Recovery of sparse linear classifiers from mixture of responses

Venkata Gandikota
University of Massachusetts Amherst, MA
Amherst, MA 01003
gandikota.venkata@gmail.com

Arya Mazumdar
University of Massachusetts Amherst, MA
Amherst, MA 01003
arya@cs.umass.edu

Soumyabrata Pal
University of Massachusetts Amherst, MA
Amherst, MA 01003
soumyabratap@umass.edu

Abstract

In the problem of learning a *mixture of linear classifiers*, the aim is to learn a collection of hyperplanes from a sequence of binary responses. Each response is a result of querying with a vector and indicates the side of a randomly chosen hyperplane from the collection the query vector belong to. This model provides a rich representation of heterogeneous data with categorical labels and has only been studied in some special settings. We look at a hitherto unstudied problem of query complexity upper bound of recovering all the hyperplanes, especially for the case when the hyperplanes are sparse. This setting is a natural generalization of the extreme quantization problem known as 1-bit compressed sensing. Suppose we have a set of ℓ unknown k-sparse vectors. We can query the set with another vector $\boldsymbol{a}$, to obtain the sign of the inner product of $\boldsymbol{a}$ and a randomly chosen vector from the ℓ-set. How many queries are sufficient to identify all the ℓ unknown vectors? This question is significantly more challenging than both the basic 1-bit compressed sensing problem (i.e., $\ell = 1$ case) and the analogous regression problem (where the value instead of the sign is provided). We provide rigorous query complexity results (with efficient algorithms) for this problem.

1 Introduction

One of the first and most basic tasks of machine learning is to train a binary linear classifier. Given a set of explanatory variables (features) and the binary responses (labels), the objective of this task is to find the hyperplane in the space of features that best separates the variables according to their responses. In this paper, we consider a natural generalization of this problem and model a classification task as a mixture of ℓ components. In this generalization, each response is stochastically generated by picking a hyperplane uniformly from the set of ℓ unknown hyperplanes, and then returning the side of that hyperplane the feature vector lies. The goal is to learn all of these ℓ hyperplanes as accurately as possible, using the least number of responses.

This can be termed as a mixture of binary linear classifiers [33]. Similar mixture of simple machine learning models have been around for at least the last thirty years [9] with mixture of linear regression models being the most studied ones [8, 19, 23, 30, 32, 34, 35, 37]. Models of this type are pretty good function approximators [4, 21] and have numerous applications in modeling heterogeneous settings such as machine translation [25], behavioral health [11], medicine [5], object recognition [28] etc. While algorithms for learning the parameters of mixture of linear regressions are solidly grounded (such as tensor decomposition based learning algorithms of [7]), in many of

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

the above applications the labels are discrete categorical data, and therefore a mixture of classifiers is a better model than mixture of regressions. To the best of our knowledge, [33] first rigorously studied a mixture of linear classifiers and provided polynomial time algorithm to approximate the subspace spanned by the component classifiers (hyperplane-normals) as well as a prediction algorithm that given a feature and label, correctly predicts the component used. In this paper we study a related but different problem: the sample complexity of learning *all* the component hyperplanes. Our model also differs from [33] where the component responses are 'smoothened out'. Here the term *sample complexity* is used with a slightly generalized meaning than traditional learning theory - as we explain next, and then switch to the term *query complexity* instead.

Recent works on mixture of sparse linear regressions concentrate on an active query based setting [24, 26, 36], where one is allowed to design a sample point and query an oracle with that point. The oracle then randomly chooses one of the component models and returns the answer according to that model. In this paper we adapt exactly this setting for binary classifiers. We assume while queried with a point (vector), an oracle randomly chooses one of the ℓ binary classifiers, and then returns an answer according to what was chosen. For the most of this paper we concentrate on recovering 'sparse' linear classifiers, which implies that each of the classifiers uses only few of the explanatory variables. This setting is in spirit of the well-studied *1-bit compressed sensing* (1bCS) problem.

1-bit compressed sensing. In 1-bit compressed sensing, linear measurements of a sparse vector are quantized to only 1 bit, e.g. indicating whether the measurement outcome is positive or not, and the task is to recover the vector up to a prescribed Euclidean error with a minimum number of measurements. An overwhelming majority of the literature focuses on the nonadaptive setting for the problem [1, 2, 14, 17, 20, 27]. Also, a large portion of the literature concentrates on learning only the support of the sparse vector from the 1-bit measurements [1, 17].

It was shown in [20] that $O(\frac{k}{\epsilon}\log(\frac{n}{\epsilon}))$ Gaussian queries[1] suffice to approximately (to the Euclidean precision ϵ) recover an unknown k-sparse vector $\boldsymbol{\beta}$ using 1-bit measurements. Given the labels of the query vectors, one recovers $\boldsymbol{\beta}$ by finding a k-sparse vector that is consistent with all the labels. If we consider enough queries, then the obtained solution is guaranteed to be close to the actual underlying vector. [1] studied a two-step recovery process, where in the first step, they use queries corresponding to the rows of a special matrix, known as *Robust Union Free Family (RUFF)*, to recover the support of the unknown vector $\boldsymbol{\beta}$ and then use this support information to approximately recover $\boldsymbol{\beta}$ using an additional $\tilde{O}(\frac{k}{\epsilon})$ Gaussian queries. Although the recovery algorithm works in two steps, the queries are nonadaptive.

Mixture of sparse linear classifiers. The main technical difficulty that arises in recovering multiple sparse hyperplanes using 1-bit measurements (labels) is to *align* the responses of different queries concerning a fixed unknown hyperplane. To understand this better, let us consider the case when $\ell = 2$ (see Figure 1). Let $\boldsymbol{\beta^1}, \boldsymbol{\beta^2}$ be two unknown k-sparse vectors corresponding to two sparse linear classifiers. On each query, the oracle samples a $\boldsymbol{\beta^i}$, for $i \in \{1, 2\}$, uniformly at random and returns the binary label corresponding to it ($+$ or $-$). One can query the oracle repeatedly with the same query vector to ensure a response from both the classifiers with overwhelmingly high probability.

For any query vector if the responses corresponding to the two classifiers are the same (*i.e.*, $(+,+)$ or $(-,-)$), then we do not gain any information separating the two classifiers. We might still be able to reconstruct some sparse hyperplanes, but the recovery guarantees of such an algorithm will be poor. On the other hand, if both the responses are different (*i.e.*, $(+,-)$), then we do not know which labels correspond to a particular classifier. For example, if the responses are $(+,-)$ and $(+,-)$ for two distinct query vectors, then we do not know if the 'plusses' correspond to the same classifier. This issue of alignment makes the problem challenging. Such alignment issues are less damning in the case of mixture of linear regressions even in the presence of noise [24, 26, 36] since we can utilize the magnitude information of the inner products (labels) to our advantage.

One of the challenges in our study is to recover the supports of the two unknown vectors. Consider the case when $\mathsf{supp}(\boldsymbol{\beta^1}) \neq \mathsf{supp}(\boldsymbol{\beta^2})$, where $\mathsf{supp}(\boldsymbol{v})$ denotes the support of the vector $\boldsymbol{v} \in \mathbb{R}^n$. In this case, we show that using an RUFF in combination with another similar class of union-free family (UFF), we can deduce the supports of both $\boldsymbol{\beta^1}$ and $\boldsymbol{\beta^2}$. Wielding known constructions of

[1] all coordinates of the query vector are sampled independently from the standard Gaussian distribution

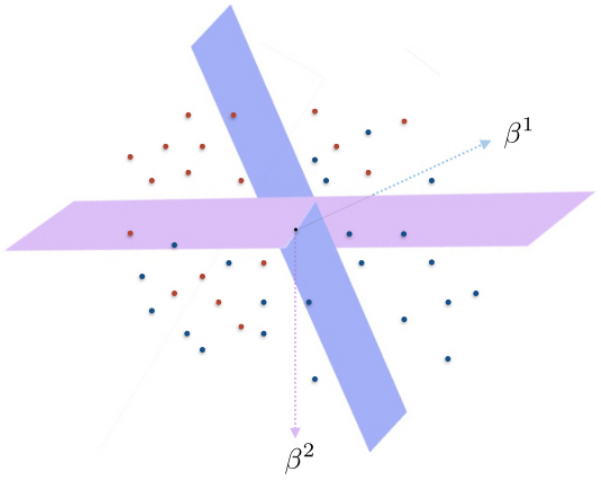

Figure 1: Recover the two lines given red triangles and blue dots. How many such points do we require in order to recover the two lines?

such UFFs from literature, we can recover the supports of both k-sparse vectors using $O(k^3 \log^2 n)$ queries. Once we obtain the supports, we use an additional $O(\frac{k}{\epsilon} \log nk)$ Gaussian queries (with a slight modification) to approximately recover the individual vectors.

We then extend this two-step process (using more general classes of UFFs) to recover a mixture of ℓ different sparse vectors under the assumption that the support of no vector is contained in the union of supports of the remaining ones (Assumption 1). The assumption implies that if the sparse vectors are arranged as columns of a matrix, then the matrix contains the identity matrix as a permutation of the rows. This separability condition appears before in [3, 12, 31] in the context of nonnegative integer matrix factorization, which is a key tool that we will subsequently use to prove our results. To quote [3] in the context of matrix factorization, "an approximate separability condition is regarded as a fairly benign assumption and is believed to hold in many practical contexts in machine learning." We believe this observation holds for our context as well (each classifier uses some unique feature).

We show that with this support separability condition, $\tilde{O}(\ell^6 k^3)$ queries suffice for support recovery of ℓ different k-sparse vectors. Further, using $\tilde{O}((\ell^3 k/\epsilon))$ queries, we can recover each of the $\boldsymbol{\beta^i}$'s, for $i \in \{1, \ldots, \ell\}$ up to ϵ precision (see Theorem 1 and Theorem 2).

The two-stage procedure described above, can be made completely non-adaptive using queries from union free families (see Theorem 3).

Furthermore, for $\ell = 2$, we see that the support condition (Assumption 1) is not necessary. We can approximately recover the two unknown vectors provided a) they are not extremely sparse and, b) each $\boldsymbol{\beta^i} \in \delta\mathbb{Z}^n$ for some $\delta > 0$. To prove this, we borrow the tools from [2] who give guarantees for 1-bit compressed sensing using sub-Gaussian vectors. In particular, we use queries with independent Bernoulli coordinates which are sub-Gaussian. These discrete random queries (as opposed to continuous Gaussians) along with condition (b), enables us to align the labels corresponding to the two unknown vectors. (see Theorem 4 for more details). Note that condition (a) is due to the result by [2] and is necessary for recovery using sub-Gaussian queries and (b) is a mild assumption on the precision of the unknown vectors, which was also necessary [24, 36] for learning the mixture of sparse linear regressions.

Technical take-away. As stated above, the main technical hurdle in the paper lies in the support recovery problem for the sparse vectors. We do it in a few (non-adaptive) steps. First, we try to design queries that will lead us to estimate the size of the intersections of supports for every pair of sparse vectors. If we can design such a set of queries, then using a nonnegative integer matrix factorization techniques we can estimate all the supports. Indeed, this is where the separability assumption comes in handy. Further, because of the separability assumption, it is also possible to design queries from which we can simulate the response of every unknown vector with a random Gaussian query

and tag it. This allows us to recover and *cluster* the responses of every unknown vector to a set of random Gaussian queries from which we can approximately recover all the unknown vectors.

To estimate the size of the intersections of supports for the vector-pairs, we rely on combinatorial designs (and related set-systems), such as pairwise independent cover free families [15]. While some such union-free families have been used to estimate the support in 1-bit compressed sensing before [1], the use of pairwise independent sets to *untangle multiple sparse vectors* is new and has almost nothing to do with the recovery of sparse vectors itself.

