect each cluster to contain similar samples. In principle, a cluster, that has the larger set of similar examples, would be most effective for training the underlying model. Thus, when we change K, three factors weigh differently.

Factor 1 : Semantic similarity within a cluster. As the number of clusters K increases, each cluster tends to become more and more homogeneous and therefore, the semantic similarity across instances within the cluster increases. This renders the model easier to train, making the model accuracy to increase with K.

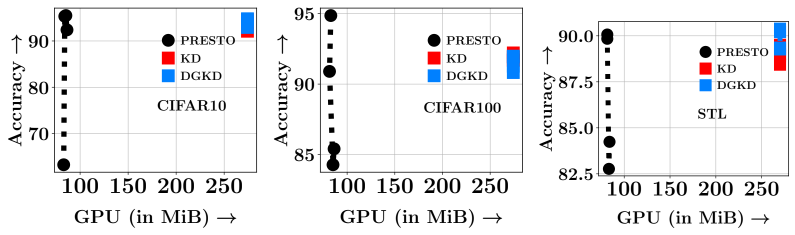
Factor 2 : Reduction in the number of examples per cluster. As the number of clusters K increases, each cluster contains less instances on average. This causes overfitting making the accuracy to decrease as K increases.

Factor 3 : Accuracy of π_ϕ. As K increases, the accuracy of the additional classifier (π_ϕ) becomes much lower since it needs to distinguish between multiple clusters at test time.

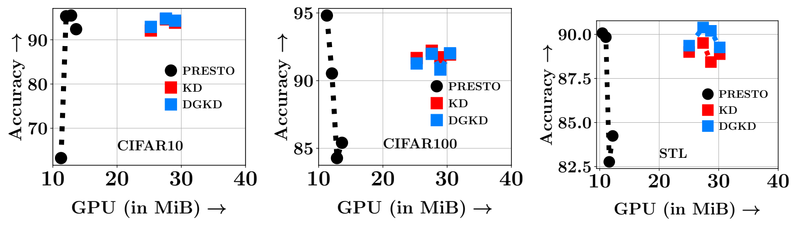
In CIFAR10, if we use very small K ($K \leq 4$), we observe that several dissimilar objects are clustered together and Factor 1 starts to weigh in more than Factors 2 and 3. Hence, accuracy is low for these values of K. However, CIFAR-10 has more redundancy or less diversity since we have only 10 objects and 6000 examples per object. Therefore, a slightly larger value of K ($K = 5, 6$) is enough for a large number of semantically similar items to get clustered together. Here, we observe clusters of the right groups of classes, for e.g., cats and dogs, cars and trucks, etc.; and, we obtain high accuracy.

The diversity of images in CIFAR100 is very high— we have 100 object classes and only 600 examples per object. Hence, at $K = 5, 6$ unlike CIFAR10, the average self similarity across inter-cluster instances does not change much from $K = 4$. Thus Factor 1 does not play any role in this regime and Factors 2 and 3 weigh in. Consequently, we have highest accuracy at $K = 4$ since at $K = 4$, it has a higher number of instances per cluster than $K = 5, 6$ and the additional classifier π_ϕ has to distinguish between smaller numbers of clusters (4 vs 5,6), while having approximately the same diversity. Now if we increase K substantially, as we described before, Factors 2 and 3 come in again, the number of instances per cluster becomes too small and the final classification task by the additional classifier also becomes difficult.

Accuracy vs GPU usage. Here we vary the number of parameters p of PRESTO (through K), which gives us $p \in [81754, 143068]$ on all datasets. Then we probe the variation in accuracy *vs.* maximum GPU memory used during both training and inference. In Figure 7, we summarize the results for PRESTO and KD and DGKD— the two most resource efficient methods from Table 4. We observe that PRESTO consumes significantly lower memory for diverse model size during both training and inference.



(a) Training



(b) Inference

Figure 7. Trade off between accuracy and maximum GPU memory for KD, DGKD and PRESTO during training (panel (a)) and inference (panel (b)).

F. Time Complexity Analysis

Although PRESTO performs data partitioning while simultaneously learning a set of mixture of models, the partitioning time (lines 19-24 in Algorithm 1) is negligible as the bulk of the time is devoted to training (lines 10-18 in Algorithm 1). Table 8 presents an analysis of the per-iteration training time for PRESTO against the relevant baselines. The table reports two numbers for PRESTO and the clustering baselines. The first number corresponds to the training time required to serially train all the bin-wise models and the external classifier for 1 epoch. The number in the bracket corresponds to the training time that would be required if we are able to parallelize the individual bin-wise training. PRESTO has a time complexity of $O(\#\text{Iterations} \cdot |D| \cdot |K|)$, we can remove the $|K|$ factor by parallelizing the cluster-wise training procedure and have a better compute efficiency. This number is required for a fair comparison to the MoE baseline which uses an end to end training cycle and does not have the sequential bin-wise training that PRESTO follows.

	Training time per iteration (s)		
Method	CIFAR10	CIFAR100	STL
PRESTO	70.35 (10.05)	49.31 (9.87)	29.00 (5.80)
Kmeans++	65.19 (9.31)	41.92 (8.38)	26.2 (5.24)
Agglomerative	75.22 (10.74)	54.44 (10.91)	34.45 (6.89)
GMM	58.28 (8.33)	39.77 (7.95)	24.19 (4.84)
MoE	7.32	9.73	4.73
Learn-MLR	44	52.58	22.74

Table 8. Comparison of training time per iteration of PRESTO against GMM (Bishop & Nasrabadi, 2006), Kmeans++ (Arthur & Vassilvitskii, 2006), MoE (Shazeer et al., 2017), Agglomerative (Müllner, 2011) and Learn-MLR (Pal et al., 2022). Note that the table reports two numbers for PRESTO and the clustering baselines. PRESTO uses a sequential bin-wise training procedure which is reflected in the first number, whereas the numbers in the bracket signifies the time taken to train if we had parallelised the bin-wise training procedure.