We leave the problem of designing a query scheme that works for any general ℓ without any assumptions as an open problem. Lack of Assumption 1 seems to be a fundamental barrier to support recovery as it ensures that a sparse vector will never be in the span of the others. However, a formal statement of this effect still eludes us. For large ℓ, finding out the dependence of query complexity on ℓ is also a natural question. Overall, this study leads to an interesting set of questions that are technically demanding as well as quite relevant to practical modeling of heterogeneous data that are ubiquitous in applications. For instance, in recommendation systems, where the goal is to identify the factors governing the preferences of individual members of a group via crowdsourcing while preserving the anonymity of their responses.

Organization. The rest of this paper is organized as follows. In the next section, we formally define the problem statement followed by a list of our contributions in Section 3. The notations and the necessary background on various families of sets are presented in Section 4. We prove Theorem 1 in Section 5. The proofs of all the other theorems and the helper lemmas required for the proof of Theorem 1 are deferred to the supplementary material. Moreover, we demonstrate the capability of our algorithms to learn movie genre preferences of two unknown users using the MovieLens [18] dataset. The experimental details are included in Section E of the supplementary material.

2 Problem Statement

Let $\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots, \boldsymbol{\beta}^\ell \in \mathbb{R}^n$ be a set of ℓ unknown k-sparse vectors. Each $\boldsymbol{\beta^i}$ defines a linear classifier that assigns a label from $\{-1, 0, 1\}$ to every vector in $\boldsymbol{v} \in \mathbb{R}^n$ according to $\mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta} \rangle)$, where the sign function $\mathsf{sign} : \mathbb{R} \to \{-1, 0, 1\}$, is defined as,

$$\mathsf{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}.$$

Remark 1. *It is also possible to handle binary output from the classifier instead of ternary as has been studied in this paper. See the full version [16] for more details.*

As described above, our work focuses on recovering the unknown classifiers in the *query model* that was used in [36, 24] to study the mixtures of sparse linear regressions. In the query model, we assume the existence of an oracle $\mathcal{O}$ which when queried with a vector $\boldsymbol{v} \in \mathbb{R}^n$, samples one of the classifiers $\boldsymbol{\beta} \in \{\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots, \boldsymbol{\beta}^\ell\}$ uniformly at random and returns the label of $\boldsymbol{v}$ assigned by the sampled classifier $\boldsymbol{\beta}$. The goal of approximate recovery is to reconstruct each of the unknown classifiers using small number of oracle queries. The problem can be formalized as follows:

Problem 1 (ϵ-recovery)**.** *Given $\epsilon > 0$, and query access to oracle $\mathcal{O}$, find k-sparse vectors $\{\hat{\boldsymbol{\beta}}^{\mathbf{1}}, \hat{\boldsymbol{\beta}}^{\mathbf{2}}, \ldots, \hat{\boldsymbol{\beta}}^{\boldsymbol{\ell}}\}$ such that for some permutation $\sigma : [\ell] \to [\ell]$*

$$\left\| \frac{\boldsymbol{\beta^i}}{\|\boldsymbol{\beta^i}\|_2} - \frac{\hat{\boldsymbol{\beta}}^{\boldsymbol{\sigma(i)}}}{\|\hat{\boldsymbol{\beta}}^{\boldsymbol{\sigma(i)}}\|_2} \right\|_2 \leq \epsilon \quad \forall\, i \in [\ell].$$

Since from the classification labels, we lose the magnitude information of the unknown vectors, we assume each $\boldsymbol{\beta^i}$ and the estimates $\hat{\boldsymbol{\beta}}^{\boldsymbol{i}}$ to have a unit norm.

Similar to the literature on one-bit compressed sensing, one of our proposed solutions employs a two-stage algorithm to recover the unknown vectors. In the first stage the algorithm recovers the support of every vector, and then in the second stage, approximately recovers the vectors using the support information.

For any vector $\boldsymbol{v} \in \mathbb{R}^n$, let $\mathsf{supp}(\boldsymbol{v}) := \{i \in [n] \mid \boldsymbol{v}_i \neq 0\}$ denote the support of $\boldsymbol{v}$. The problem of support recovery is then defined as follows:

Problem 2 (Support Recovery)**.** *Given query access to oracle $\mathcal{O}$, construct $\{\hat{\boldsymbol{\beta}}^{\mathbf{1}}, \hat{\boldsymbol{\beta}}^{\mathbf{2}}, \ldots, \hat{\boldsymbol{\beta}}^{\boldsymbol{\ell}}\}$ such that for some permutation $\sigma : [\ell] \to [\ell]$*

$$\mathsf{supp}(\hat{\boldsymbol{\beta}}^{\boldsymbol{\sigma(i)}}) = \mathsf{supp}(\boldsymbol{\beta}^{\boldsymbol{i}}) \quad \forall\, i \in [\ell]$$

For both these problems, we primarily focus on minimizing the query complexity of the problem, *i.e.*, minimizing the number of queries that suffice to approximately recover all the sparse unknown vectors or their supports. However, all the algorithms proposed in this work also run in time $O(\mathsf{poly}(q))$, where q is the query complexity of the algorithm.

3 Our contributions

In order to present our first set of results, we need certain assumption regarding the separability of supports of the unknown vectors. In particular, we want each component of the mixture to have a unique identifying coordinate. More formally, it can be stated as follows:

Assumption 1. *For every $i \in [\ell]$, $\mathsf{supp}(\boldsymbol{\beta}^{\boldsymbol{i}}) \not\subseteq \bigcup_{j:j\neq i} \mathsf{supp}(\boldsymbol{\beta}^{\boldsymbol{j}})$, i.e. the support of any unknown vector is not contained in the union of the support of the other unknown vectors.*

Two-stage algorithm: First, we propose a two-stage algorithm for ϵ-recovery of the unknown vectors. In the first stage of the algorithm, we recover the support of the unknown vectors (Theorem 1), followed by ϵ-recovery using the deduced supports (Theorem 2) in the second stage. Each stage in itself is non-adaptive, *i.e.*, the queries do not depend on the responses of previously made queries.

Theorem 1. *Let $\{\boldsymbol{\beta}^{\mathbf{1}}, \ldots, \boldsymbol{\beta}^{\boldsymbol{\ell}}\}$ be a set of ℓ unknown k-sparse vectors in $\mathbb{R}^n$ that satisfy Assumption 1. There exists an algorithm to recover the support of every unknown vector $\{\boldsymbol{\beta}^{\boldsymbol{i}}\}_{i\in[\ell]}$ with probability at least $1 - O(1/n^2)$, using $O(\ell^6 k^3 \log^2 n)$ non-adaptive queries to oracle $\mathcal{O}$.*

Now using this support information, we can approximately recover the unknown vectors using an additional $\tilde{O}(\ell^3 k)$ non-adaptive queries.

Theorem 2. *Let $\{\boldsymbol{\beta}^{\mathbf{1}}, \ldots, \boldsymbol{\beta}^{\boldsymbol{\ell}}\}$ be a set of ℓ unknown k-sparse vectors in $\mathbb{R}^n$ that satisfy Assumption 1. There exists a two-stage algorithm that uses $O\Big(\ell^6 k^3 \log^2 n + (\ell^3 k/\epsilon)\log(nk/\epsilon)\log(k/\epsilon)\Big)$ oracle queries for the ϵ-recovery of all the unknown vectors with probability at least $1 - O(1/n)$.*

Remark 2. *We note that for the two-stage recovery algorithm to be efficient, we require the magnitude of non-zero entries of the unknown vectors to be non-negligible (at least $1/exp(n)$). This assumption however is not required to bound the query complexity of the algorithm which is the main focus of this work.*

Completely non-adaptive algorithm: Next, we show that the entire ϵ-recovery algorithm can be made non-adaptive (single-stage) at the cost of increased query complexity.

Theorem 3. *Let $\{\boldsymbol{\beta}^{\mathbf{1}}, \ldots, \boldsymbol{\beta}^{\boldsymbol{\ell}}\}$ be a set of ℓ unknown k-sparse vectors in $\mathbb{R}^n$ that satisfy Assumption 1. There exists an algorithm that uses $O\Big((\ell^{\ell+3}k^{\ell+2}/\epsilon)\log n \log(n/\epsilon)\log(k/\epsilon)\Big)$ non-adaptive oracle queries for the ϵ-recovery of all the unknown vectors with probability at least $1 - O(1/n)$.*

Note that even though the one-stage algorithm uses many more queries than the two-stage algorithm, a completely non-adaptive is highly parallelizable as one can choose all the query vectors in advance. Also, in the $\ell = O(1)$ regime, the query complexity is comparable to its two-stage analogue.

While we mainly focus on minimizing the query complexity, all the algorithms proposed in this work run in $\mathsf{poly}(n)$ time assuming every oracle query takes $\mathsf{poly}(n)$ time and $\ell = o(\log n)$.

Non-adaptive algorithm for $\ell = 2$ without Assumption 1: For $\ell = 2$, we do not need the separability condition (Assumption 1) required earlier for support recovery. Even for ϵ-recovery, instead of Assumption 1, we just need a mild assumption on the precision δ, and the sparsity of the unknown vectors. In particular, we propose an algorithm for the ϵ-recovery of the two unknown vectors using $\tilde{O}(k^3 + k/\epsilon)$ queries provided the unknown vectors have some finite precision and are not extremely sparse.

Assumption 2. *For $\boldsymbol{\beta} \in \{\boldsymbol{\beta}^{\mathbf{1}}, \boldsymbol{\beta}^{\mathbf{2}}\}$, $\|\boldsymbol{\beta}\|_\infty = o(1)$.*

Assumption 2 ensures that we can safely invoke the result of [2] who use the exact same assumption in the context of 1-bit compressed sensing using sub-Gaussian queries.

Theorem 4. *Let $\boldsymbol{\beta^1}, \boldsymbol{\beta^2}$ be two k-sparse vectors in $\mathbb{R}^n$ that satisfy Assumption 2. Let $\delta > 0$ be the largest real such that $\boldsymbol{\beta^1}, \boldsymbol{\beta^2} \in \delta\mathbb{Z}^n$. There exists an algorithm that uses $O(k^3 \log^2 n + (k^2/\epsilon^4\delta^2)\log^2(n/k\delta^2))$ (adaptive) oracle queries for the ϵ-recovery of $\boldsymbol{\beta^1}, \boldsymbol{\beta^2}$ with probability at least $1 - O(1/n)$.*

Moreover, if $\mathsf{supp}(\boldsymbol{\beta^1}) \neq \mathsf{supp}(\boldsymbol{\beta^2})$, then there exists a two-stage algorithm for the ϵ-recovery of the two vectors using only $O(k^3 \log^2 n + (k/\epsilon)\log(nk/\epsilon)\log(k/\epsilon))$ non-adaptive oracle queries.

Also, the ϵ-recovery algorithm proposed for Theorem 4 runs in time $\mathsf{poly}(n, 1/\delta)$.

No sparsity constraint: We can infact avoid the sparsity constraint altogether for the case of $\ell = 2$. Since in this setting, we consider the support of both unknown vectors to include all coordinates, we do not need a support recovery stage. We then get a single stage and therefore completely non-adaptive algorithm for ϵ-recovery of the two unknown vectors.

Corollary 1. *Let $\boldsymbol{\beta^1}, \boldsymbol{\beta^2}$ be two unknown vectors in $\mathbb{R}^n$ that satisfy Assumption 2. Let $\delta > 0$ be the largest real such that $\boldsymbol{\beta^1}, \boldsymbol{\beta^2} \in \delta\mathbb{Z}^n$. There exists an algorithm that uses $O((n^2/\epsilon^4\delta^2)\log(1/\delta))$ non-adaptive oracle queries for the ϵ-recovery of $\boldsymbol{\beta^1}, \boldsymbol{\beta^2}$ with probability at least $1 - O(1/n)$.*

4 Preliminaries

Let $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For any vector $\boldsymbol{v} \in \mathbb{R}^n$, $\mathsf{supp}(\boldsymbol{v})$ denotes the support and $\boldsymbol{v}_i$ denote the i^{th} entry (coordinate) of the vector $\boldsymbol{v}$. We will use $\boldsymbol{e_i}$ to denote a vector which has 1 only in the i^{th} position and is 0 everywhere else. We will use the notation $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ to denote the inner product between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of the same dimension. For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, let $\boldsymbol{A_i} \in \mathbb{R}^n$ be its i^{th} column and $\boldsymbol{A}[j]$ denote its j^{th} row. and let $\boldsymbol{A}_{i,j}$ be the (i,j)-th entry of $\boldsymbol{A}$. We will denote by Inf a *very large positive number*. Also, let $\mathcal{N}(0,1)$ denote the standard normal distribution. We will use $\mathcal{P}_n$ to denote a the set of all $n \times n$ permutation matrices, *i.e.*, the set of all $n \times n$ binary matrices that are obtained by permuting the rows of an $n \times n$ identity matrix (denoted by $\boldsymbol{I}_n$). Let $\mathsf{round} : \mathbb{R} \to \mathbb{Z}$ denote a function that returns the closest integer to a given real input.

Let us further introduce a few definitions that will be used throughout the paper.

Definition 3. *For a particular entry $i \in [n]$, define $\mathcal{S}(i)$ to be the set of all unknown vectors whose i^{th} entry is non-zero.*

$$\mathcal{S}(i) := \{\boldsymbol{\beta^j}, j \in [\ell] \mid \boldsymbol{\beta^j}_i \neq 0\}$$

Definition 4. *For a particular query vector $\boldsymbol{v}$, define $\mathsf{poscount}(\boldsymbol{v}), \mathsf{negcount}(\boldsymbol{v})$ and $\mathsf{nzcount}(\boldsymbol{v})$ to be the number of unknown vectors that assign a positive, negative, and non-zero label to $\boldsymbol{v}$ respectively.*

$$\begin{aligned}
\mathsf{poscount}(\boldsymbol{v}) &:= |\{\boldsymbol{\beta^j} \mid \mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta^j} \rangle) = +1, j \in [\ell]\}| \\
\mathsf{negcount}(\boldsymbol{v}) &:= |\{\boldsymbol{\beta^j} \mid \mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta^j} \rangle) = -1, j \in [\ell]\}| \\
\mathsf{nzcount}(\boldsymbol{v}) &:= \mathsf{poscount}(\boldsymbol{v}) + \mathsf{negcount}(\boldsymbol{v}) \\
&= |\{\boldsymbol{\beta^j} \mid \mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta^j} \rangle) \neq 0, j \in [\ell]\}|.
\end{aligned}$$

Definition 5 (Gaussian query)**.** *A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called a Gaussian query vector if each entry $\boldsymbol{v}_i$ of $\boldsymbol{v}$ is sampled independently from the standard Normal distribution, $\mathcal{N}(0,1)$.*

4.1 Estimating the counts

In this section we show how to accurately estimate each of the counts *i.e.*, $\mathsf{poscount}(\boldsymbol{v})$, $\mathsf{negcount}(\boldsymbol{v})$ and $\mathsf{nzcount}(\boldsymbol{v})$ with respect to any query vector $\boldsymbol{v}$, with high probability (see Algorithm 1).

The idea is to simply query the oracle with the same query vector repeatedly and estimate the counts empirically using the responses of the oracle. Let T denote the number of times a fixed query vector $\boldsymbol{v}$ is repeatedly queried. We refer to this quantity as the *batchsize*. We now show that the empirical estimates of each of the counts equals the real counts with high probability.

Lemma 6. *For any query vector $\boldsymbol{v}$, Algorithm 1 with batchsize T provides the correct estimates of $\mathsf{poscount}(\boldsymbol{v})$, $\mathsf{negcount}(\boldsymbol{v})$ and $\mathsf{nzcount}(\boldsymbol{v})$ with probability at least $1 - 8e^{-T/2\ell^2}$.*

Algorithm 1 QUERY$(\boldsymbol{v}, T)$

Require: Query access to oracle $\mathcal{O}$.
1: **for** $i = 1, 2, \ldots, T$ **do**
2: Query the oracle with vector $\boldsymbol{v}$ and obtain response $y^i \in \{-1, 0, +1\}$.
3: **end for**
4: Let $\hat{\mathsf{pos}} := \mathsf{round}\Big(\frac{\ell \sum_i \mathbb{1}[y^i = +1]}{T}\Big)$
5: Let $\hat{\mathsf{neg}} := \mathsf{round}\Big(\frac{\ell \sum_i \mathbb{1}[y^i = -1]}{T}\Big)$
6: Let $\hat{\mathsf{nz}} := \hat{\mathsf{pos}} + \hat{\mathsf{neg}}$.
7: Return $\hat{\mathsf{pos}}, \hat{\mathsf{neg}}, \hat{\mathsf{nz}}$.

4.2 Family of sets

We now review literature on some important families of sets called *union free families* [1] and *cover free families* [22] that found applications in cryptography, group testing and 1-bit compressed sensing. These special families of sets are used crucially in this work to design the query vectors for the support recovery and the ϵ-recovery algorithms.

Definition 7 (Robust Union Free Family $(d, t, \alpha) - \mathsf{RUFF}$)**.** *Let d, t be integers and $0 \leq \alpha \leq 1$. A family of sets, $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ where each $\mathcal{H}_i \subseteq [m]$ and $|\mathcal{H}| = d$ is a (d, t, α)-RUFF if for any set of t indices $T \subset [n], |T| = t$, and any index $j \notin T$,*

$$\left| \mathcal{H}_j \setminus \left(\bigcup_{i \in T} \mathcal{H}_i \right) \right| > (1 - \alpha)d.$$

We refer to n as the size of the family of sets, and m to be the alphabet over which the sets are defined. RUFFs were studied earlier in the context of support recovery of 1bCS [1], and a simple randomized construction of (d, t, α)-RUFF with $m = O(t^2 \log n)$ was proposed by De Wolf [10].

Lemma 8. *[1, 10] Given n, t and $\alpha > 0$, there exists an (d, t, α)-RUFF, $\mathcal{F}$ with $m = O\big((t^2 \log n)/\alpha^2\big)$ and $d = O((t \log n)/\alpha)$.*

RUFF is a generalization of the family of sets known as the Union Free Familes (UFF) - which are essentially $(d, t, 1)$-RUFF. In this work, we require yet another generalization of UFF known as Cover Free Families (CFF) that are also sometimes referred to as superimposed codes [13].

Definition 9 (Cover Free Family (r, t)-CFF)**.** *A family of sets $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ where each $\mathcal{H}_i \subseteq [m]$ is an (r, t)-CFF if for any pair of disjoint sets of indices $T_1, T_2 \subset [n]$ such that $|T_1| = r, |T_2| = t, T_1 \cap T_2 = \emptyset$,*

$$\left| \bigcap_{i \in T_1} \mathcal{H}_i \setminus \bigcup_{i \in T_2} \mathcal{H}_i \right| > 0.$$

Several constructions and bounds on existence of CFFs are known in literature. We state the following lemma regarding the existence of CFF which can be found in [29, 15]. We also include a proof in the supplementary material for the sake of completeness.

Lemma 10. *For any given integers r, t, there exists an (r, t)-CFF, $\mathcal{F}$ of size n with $m = O(t^{r+1} \log n)$.*

Note that $(1, t)$-CFF is exactly a UFF. The $(2, t)$-CFF is of particular interest to us and will henceforth be referred to as the *pairwise union free family* (PUFF). From Lemma 10 we know the existence of PUFF of size n with $m = O(t^3 \log n)$.

Corollary 2. *For any given integer t, there exists a $(2, t)$-CFF, $\mathcal{F}$ of size n with $m = O(t^3 \log n)$.*

5 Support Recovery (Proof of Theorem 1)

In this section, we present an efficient algorithm to recover the support of all the ℓ unknown vectors using a small number of oracle queries. The proof of Theorem 1 follows from the guarantees of Algorithm 2. The proofs of the helper lemmas used in this theorem are deferred to Section B in the supplementary material.

Consider the support matrix $\boldsymbol{X} \in \{0,1\}^{n \times \ell}$ where the i-th column is the indicator of $\mathsf{supp}(\boldsymbol{\beta^i})$. The goal in Theorem 1 is to recover this unknown matrix $\boldsymbol{X}$ (up to permutations of columns) using a small number of oracle queries. In Algorithm 2, we recover $\boldsymbol{X}$ from $\boldsymbol{XX^T}$, where the latter can be constructed using only the estimates of nzcount for some specially designed queries. The unknown matrix $\boldsymbol{X}$ is recovered from the constructed $\boldsymbol{XX^T}$ by rank factorization with binary constraints. The factorization is efficient and also turns out to be unique (up to permutations of columns) because of the separability assumption (Assumption 1) on the supports of the unknown vectors.

The main challenge lies in constructing the matrix $\boldsymbol{XX^T}$ using just the oracle queries. Recall that for any $i \in [n]$, $\mathcal{S}(i)$ denotes the set of unknown vectors that have a non-zero entry in the i-th coordinate. Note that the i-th row of $\boldsymbol{X}$, for any $i \in [n]$, is essentially the indicator of $\mathcal{S}(i)$. From this observation, it follows that the (i,j)-th entry of $\boldsymbol{XX^T}$ is captured by the term $|\mathcal{S}(i) \cap \mathcal{S}(j)|$.

We observe that the quantity $|\mathcal{S}(i) \cap \mathcal{S}(j)|$ can be computed from oracle queries in two steps. First, we use query vectors from an RUFF with appropriate parameters to compute $|\mathcal{S}(i)|$ for every $i \in [n]$ (see Algorithm 3). Then, using queries from a PUFF (Algorithm 4) to obtain $|\mathcal{S}(i) \cup \mathcal{S}(j)|$ for every pair (i,j). To state it formally,

Lemma 11. *There exists an algorithm to compute $|\mathcal{S}(i)|$ for each $i \in [n]$ with probability at least $1 - O\left(1/n^2\right)$ using $O(\ell^4 k^2 \log(\ell k n) \log n)$ oracle queries.*

Lemma 12. *There exists an algorithm to compute $|\mathcal{S}(i) \cup \mathcal{S}(j)|$ for every pair (i,j) with probability at least $1 - O\left(1/n^2\right)$ using $O(\ell^6 k^3 \log(\ell k n) \log n)$ oracle queries.*

By combining these two steps, we can obtain the (i,j)-th entry of XX^T as $|\mathcal{S}(i) \cap \mathcal{S}(j)| = |\mathcal{S}(i)| + |\mathcal{S}(j)| - |\mathcal{S}(i) \cup \mathcal{S}(j)|$. Equipped with these two Lemmas, we now prove the guarantees of Algorithm 2 that completes the proof of Theorem 1.

Algorithm 2 RECOVER–SUPPORT

Require: Query access to oracle $\mathcal{O}$.
Require: Assumption 1 to be true.
1: Estimate $|S(i)|$ for every $i \in [n]$ using Algorithm 3.
2: Estimate $|S(i) \cup S(j)|$ for every $i, j \in [n]$ using Algorithm 4.
3: **for** every pair $(i,j) \in [n] \times [n]$ **do**
4: Set $Z_{i,j} = |\mathcal{S}(i)| + |\mathcal{S}(j)| - |\mathcal{S}(i) \cup \mathcal{S}(j)|$
5: **end for**
6: Return $\hat{\boldsymbol{X}} \in \{0,1\}^{n \times \ell}$ such that $\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}^T = \boldsymbol{Z}$.

Proof of Theorem 1. Using Algorithm 3 and Algorithm 4, we compute $|\mathcal{S}(i) \cap \mathcal{S}(j)| = |\mathcal{S}(i)| + |\mathcal{S}(i)| - |\mathcal{S}(i) \cup \mathcal{S}(j)|$ for every pair $(i,j) \in [n] \times [n]$, and hence populate the entries of $Z = XX^T$. To obtain X from Z, we perform a rank factorization of Z with a binary constraint on the factors. We now show that Assumption 1 ensures that this factorization is unique up to permutations.

Suppose $\boldsymbol{Y} \neq \boldsymbol{X}$ is a binary matrix such that $\boldsymbol{YY}^T = \boldsymbol{XX}^T$. Therefore, there exists a rotation matrix $\boldsymbol{R} \in \mathbb{R}^{\ell \times \ell}$ such that $\boldsymbol{Y} = \boldsymbol{XR}$. From Assumption 1 we know that there exists an $\ell \times \ell$ submatrix $\tilde{\boldsymbol{X}}$ of $\boldsymbol{X}$ that is a permutation matrix. For the corresponding submatrix $\tilde{\boldsymbol{Y}}$ of $\boldsymbol{Y}$ (obtained by choosing the same subset of rows), it must hold that

$$\tilde{\boldsymbol{Y}}\tilde{\boldsymbol{Y}}^T = \tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T = \boldsymbol{I}$$

where $\boldsymbol{I}$ is the $\ell \times \ell$ identity matrix. Since $\boldsymbol{Y}$ has binary entries, $\tilde{\boldsymbol{Y}}$ must be a permutation matrix as well. This implies that $\boldsymbol{R}$ is a permutation matrix and a constrained rank factorization can recover $\boldsymbol{X}$ up to a permutation of columns. Therefore, Algorithm 2 successfully recovers the support of all the ℓ unknown vectors.

The total number of queries needed by Algorithm 2 is the sum total of the queries needed by Algorithm 3 and Algorithm 4 which is $O(\ell^6 k^3 \log(\ell k n) \log(n))$.

Moreover, since Algorithm 3 and Algorithm 4 each succeed with probability at least $1 - O(1/n^2)$. By a union bound, it follows that Algorithm 2 succeeds with probability at least $1 - O(1/n^2)$. $\square$

6 Conclusion and Open Questions

In this work, we initiated the study of recovering a mixture of ℓ different sparse linear classifiers given query access to an oracle. The problem generalizes the well-studied work on 1-bit compressed sensing ($\ell = 1$) and also complements the literature on learning mixtures of sparse linear regression in a similar query model.

Our results for $\ell > 2$, rely on the assumption that the supports of all the unknown vectors are separable. This separability assumption translates to each classifier using a unique feature not being used in others, which happen often in practice. The approximate recovery problem without the separability assumption is non-trivial even for $\ell = 2$ case, for which we provide guarantees with much milder assumptions on the precision of the classifiers. We leave the problem of support recovery and ϵ-recovery without any assumptions as an open problem.

We primarily focus on providing upper bounds on the query complexity of the support recovery and approximate recovery of the unknown vectors. However, proving optimality results for any such recovery is an interesting open direction. It is known that even to recover the support of a single k-sparse vector in the 1-bit compressed sensing setting, about $\Omega(k^2 \log n)$ queries are required. This corresponds to $\ell = 1$ case, and the lower bound holds trivially for any general ℓ as well. However, a nontrivial lower bound on the query complexity characterizing the asymptotic dependence on ℓ, the number of components, will be of interest.

Broader Impact

This paper is a theoretical study that brings together two seemingly disjoint but equally impactful fields of sparse recovery and mixture models: the first having numerous applications in signal processing while the second being the main statistical model for clustering. Given that, this work belongs to the foundational area of data science and enhances our understanding of some basic theoretical questions. We feel the methodology developed in this paper is instructive, and exemplifies the use of several combinatorial objects and techniques in signal recovery and classification, that are hitherto underused. Therefore we foresee the technical content of this paper to form good teaching material in foundational data science and signal processing courses. The content of this paper can raise interest of students or young researchers in discrete mathematics to applications areas and problems of signal processing and machine learning.

While primarily of theoretical interest, the results of the paper can be immediately applicable to some real-life scenarios and be useful in recommendation systems, one of the major drivers of data science research. In particular, if in any case of feedback/rating from users of a service there is ambiguity about the source of the feedback, our framework can be used. This is also applicable to crowdsourcing applications.

Acknowledgements: This research is supported in part by NSF CCF 1909046 and NSF 1934846.

References

[1] Jayadev Acharya, Arnab Bhattacharyya, and Pritish Kamath. Improved bounds for universal one-bit compressive sensing. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2353–2357. IEEE, 2017.

[2] Albert Ai, Alex Lapanowski, Yaniv Plan, and Roman Vershynin. One-bit compressed sensing with non-gaussian measurements. *Linear Algebra and its Applications*, 441:222–239, 2014.

[3] Sanjeev Arora, Rong Ge, Ravi Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization—provably. *SIAM Journal on Computing*, 45(4):1582–1611, 2016.

[4] Christopher M Bishop. Latent variable models. In *Learning in graphical models*, pages 371–403. Springer, 1998.

[5] Ekin Blackwell, Carlos F Mendes De Leon, and Gregory E Miller. Applying mixed regression models to the analysis of repeated-measures data in psychosomatic medicine. *Psychosomatic medicine*, 68(6):870–878, 2006.

[6] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[7] Arun Tejasvi Chaganty and Percy Liang. Spectral experts for estimating mixtures of linear regressions. In *International Conference on Machine Learning*, pages 1040–1048, 2013.

[8] Yudong Chen, Xinyang Yi, and Constantine Caramanis. A convex formulation for mixed regression with two components: Minimax optimal rates. In *Conference on Learning Theory*, pages 560–604, 2014.

[9] Richard D De Veaux. Mixtures of linear regressions. *Computational Statistics & Data Analysis*, 8(3):227–245, 1989.

[10] Ronald De Wolf. Efficient data structures from unionfree families of sets, 2012.

[11] Partha Deb and Ann M Holmes. Estimates of use and costs of behavioural health care: a comparison of standard and finite mixture models. *Health economics*, 9(6):475–489, 2000.

[12] David Donoho and Victoria Stodden. When does non-negative matrix factorization give a correct decomposition into parts? In *Advances in neural information processing systems*, pages 1141–1148, 2004.

[13] Arkadii G D'yachkov, IV Vorobyev, NA Polyanskii, and V Yu Shchukin. Bounds on the rate of superimposed codes. In *2014 IEEE International Symposium on Information Theory*, pages 2341–2345. IEEE, 2014.

[14] Larkin Flodin, Venkata Gandikota, and Arya Mazumdar. Superset technique for approximate recovery in one-bit compressed sensing. In *Advances in Neural Information Processing Systems*, pages 10387–10396, 2019.

[15] Zoltán Füredi. On r-cover-free families. *Journal of Combinatorial Theory, Series A*, 73(1):172–173, 1996.

[16] Venkata Gandikota, Arya Mazumdar, and Soumyabrata Pal. Recovery of sparse linear classifiers from mixture of responses. *arXiv preprint arXiv:2010.12087*, 2020.

[17] Sivakant Gopi, Praneeth Netrapalli, Prateek Jain, and Aditya Nori. One-bit compressed sensing: Provable support and vector recovery. In *International Conference on Machine Learning*, pages 154–162, 2013.

[18] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[19] Mian Huang, Runze Li, and Shaoli Wang. Nonparametric mixture of regression models. *Journal of the American Statistical Association*, 108(503):929–941, 2013.

[20] Laurent Jacques, Jason N Laska, Petros T Boufounos, and Richard G Baraniuk. Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *IEEE Transactions on Information Theory*, 59(4):2082–2102, 2013.

[21] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[22] William Kautz and Roy Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10(4):363–377, 1964.

[23] Abbas Khalili and Jiahua Chen. Variable selection in finite mixture of regression models. *Journal of the american Statistical association*, 102(479):1025–1038, 2007.

[24] Akshay Krishnamurthy, Arya Mazumdar, Andrew McGregor, and Soumyabrata Pal. Sample complexity of learning mixture of sparse linear regressions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[25] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 761–768. Association for Computational Linguistics, 2006.

[26] Arya Mazumdar and Soumyabrata Pal. Recovery of sparse signals from a mixture of linear samples. In *International Conference on Machine Learning (ICML)*, 2020.

[27] Yaniv Plan and Roman Vershynin. One-bit compressed sensing by linear programming. *Communications on Pure and Applied Mathematics*, 66(8):1275–1297, 2013.

[28] Ariadna Quattoni, Michael Collins, and Trevor Darrell. Conditional random fields for object recognition. In *Advances in neural information processing systems*, pages 1097–1104, 2005.

[29] Miklós Ruszinkó. On the upper bound of the size of the r-cover-free families. *Journal of Combinatorial Theory, Series A*, 66(2):302–310, 1994.

[30] Yanyao Shen and Sujay Sanghavi. Iterative least trimmed squares for mixed linear regression. *arXiv preprint arXiv:1902.03653*, 2019.

[31] Martin Slawski, Matthias Hein, and Pavlo Lutsik. Matrix factorization with binary components. In *Advances in Neural Information Processing Systems*, pages 3210–3218, 2013.

[32] Weixing Song, Weixin Yao, and Yanru Xing. Robust mixture regression model fitting by laplace distribution. *Computational Statistics & Data Analysis*, 71:128–137, 2014.

[33] Yuekai Sun, Stratis Ioannidis, and Andrea Montanari. Learning mixtures of linear classifiers. In *ICML*, pages 721–729, 2014.

[34] Taiyao Wang and Ioannis Ch Paschalidis. Convergence of parameter estimates for regularized mixed linear regression models. *arXiv preprint arXiv:1903.09235*, 2019.

[35] Xinyang Yi, Constantine Caramanis, and Sujay Sanghavi. Solving a mixture of many random linear equations by tensor decomposition and alternating minimization. *arXiv preprint arXiv:1608.05749*, 2016.

[36] Dong Yin, Ramtin Pedarsani, Yudong Chen, and Kannan Ramchandran. Learning mixtures of sparse linear regressions using sparse graph codes. *IEEE Transactions on Information Theory*, 65(3):1430–1451, 2019.

[37] Hong-Tu Zhu and Heping Zhang. Hypothesis testing in mixture regression models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(1):3–16, 2004.

Recovery of sparse linear classifiers from mixture of responses Supplementary Material

A Missing Proofs from Section 4

Proof of Lemma 6. The proof of the lemma follows from a simple application of Chernoff bound.

Let $\{y^i\}_{i=1}^T$ be the set of responses obtained by querying the oracle repeatedly T times with vector $\boldsymbol{v}$. Let $Z = \sum_i \mathbb{1}[y^i = +1]$, and therefore $\mathbb{E}Z = \frac{T \times \mathsf{poscount}}{\ell}$.

Note that Algorithm 1 makes a mistake in estimating poscount only if

$$|Z - \frac{T \times \mathsf{poscount}}{\ell}| \geq \frac{T}{2\ell}.$$

Since the responses in each batch are independent, using Chernoff bound [6], we get an upper bound on the probability that Algorithm 1 makes a mistake in estimating poscount as

$$\Pr\Big(|Z - \mathbb{E}Z| \geq \frac{T}{2\ell}\Big) \leq 2e^{-\frac{T}{2\ell^2}}.$$

The same argument and conclusion holds for observing the negcount of the query vector as well. Also, since $\mathsf{nzcount} = T - (\mathsf{poscount} + \mathsf{negcount})$, using union bound, it follows that $\hat{\mathsf{nz}} \neq \mathsf{nzcount}$ with probability at most $4e^{-\frac{T}{2\ell^2}}$. □

Proof of Lemma 10. We give a non-constructive proof for the existence of $(r,t) - \mathsf{CFF}$ of size n and alphabet $m = O(t^{r+1}\log n)$. Recall that a family of sets $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ where each $\mathcal{H}_i \subseteq [m]$ is an $(r,t) - \mathsf{CFF}$ if the following holds: for all distinct $j_0, j_1, \ldots, j_{t+r-1} \in [n]$, it is the case that

$$\bigcap_{p\in\{0,1,\ldots,r-1\}} \mathcal{H}_{j_p} \not\subseteq \bigcup_{q\in\{r,r+1,\ldots,t+r-1\}} \mathcal{H}_{j_q}.$$

Since PUFF is a special case of $(r,t) - \mathsf{CFF}$ for $r = 2$, this result holds for PUFF as well.

Consider a matrix $\boldsymbol{G}$ of size $m \times n$ where each entry is generated independently from a $\mathrm{Bernoulli}(p)$ distribution with p as a parameter. Consider a distinct set of $t + r$ indices $j_0, j_1, \ldots, j_{t+1}, \ldots, j_{k+r-1} \in [n]$. For a particular row of the matrix $\boldsymbol{G}$, the event that there exists a 1 in the indices $j_0, j_1, \ldots, j_{r-1}$ and 0 in the indices $j_r, j_{r+1}, \ldots, j_{t+r-1}$ holds with probability $p^r(1-p)^t$. Therefore, for a fixed row, this event does not hold with probability $1-p^r(1-p)^t$ and the probability that for all the rows the event does not hold is $(1-p^r(1-p)^t)^m$. Notice that the number of such possible sets of $t + r$ columns is $\binom{n}{t+r}\binom{t+r}{r}$. By taking a union bound, the probability (P_e) that the event does not hold for all the rows for at least one set of $t + r$ indices is

$$P_e \leq \binom{n}{t+r}\binom{t+r}{r}\big(1 - p^r(1-p)^t\big)^m$$

Since we want to minimize the upper bound, we want to maximize $p^r(1-p)^t$. Substituting $p = \frac{1}{t+1}$, we get that

$$p^r(1-p)^t = \Big(\frac{t}{t+1}\Big)^t \cdot \frac{1}{(t+1)^r} > \frac{1}{e(t+1)^r}.$$

Further, using the fact that $\binom{n}{t} \leq \Big(\frac{en}{t}\Big)^t$, we obtain

$$P_e \leq \frac{(en)^{t+r}}{(t+r)^t}\Big(1 - \frac{1}{e(t+1)^r}\Big)^m \leq \frac{(en)^{t+r}}{(t+r)^t}\exp\Big(-\frac{m}{e(t+1)^r}\Big) < \alpha$$

for some very small number η. Taking $\log$ on both sides and after some rearrangement, we obtain

$$m > e(t+1)^r\Big((t+r)\log\frac{en}{t+r} + r\log(t+r) + \log\frac{1}{\eta}\Big).$$

Hence, using $m = O(t^{r+1}\log n)$, the event holds for at least one row for every set of $t + r$ indices with high probability. Therefore, with high probability, the family of sets $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ corresponding to the rows of $\boldsymbol{G}$ is a $(r,t) - \mathsf{CFF}$. □

B Two-stage Approximate Recovery

In this section, we prove the helper Lemmas 11 and 12 to compete the proof of Theorem 1 and also present the proof of Theorem 2. The two stage approximate recovery algorithm, as the name suggests, proceeds in two sequential steps. In the first stage, we recover the support of all the ℓ unknown vectors (presented in Algorithm 2 in Section 5). In the second stage, we use these deduced supports to approximately recover the unknown vectors (Algorithm 5 described in Section B.2).

B.1 Support recovery (Missing proofs from Section 5)

Compute $|\mathcal{S}(i)|$ using Algorithm 3. First, we show how to compute $|\mathcal{S}(i)|$ for every index $i \in [n]$. Let $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ be a $(d, \ell k, 0.5)$-RUFF of size n over alphabet $[m]$. Construct the binary matrix $\boldsymbol{A} \in \{0,1\}^{m \times n}$ from $\mathcal{F}$, as $\boldsymbol{A}_{i,j} = 1$ if and only if $i \in \mathcal{H}_j$. Each column $j \in [n]$ of $\boldsymbol{A}$ is essentially the indicator vector of the set $\mathcal{H}_j$. We use the rows of matrix $\boldsymbol{A}$ as query vectors to compute $|\mathcal{S}(i)|$ for each $i \in [n]$. For each such query vector $\boldsymbol{v}$, we compute the $\mathsf{nzcount}(\boldsymbol{v})$ using Algorithm 1 with batchsize $T = O(\ell^2 \log \ell k n)$. The large value of T ensures that the estimated nzcount is correct for all the queries with very high probability.

For every $h \in \{0, \ldots, \ell\}$, let $\boldsymbol{b}^h \in \{0,1\}^m$ be the indicator of the queries that have nzcount at least h. We show in Lemma 11 that the set of columns of $\boldsymbol{A}$ that have large intersection with $\boldsymbol{b}^h$, exactly correspond to the indices $i \in [n]$ that satisfy $|\mathcal{S}(i)| \geq h$. This allows us to recover $|\mathcal{S}(i)|$ exactly for each $i \in [n]$.

Algorithm 3 COMPUTE–$|\mathcal{S}(i)|$

Require: Construct binary matrix $\boldsymbol{A} \in \{0,1\}^{m \times n}$ from $(d, \ell k, 0.5)-\mathsf{RUFF}$ of size n over alphabet $[m]$, with $m = c_1 \ell^2 k^2 \log n$ and $d = c_2 \ell k \log n$.

Initialize $\boldsymbol{b}^0, \boldsymbol{b}^1, \boldsymbol{b}^2, \ldots, \boldsymbol{b}^\ell$ to all zero vectors of dimension m.
Let batchsize $T = 4\ell^2 \log mn$.
for $i = 1, \ldots, m$ **do**
  Set $w := \mathsf{nzcount}(\boldsymbol{A}[i])$ (obtained using Algorithm 1 with batchsize T.)
  for $h = 0, 1, \ldots, w$ **do**
    Set $\boldsymbol{b}^h_i = 1$.
  end for
end for
for $h = 0, 1, \ldots, \ell$ **do**
  Set $\mathcal{C}_h = \{i \in [n] \mid |\mathsf{supp}(\boldsymbol{b}^h) \cap \mathsf{supp}(\boldsymbol{A}_i)| \geq 0.5d\}$.
end for
for $i = 1, 2, \ldots, n$ **do**
  Set $|\mathcal{S}(i)| = h$ if $i \in \{\mathcal{C}_h \setminus \mathcal{C}_{h+1}\}$ for some $h \in \{0, 1, \ldots, \ell - 1\}$.
  Set $|\mathcal{S}(i)| = \ell$ if $i \in \mathcal{C}_\ell$
end for

Proof of Lemma 11. Since $\boldsymbol{A}$ has $m = O(\ell^2 k^2 \log n)$ distinct rows, and each row is queried $T = O(\ell^2 \log(mn))$ times, the total query complexity of Algorithm 3 is $O(\ell^4 k^2 \log(\ell k n) \log n)$.

To prove the correctness, we first see that the nzcount for each query is estimated correctly using Algorithm 1 with overwhelmingly high probability. From Lemma 6 with $T = 4\ell^2 \log(mn)$, it follows that each nzcount is estimated correctly with probability at least $1 - \frac{1}{mn^2}$. Therefore, by taking a union bound over all rows of $\boldsymbol{A}$, we estimate all the counts accurately with probability at least $1 - \frac{1}{n^2}$.

We now show, using the properties of RUFF, that $|\mathsf{supp}(\boldsymbol{b}^h) \cap \mathsf{supp}(\boldsymbol{A}_i)| \geq 0.5d$ if and only if $|\mathcal{S}(i)| \geq h$, for any $0 \leq h \leq \ell$.

Let $i \in [n]$ be an index such that $|\mathcal{S}(i)| \geq h$, i.e., there exist at least h unknown vectors that have a non-zero entry in their i^{th} coordinate. Also, let $U := \cup_{i \in [\ell]} \mathsf{supp}(\boldsymbol{\beta}^i)$ denote the union of supports of all the unknown vectors. Since each unknown vector is k-sparse, it follows that $|U| \leq \ell k$. To show that $|\mathsf{supp}(\boldsymbol{b}^h) \cap \mathsf{supp}(\boldsymbol{A}_i)| \geq 0.5d$, consider the set of rows of $\boldsymbol{A}$ indexed by $W := \{\mathsf{supp}(\boldsymbol{A}_i) \setminus \cup_{j \in U \setminus \{i\}} \mathsf{supp}(\boldsymbol{A}_j)\}$. Since $\boldsymbol{A}$ is a $(d, \ell k, 0.5) - \mathsf{RUFF}$, we know that $|W| \geq 0.5d$. We now show that $\boldsymbol{b}^h_t = 1$ for every $t \in W$. This follows from the observation that

for $t \in W$, and each unknown vector $\boldsymbol{\beta} \in \mathcal{S}(i)$, the query $\mathsf{sign}(\langle \boldsymbol{A}[t], \boldsymbol{\beta} \rangle) = \mathsf{sign}(\boldsymbol{\beta}_i) \neq 0$. Since $|\mathcal{S}(i)| \geq h$, we conclude that $\mathsf{nzcount}(\boldsymbol{A}[t]) \geq h$, and therefore, $\boldsymbol{b}_t^h = 1$.

To prove the converse, consider an index $i \in [n]$ such that $|\mathcal{S}(i)| < h$. Using a similar argument as above, we now show that $|\mathsf{supp}(\boldsymbol{b}^h) \cap \mathsf{supp}(\boldsymbol{A}_i)| < 0.5d$. Consider the set of rows of $\boldsymbol{A}$ indexed by $W := \{\mathsf{supp}(\boldsymbol{A}_i) \setminus \cup_{j \in U \setminus \{i\}} \mathsf{supp}(\boldsymbol{A}_j)\}$. Now observe that for each $t \in W$, and any unknown vector $\boldsymbol{\beta} \notin \mathcal{S}(i)$, the query $\mathsf{sign}(\langle \boldsymbol{A}[t], \boldsymbol{\beta} \rangle) = 0$. Therefore $\mathsf{nzcount}(\boldsymbol{A}[t]) \leq |\mathcal{S}(i)| < h$, and $\boldsymbol{b}_t^h = 0$ for all $t \in W$. Since $|W| \geq 0.5d$, it follows that $|\mathsf{supp}(\boldsymbol{b}^h) \cap \mathsf{supp}(\boldsymbol{A}_i)| < 0.5d$.

For any $0 \leq h \leq \ell$, Algorithm 3. therefore correctly identifies the set of indices $i \in [n]$ such that $|\mathcal{S}(i)| \geq h$. In particular, the set $C_h := \{i \in [n] \mid |\mathcal{S}(i)| \geq h\}$. Therefore, the set $C_h \setminus C_{h+1}$ is exactly the set of indices $i \in [n]$ such that $|\mathcal{S}(i)| = h$.

□

Compute $|\mathcal{S}(i) \cup \mathcal{S}(j)|$ using Algorithm 4. In this section we present an algorithm to compute $|\mathcal{S}(i) \cup \mathcal{S}(j)|$, for every $i, j \in [n]$, using $|\mathcal{S}(i)|$ computed in the previous step. We will need an $\ell k - \mathsf{PUFF}$ for this purpose. Let $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ be the required $\ell k - \mathsf{PUFF}$ of size n over alphabet $m' = O(\ell^3 k^3 \log n)$.

Construct a set of $\ell+1$ matrices $\mathcal{B} = \{\boldsymbol{B^{(1)}}, \ldots, \boldsymbol{B^{(\ell+1)}}\}$ where, each $\boldsymbol{B^{(w)}} \in \mathbb{R}^{m' \times n}, w \in [\ell+1]$, is obtained from the PUFF $\mathcal{F}$ in the following way: For every $(i, j) \in [m'] \times [n]$, set $\boldsymbol{B}_{i,j}^{(w)}$ to be a random number sampled uniformly from $[0, 1]$ if $i \in H_j$, and 0 otherwise. We remark that the choice of uniform distribution in $[0, 1]$ is arbitrary, and any continuous distribution works.

Since every $\boldsymbol{B^{(w)}}$ is generated identically, they have the exact same support, though the non-zero entries are different. Also, by definition, the support of the columns of every $\boldsymbol{B^{(w)}}$ corresponds to the sets in $\mathcal{F}$.

Let $U := \cup_{i \in [\ell]} \mathsf{supp}(\boldsymbol{\beta}^i)$ denote the union of supports of all the unknown vectors. Since each unknown vector is k-sparse, it follows that $|U| \leq \ell k$. From the properties of $\ell k - \mathsf{PUFF}$, we know that for any pair of indices $(i, j) \in U \times U$, the set $(\mathcal{H}_i \cap \mathcal{H}_j) \setminus \bigcup_{q \in U \setminus \{i,j\}} \mathcal{H}_q$ is non-empty. This implies that for every $w \in [\ell + 1]$, there exists at least one row of $\boldsymbol{B}^{(w)}$ that has a non-zero entry in the i^{th} and j^{th} index, and 0 in all other indices $p \in U \setminus \{i, j\}$. In Algorithm 4 we use these rows as queries to estimate their $\mathsf{nzcount}$. In Lemma 12, we show that this quantity is exactly $|S(i) \cup S(j)|$ for that particular pair $(i, j) \in U \times U$.

Proof of Lemma 12. Computing each count requires $O(T\ell)$ queries. Therefore, the total number of oracle queries made by Algorithm 4 is at most $O(m'T\ell) = O(\ell^6 k^3 \log(\ell k n) \log n)$ for $m' = O(\ell^3 k^3 \log n)$ and $T = 10\ell^2 \log(nm')$. Also, observe that each $\mathsf{nzcount}$ is estimated correctly with probability at least $1 - O\left(1/\ell m' n^2\right)$. Therefore from union bound it follows that all the $(\ell + 1)m'$ estimations of $\mathsf{nzcount}$ are correct with probability at least $1 - O\left(1/n^2\right)$.

Recall that the set U denotes the union of supports of all the unknown vectors. This set is equivalent to $\{i \in [n] \mid |\mathcal{S}(i)| > 0\}$. First, note that if $|\mathcal{S}(i)| = 0$, there are no unknown vectors supported on the i^{th} index. Therefore, $|\mathcal{S}(i) \cup \mathcal{S}(j)| = |\mathcal{S}(j)|$. Also, if $i = j$, then the computation of $|\mathcal{S}(i) \cup \mathcal{S}(j)|$ is trivial.

We now focus on the only non-trivial case when $(i, j) \in U \times U$ and $i \neq j$. Since for every $w \in [\ell + 1]$, the support of the columns of $\boldsymbol{B^{(w)}}$ are the indicators of sets in $\mathcal{F}$, the PUFF property implies that there exists at least one row (say, with index $p \in [m']$) of every $\boldsymbol{B}^{(w)}$ which has a non-zero entry in the i^{th} and j^{th} index, and 0 in all other indices $q \in U \setminus \{i, j\}$, i.e.,

$$\boldsymbol{B_{p,i}^{(w)}} \neq 0, \boldsymbol{B_{p,j}^{(w)}} \neq 0, \text{ and } \boldsymbol{B_{p,q}^{(w)}} = 0 \text{ for all } q \in U \setminus \{i, j\}.$$

To prove the correctness of the algorithm, we need to show the following:

$$|\mathcal{S}(i) \cup \mathcal{S}(j)| = \max_{w \in [\ell+1]} \{\mathsf{nzcount}(\boldsymbol{B^{(w)}}[p])\}$$

First observe that using the row $\boldsymbol{B^{(w)}}[p]$ as query will produce non-zero value for only those unknown vectors $\beta \in \mathcal{S}(i) \cup \mathcal{S}(j)$. This establishes the fact that $|\mathcal{S}(i) \cup \mathcal{S}(j)| \geq \mathsf{nzcount}(\boldsymbol{B^{(w)}}[p])$.

Algorithm 4 RECOVER–$|\mathcal{S}(i) \cup \mathcal{S}(j)|$

Require: $|\mathcal{S}(i)|$ for every $i \in [n]$.

Require: For every $w \in [\ell+1]$, construct $\boldsymbol{B^{(w)}} \in \mathbb{R}^{m' \times n}$ from $\ell k -$ PUFF of size n over alphabet $m' = c_3 \ell^3 k^3 \log n$.

Let $U := \{i \in [n] \mid |\mathcal{S}(i)| > 0\}$
Let batchsize $T = 10\ell^2 \log(nm')$
for every $p \in [m']$ **do**
    Let $\mathsf{count}(p) := \max_{w \in [\ell+1]} \{\mathsf{nzcount}(\boldsymbol{B^{(w)}}[p])\}$ (obtained using Algorithm 1 with batchsize T).
end for
for every pair $(i, j) \in [n] \times [n]$ **do**
    if $i == j$ **then**
        Set $|\mathcal{S}(i) \cup \mathcal{S}(j)| = |\mathcal{S}(i)|$
    else if $i \notin U$ **then**
        Set $|\mathcal{S}(i) \cup \mathcal{S}(j)| = |\mathcal{S}(j)|$
    else if $j \notin U$ **then**
        Set $|\mathcal{S}(i) \cup \mathcal{S}(j)| = |\mathcal{S}(i)|$
    else
        Let $p \in [m']$ such that $\boldsymbol{B^{(1)}_{p,i}} \neq 0$, $\boldsymbol{B^{(1)}_{p,j}} \neq 0$, and $\boldsymbol{B^{(1)}_{p,q}} = 0$ for all $q \in U \setminus \{i, j\}$.
        Set $|\mathcal{S}(i) \cup \mathcal{S}(j)| = \mathsf{count}(p)$.
    end if
end for

To show the other side of the inequality, consider the set of $(\ell + 1)$ 2-dimensional vectors obtained by the restriction of rows $\boldsymbol{B^{(w)}}[p]$ to the coordinates (i, j),

$$\{(\boldsymbol{B^{(w)}_{p,i}}, \boldsymbol{B^{(w)}_{p,j}}) \mid w \in [\ell + 1]\}.$$

Since these entries are picked uniformly at random from $[0, 1]$, they are pairwise linearly independent. Therefore, each $\beta \in \mathcal{S}(i) \cup \mathcal{S}(j)$ can have $\mathsf{sign}(\langle \boldsymbol{B^{(w)}}[p], \boldsymbol{\beta} \rangle) = 0$ for at most 1 of the w queries. So by pigeonhole principle, at least one of the query vectors $\boldsymbol{B^{(w)}}[p]$ will have $\mathsf{sign}(\langle \boldsymbol{B^{(w)}}[p], \boldsymbol{\beta} \rangle) \neq 0$ for all $\beta \in \mathcal{S}(i) \cup \mathcal{S}(j)$. Hence, $|\mathcal{S}(i) \cup \mathcal{S}(j)| \leq \max_w \{\mathsf{nzcount}(\boldsymbol{B^{(w)}}[p])\}$.

□

B.2 Approximate Recovery

Once we have the obtained the support of all unknown vectors, the task of approximate recovery can be achieved using a set of *Gaussian queries*. Recall from Definition 5, a Gaussian query refers to an oracle query with vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \in \mathbb{R}^n$ where each $\boldsymbol{v}_i$ is sampled independently from the standard Normal distribution, $\boldsymbol{v}_i \sim \mathcal{N}(0, 1)$. The use of Gaussian queries in the context of 1-bit compressed sensing ($\ell = 1$) was studied by [20].

Lemma 13 ([20])**.** *For any $\epsilon > 0$, there exists an ϵ-recovery algorithm to efficiently recover an unknown vector in $\mathbb{R}^n$ using $O\left(\frac{n}{\epsilon} \log \frac{n}{\epsilon}\right)$ Gaussian queries.*

In the current query model however, the approximate recovery is a bit intricate since we do not possess the knowledge of the particular unknown vector that was sampled by the oracle. To circumvent this problem, we will leverage the special support structure of the unknown vectors. From Assumption 1, we know that every unknown vector $\boldsymbol{\beta}^t, t \in [\ell]$, has at least one coordinate which is not contained in the support of the other unknown vectors. We will denote the first such coordinate by $\mathsf{rep}(\boldsymbol{\beta}^t)$. Define,

$$\mathsf{rep}(\boldsymbol{\beta}^t) := \mathsf{min}_p\{ \quad p \in \mathsf{supp}(\boldsymbol{\beta}^t) \setminus \bigcup_{q \in [\ell] \setminus \{t\}} \mathsf{supp}(\boldsymbol{\beta}^q)\} \in [n].$$

For ϵ-recovery of a fixed unknown vector $\boldsymbol{\beta}^t$, we will use the set of representative coordinates $\{\mathsf{rep}(\boldsymbol{\beta}^{t'})\}_{t' \neq t}$, to correctly identify its responses with respect to a set of Gaussian queries. In order to achieve this, we first have to recover the sign of $\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)}$ for every $t \in [\ell]$, using an RUFF, which is described in Algorithm 6.

Lemma 14. *Algorithm 6 recovers* $\mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)})$ *for all* $t \in [\ell]$.

With the knowledge of all the supports, and the sign of every representative coordinate, we are now ready to prove Theorem 2. The details are presented in the Algorithm 5.

Algorithm 5 ϵ-RECOVERY, TWO STAGE

Require: Query access to oracle $\mathcal{O}$.
Require: Assumption 1 to be true.
Estimate $\mathsf{supp}(\boldsymbol{\beta}^t)$ for all $t \in [\ell]$ using Algorithm 2.
Estimate $\mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)})$ for all $t \in [\ell]$ using Algorithm 6.
Let Inf be a large positive number.
Let batchsize $T = 4\ell^2 \log(nk/\epsilon)$.
for $t = 1, \ldots, \ell$ **do**
  for $i = 1, \ldots, \tilde{O}(k/\epsilon)$ **do**
    Define $\boldsymbol{v}^t_j := \begin{cases} \mathsf{Inf} & \text{if } j = \mathsf{rep}(\boldsymbol{\beta}^{t'}), \text{for some } t' \neq t \\ \mathcal{N}(0,1) & \text{otherwise} \end{cases}$
    Obtain $\mathsf{poscount}(\boldsymbol{v}^t)$ using Algorithm 1 with batchsize T.
    Let $p_t := |\{t' \neq t \mid \mathsf{sign}(\boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})}) = +1\}|$
    if $\mathsf{poscount}(\boldsymbol{v}^t) \neq p_t$ **then**
      Set $y^t_i = +1$.
    else
      Set $y^t_i = -1$.
    end if
  end for
  From $\{y^t_1, y^t_2, \ldots, y^t_{\tilde{O}(k/\epsilon)}\}$, and $\mathsf{supp}(\boldsymbol{\beta}^t)$ recover $\hat{\boldsymbol{\beta}}^t$ by using Lemma 13.
end for
Return $\{\hat{\boldsymbol{\beta}}^t, t \in [\ell]\}$.

Proof of Theorem 2. For the ϵ-recovery of a fixed unknown vector $\boldsymbol{\beta}^t, t \in [\ell]$, we will generate its correct response with respect to a set of $\tilde{O}(k/\epsilon)$ Gaussian queries using *modified* Gaussian queries. A modified Gaussian query $\boldsymbol{v}^t$ for the t-th unknown vector, is a Gaussian query with a large positive entry in the coordinates indexed by $\mathsf{rep}(\boldsymbol{\beta}^{t'})$, for every $t' \neq t$.

Consider a fixed unknown vector $\boldsymbol{\beta}^t$. Let $\boldsymbol{v} \in \mathbb{R}^n$ be a Gaussian query, i.e., every entry of $\boldsymbol{v}$ is sampled independently from $\mathcal{N}(0,1)$. Algorithm 5 constructs a modified Gaussian query $\boldsymbol{v}^t$ from $\boldsymbol{v}$ as follows:

$$\boldsymbol{v}^t_j = \begin{cases} \mathsf{Inf} & \text{if} \quad j = \mathsf{rep}(\boldsymbol{\beta}^{t'}) \quad \text{for some } t' \neq t \\ \boldsymbol{v}_j & \text{otherwise} \end{cases}.$$

From construction, we know that $\boldsymbol{v}^t_j = \boldsymbol{v}_j$ for all $j \in \mathsf{supp}(\boldsymbol{\beta}^t)$. Therefore,

$$\langle \boldsymbol{v}^t, \boldsymbol{\beta}^t \rangle = \langle \boldsymbol{v}, \boldsymbol{\beta}^t \rangle \qquad \text{and therefore} \qquad \mathsf{sign}(\langle \boldsymbol{v}^t, \boldsymbol{\beta}^t \rangle) = \mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta}^t \rangle).$$

On the other hand, if Inf is chosen to be large enough,

$$\mathsf{sign}(\langle \boldsymbol{v}^t, \boldsymbol{\beta}^{t'} \rangle) = \mathsf{sign}(\boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})}) \qquad \forall t' \neq t,$$

since $\mathsf{Inf} \cdot \boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})}$ dominates the sign of the inner product. Note that in order to obtain an upper bound on the value of Inf, we have to assume that the non-zero entries of every unknown vector have some non-negligible magnitude (at least $1/\mathrm{poly}(n)$).

Note that the $\mathsf{sign}(\boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})})$ was already computed using Algorithm 6, and therefore, the response of the modified Gaussian query with each $\boldsymbol{\beta}^{t'}, t' \neq t$ is known. Now if $\mathsf{poscount}(\boldsymbol{v}^t)$ is different from the number of positive instances of $\mathsf{sign}(\boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})}), t' \neq t$, then it follows that $\mathsf{sign}(\langle \boldsymbol{v}^t, \boldsymbol{\beta}^t \rangle) = +1$. From this we can successfully obtain the response of $\boldsymbol{\beta}^t$ corresponding to a Gaussian query $\boldsymbol{v}$.

Algorithm 5 simulates $O(k/\epsilon \cdot \log(k/\epsilon))$ Gaussian queries for every $\boldsymbol{\beta}^t, t \in [\ell]$ using the modified Gaussian queries $\boldsymbol{v}^t$. Approximate recovery is then possible using Lemma 13 (restricted to the k-non zero coordinates in the $\mathsf{supp}(\boldsymbol{\beta}^t)$).

We now argue about the query complexity and the success probability of Algorithm 5.

For every unknown vector $\boldsymbol{\beta}^t, t \in [\ell]$, we simulate $O(k/\epsilon \cdot \log(k/\epsilon))$ Gaussian queries. Simulating each Gaussian query involves $T = O(\ell^2 \log(nk/\epsilon))$ oracle queries to estimate the $\mathsf{poscount}$. Note that Algorithm 6 can be run simultaneously with Algorithm 3 since they use the same set of queries. The sign recovery algorithm, therefore, does not increase the query complexity of approximate recovery. The total query complexity of Algorithm 5 after the support recovery procedure is at most $O\left((\ell^3 k/\epsilon)\log(nk/\epsilon)\log(k/\epsilon)\right)$.

From Lemma 6, each $\mathsf{poscount}$ is correct with probability at least $1 - O(\epsilon/(n^2k^2))$ and therefore by a union bound over all the $O(\ell k/\epsilon \cdot log(k/\epsilon))$ $\mathsf{poscount}$ estimates, the algorithm succeeds with probability at least $1 - O(1/n)$. □

Proof of Lemma 14. Consider the $(d, \ell k, 0.5) - \mathsf{RUFF}$, $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$, of size n over alphabet $m = O(\ell^2 k^2 \log n)$ used in Algorithm 3. Let $\boldsymbol{A} \in \{0,1\}^{m \times n}$ be the binary matrix constructed from the RUFF in a similar manner, i.e., $\boldsymbol{A}_{i,j} = 1$ if and only if $i \in \mathcal{H}_j$. From the properties of RUFF, we know that for every $t \in [\ell]$, there exists a row (indexed by $i \in [m]$) of $\boldsymbol{A}$ such that $\boldsymbol{A}_{i,u(\boldsymbol{\beta}^t)} \neq 0$, and $\boldsymbol{A}_{i,j} = 0$ for all $j \in U \setminus \{u(\boldsymbol{\beta}^t)\}$, where, $U = \cup_{i \in [\ell]} \mathsf{supp}(\boldsymbol{\beta}^i)$. Therefore, the query with $\boldsymbol{A}[i]$ yields non-zero sign with only $\boldsymbol{\beta}^t$. Since,

$$\mathsf{sign}(\langle \boldsymbol{A}[i], \boldsymbol{\beta}^t \rangle) = \mathsf{sign}(\langle \boldsymbol{e}_{u(\boldsymbol{\beta}^t)}, \boldsymbol{\beta}^t \rangle) = \mathsf{sign}(\boldsymbol{\beta}^t_{u(\boldsymbol{\beta}^t)})$$

$\mathsf{sign}(\boldsymbol{\beta}^t_{u(\boldsymbol{\beta}^t)})$ can be deduced.

Algorithm 6 COMPUTE–$\mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)})$

Require: Binary matrix $\boldsymbol{A} \in \{0,1\}^{m \times n}$ from $(d, \ell k, 0.5) - \mathsf{RUFF}$ of size n over alphabet $[m]$, with $m = O(\ell^2 k^2 \log n)$ and $d = O(\ell k \log n)$.
Require: $\mathsf{rep}(\boldsymbol{\beta}^t) \in [n]$ for all $t \in [\ell]$.
Let batchsize $T = 4\ell^2 \log mn$.
Let $U := \cup_{i \in [\ell]} \mathsf{supp}(\boldsymbol{\beta}^i)$.
for $t = 1, \ldots, \ell$ **do**
  Let $i \in \{\mathsf{supp}(\boldsymbol{A}_{\mathsf{rep}(\boldsymbol{\beta}^t)}) \setminus \cup_{j \in U \setminus \{\mathsf{rep}(\boldsymbol{\beta}^t)\}} \mathsf{supp}(\boldsymbol{A}_j)\}$
  if $\mathsf{poscount}(\boldsymbol{A}[i]) > 0$ (obtained using Algorithm 1 with batchsize T.) **then**
    $\mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)}) = +1$.
  else
    $\mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)}) = -1$.
  end if
end for

□

C Single stage process for ϵ-recovery

The approximate recovery procedure (Algorithm 5), described in Section B.2, crucially utilizes the support information of every unknown vector to design its queries. This requirement forces the algorithm to proceed in two sequential stages.

In particular, Algorithm 5, with the knowledge of the support and the representative coordinates of all the unknown vectors, designed modified Gaussian queries that in turn simulated Gaussian queries for a fixed unknown vector. In this section, we achieve this by using the rows of a matrix obtained from an $(\ell, \ell k) - \mathsf{CFF}$. The property of the CFF allows us to simulate enough Gaussian queries for every unknown vector without the knowledge of their supports. This observation gives us a completely non-adaptive algorithm for approximate recovery of all the unknown vectors.

Consider a matrix $\boldsymbol{A}$ of dimension $m \times n$ constructed from an $(\ell, \ell k)-\mathsf{CFF}$, $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ of size n over alphabet m, as follows:

$$\boldsymbol{A}_{i,j} = \begin{cases} \mathsf{Inf} & \text{if } i \in \mathcal{H}_j \\ v \sim \mathcal{N}(0,1) & \text{otherwise} \end{cases}.$$

In Lemma 15, we show that for every unknown vector $\boldsymbol{\beta}^t$, there exists a row of $\boldsymbol{A}$ that simulates the Gaussian query for it. Therefore, using $\tilde{O}(k/\epsilon)$ independent blocks of such queries will ensure sufficient Gaussian queries for every unknown vector which then allows us to approximately recover these vectors.

Recall the definition of a representative coordinate of an unknown vector $\boldsymbol{\beta}^t$,

$$\mathsf{rep}(\boldsymbol{\beta}^t) := \mathsf{min}_p\{ \quad p \in \mathsf{supp}(\boldsymbol{\beta}^t) \setminus \bigcup_{q \in [\ell] \setminus \{t\}} \mathsf{supp}(\boldsymbol{\beta}^q)\} \in [n].$$

Lemma 15. *For every $t \in [\ell]$, there exists at least one row $\boldsymbol{v}^t$ in $\boldsymbol{A}$ that simulates a Gaussian query for $\boldsymbol{\beta}^t$, and $\mathsf{sign}(\langle \boldsymbol{v}^t, \boldsymbol{\beta}^{t'} \rangle) = \mathsf{sign}(\boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})})$ for all $t' \neq t$.*

Proof of Lemma 15. For any fixed $t \in [\ell]$, consider the set of indices

$$\mathcal{X} = \{\mathsf{rep}(\boldsymbol{\beta}^{t'}) \mid t' \in [\ell] \setminus \{t\}\}.$$

Recall that from the property of $(\ell, \ell k) - \mathsf{CFF}$, we must have

$$\bigcap_{j \in \mathcal{X}} \mathsf{supp}(\boldsymbol{A}_j) \not\subseteq \bigcup_{j \in \cup_{q \in [\ell]} \mathsf{supp}(\boldsymbol{\beta}^q) \setminus \mathcal{X}} \mathsf{supp}(\boldsymbol{A}_j).$$

Therefore, there must exist at least one row $\boldsymbol{v}^t$ in $\boldsymbol{A}$ which has a large positive entry, Inf, in all the coordinates indexed by $\mathcal{X}$. Moreover, $\boldsymbol{v}^t$ has a random Gaussian entry in all the other coordinates indexed by the union of support of all unknown vectors. Since $\boldsymbol{\beta}^t$ is 0 for all coordinates in $\mathcal{X}$, the query $\mathsf{sign}(\langle \boldsymbol{v}^t, \boldsymbol{\beta}^t \rangle)$ simulates a Gaussian query. Also,

$$\mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta}^{t'} \rangle) = \mathsf{sign}(\mathsf{rep}(\boldsymbol{\beta}^{t'})) \qquad \forall t' \neq t$$

since $\mathsf{Inf} \times \boldsymbol{\beta}^{t'}_{\mathsf{rep}(\boldsymbol{\beta}^{t'})}$ dominates the inner product. □

We are now ready to present the completely non-adaptive algorithm for the approximate recovery of all the unknown vectors.

Proof of Theorem 3. The proof of Theorem 3 follows from the guarantees of Algorithm 7. The query vectors of Algorithm 7 can be represented by the rows of the following matrix:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{A} \\ \tilde{\boldsymbol{A}} + \boldsymbol{B}^{(1)} \\ \tilde{\boldsymbol{A}} + \boldsymbol{B}^{(2)} \\ \vdots \\ \tilde{\boldsymbol{A}} + \boldsymbol{B}^{(\mathsf{D})} \end{bmatrix}$$

where, $\mathsf{D} = O(k/\epsilon \cdot \log k/\epsilon)$ and $\boldsymbol{A}$ is the matrix obtained from the $(d, \ell k, 0.5) - \mathsf{RUFF}$ required by Algorithm 2 and Algorithm 6. The matrix $\tilde{\boldsymbol{A}}$ is obtained from an $(\ell, \ell k) - \mathsf{CFF}$, $\mathcal{F} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n\}$ by setting $\tilde{\boldsymbol{A}}_{i,j} = \mathsf{Inf}$ if $i \in \mathcal{H}_j$ and 0 otherwise, and each matrix $\boldsymbol{B}^{(w)}$ for $w \in [\mathsf{D}]$ is a Gaussian matrix with every entry $\boldsymbol{B}^{(w)}_{i,j}$ drawn uniformly at random from standard Normal distribution.

Algorithm 7 decides all its query vectors at the start and hence is completely non-adaptive. It first invokes Algorithm 2 and Algorithm 6 to recover the support and the sign of the representative coordinate of every unknown vector $\boldsymbol{\beta}^t$. Now using the queries from the rows of the matrix $\boldsymbol{R}$, the algorithm generates at least $\mathsf{D} = \tilde{O}(k/\epsilon)$ Gaussian queries for each unknown vector.

Algorithm 7 ϵ-RECOVERY, SINGLE STAGE

Require: Assumption 1 to be true.

Require: Binary matrix $\tilde{\boldsymbol{A}} \in \{0,1\}^{m\times n}$ from $(\ell, \ell k)$ − CFF of size n over alphabet $m = O((\ell k)^{\ell+1}\log n)$.

Estimate $\mathsf{supp}(\boldsymbol{\beta}^t)$ and $\mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)})$ for all $t \in [\ell]$ using Algorithm 2 and Algorithm 6 respectively.
Set Inf to be a large positive number.
Set $\mathsf{D} = O(k/\epsilon \cdot \log(k/\epsilon))$.
Set batchsize $T = 4\ell^2 \log(mnk/\epsilon)$.
for $i = 1, \ldots, m$ **do**
  for $w = 1, 2, \ldots, \mathsf{D}$ **do**
    Construct query vector $\boldsymbol{v}$, where $\boldsymbol{v}_j = \begin{cases} \mathsf{Inf} & \text{if } \tilde{\boldsymbol{A}}_{i,j} = 1 \\ \mathcal{N}(0,1) & \text{otherwise} \end{cases}$.
    $\mathsf{Query}\Big(\boldsymbol{v}, T\Big)$ and set $\boldsymbol{P}_{i,w} = \mathsf{poscount}(\boldsymbol{v})$.
  end for
end for
for $t = 1, \ldots, \ell$ **do**
  Let $\mathcal{X} := \{\mathsf{rep}(\boldsymbol{\beta}^{t'}) \mid t' \in [\ell] \setminus t\}$ and $U := \cup_q \mathsf{supp}(\boldsymbol{\beta}^q)$
  Let $i \in \{\cap_{j\in\mathcal{X}} \mathsf{supp}(\tilde{\boldsymbol{A}}_j) \setminus \bigcup_{j\in U\setminus\mathcal{X}} \mathsf{supp}(\tilde{\boldsymbol{A}}_j)\} \subset [m]$.
  Let $p := |\{t' \neq t \mid \mathsf{sign}(\boldsymbol{\beta}^t_{\mathsf{rep}(\boldsymbol{\beta}^t)}) = +1\}|$
  for $w = 1, \ldots, \mathsf{D}$ **do**
    if $\boldsymbol{P}_{i,w} \neq p$ **then**
      Set $y^t_w = +1$
    else
      Set $y^t_w = -1$
    end if
  end for
  From $\{y^t_w \mid w \in [\mathsf{D}]\}$ and $\mathsf{supp}(\boldsymbol{\beta}^t)$ recover $\hat{\boldsymbol{\beta}}^t$ by using Lemma 13.
end for
Return $\{\hat{\boldsymbol{\beta}}^t \mid t \in [\ell]\}$.

It follows from Lemma 15 that each matrix $\tilde{\boldsymbol{A}} + \boldsymbol{B}^{(w)}$, for $w \in [\mathsf{D}]$, contains at least one Gaussian query for every unknown vector. Therefore, in total, $\boldsymbol{R}$ contains at least $D = O(k/\epsilon \cdot \log k/\epsilon)$ Gaussian queries for every unknown vector $\boldsymbol{\beta}^t$. Using the responses of these Gaussian queries, we can then approximately recover every $\boldsymbol{\beta}^t$ using Lemma 13.

The total query complexity is therefore the sum of query complexities of support recovery process (which from Theorem 1 we know to be at most $O(\ell^6 k^3 \log(n)\log(\ell k n))$), and the total number of queries needed to generate $O(k/\epsilon \cdot \log(k/\epsilon))$ Gaussian queries (which is mTD) for each unknown vector. Therefore the net query complexity is $O\Big((\ell^{\ell+3}k^{\ell+2}/\epsilon)\log n \log(k/\epsilon)\log(n/\epsilon))\Big)$. Each Algorithm 2, 6 and the Gaussian query generation succeed with probability at least $1 - O(1/n)$, therefore from union bound, Algorithm 7 succeeds with probability at least $1 - O(1/n)$. □

D Relaxing Assumption 1 for $\ell = 2$

In this section, we will circumvent the necessity for Assumption 1 when there are only two unknown vectors - $\{\boldsymbol{\beta}^1, \boldsymbol{\beta}^2\}$. We present a two-stage algorithm to approximately recover both the unknown vectors. In the first stage, the algorithm recovers the support of both the vectors, and then using the support information it approximately recovers the two vectors.

We would like to mention that if $\mathsf{supp}(\boldsymbol{\beta}^1) \neq \mathsf{supp}(\boldsymbol{\beta}^2)$, we do not need any further assumptions on the unknown vectors for their approximate recovery. However, if the two vectors have the exact same support, then we need to impose some mild assumptions in order to approximately recover the vectors.

D.1 Support Recovery

In this section, we show that supports of both the unknown vectors can be inferred directly from $\{|\mathcal{S}(i)|\}_{i\in[n]}$ and $\{|\mathcal{S}(i)\cap\mathcal{S}(j)|\}_{i,j\in[n]}$. These quantities were computed using Algorithm 3 and using Algorithm 4 respectively. Moreover, the guarantees of both these algorithms (shown in Lemma 11, and Lemma 12) do not require the unknown vectors to satisfy any special assumption.

Lemma 16. *There exists an algorithm to recover the support of any two k-sparse unknown vectors using $O(k^3\log^2 n)$ oracle queries with probability at least $1-O(1/n^2)$.*

Proof of Lemma 16. Consider Algorithm 8. The query complexity and success guarantees both follow from Lemma 11 and Lemma 12. We now prove the correctness of Algorithm 8.

Algorithm 8 RECOVER–SUPPORT $\ell=2$

Require: Access to oracle $\mathcal{O}$
Estimate $|S(i)|$ for every $i\in[n]$ using Algorithm 3.
Estimate $|S(i)\cap S(j)|$ for every $i,j\in[n]$ using Algorithm 4.
if $|S(i)|\in\{0,2\}$ for all $i\in[n]$ **then**
  $\mathsf{supp}(\boldsymbol{\beta}^1)=\mathsf{supp}(\boldsymbol{\beta}^2)=\{i\in[n]||\mathcal{S}(i)|\neq 0\}$.
else
  Let $i_0=\min\{i||S(i)|=1\}$, and let $i_0\in\mathsf{supp}(\boldsymbol{\beta}^1)$
  for $j\in[n]\setminus\{i_0\}$ **do**
    if $|S(j)|=2$ **then**
      Add j to $\mathsf{supp}(\boldsymbol{\beta}^1)$, and $\mathsf{supp}(\boldsymbol{\beta}^2)$.
    else if $|S(j)|=1$ and $|\mathcal{S}(i_0)\cap\mathcal{S}(j)|=0$ **then**
      Add j to $\mathsf{supp}(\boldsymbol{\beta}^2)$.
    else if $|S(j)|=1$ and $|\mathcal{S}(i_0)\cap\mathcal{S}(j)|=1$ **then**
      Add j to $\mathsf{supp}(\boldsymbol{\beta}^1)$.
    end if
  end for
end if

Case 1: ($\mathsf{supp}(\boldsymbol{\beta}^1)\neq\mathsf{supp}(\boldsymbol{\beta}^2)$). First note that the set of coordinates, $i\in[n]$ with $|\mathcal{S}(i)|=2$ belong to the support of both the unknown vectors. For the remaining indices in $T:=\{i\in[n]||S(i)|=1\}$, we use the following approach to decide the unknown vector whose support they belongs to.

If $|T|=1$, then without loss of generality we can assume $i\in\mathsf{supp}(\boldsymbol{\beta}^1)$. Else if $|T|>1$, we set the smallest index $i_0\in T$ to be in $\mathsf{supp}(\boldsymbol{\beta}^1)$. We then use this index as a pivot to figure out all the other indices $j\in T\cap\mathsf{supp}(\boldsymbol{\beta}^1)$. If both i_0, and j lie in $\mathsf{supp}(\boldsymbol{\beta}^1)$, then $|\mathcal{S}(i_0)\cap\mathcal{S}(j)|=1$, otherwise $|\mathcal{S}(i_0)\cap\mathcal{S}(j)|=0$. So, using Algorithm 8, we can identify the supports of both the unknown vectors.

Case 2: ($\mathsf{supp}(\boldsymbol{\beta}^1)=\mathsf{supp}(\boldsymbol{\beta}^2)$). In this case, we observe that $|\mathcal{S}(i)|\in\{2,0\}$ for all $i\in[n]$. Therefore, both the unknown vectors have the exact same support, and nothing further needs to be done since $\mathsf{supp}(\boldsymbol{\beta}^1)=\mathsf{supp}(\boldsymbol{\beta}^2)=\{i\in[n]||\mathcal{S}(i)|\neq 0\}$. □

D.2 Approximate Recovery

In this section, we present the approximate recovery algorithm. The queries are designed based on the supports of the two vectors.

We split the analysis in two parts. First, we consider the case when the two vectors have different supports, i.e. $\mathsf{supp}(\boldsymbol{\beta}^1)\neq\mathsf{supp}(\boldsymbol{\beta}^2)$. In this case, we use Lemma 17 to approximately recover the two vectors.

Lemma 17. *If $\mathsf{supp}(\boldsymbol{\beta}^1)\neq\mathsf{supp}(\boldsymbol{\beta}^2)$, then there exists an algorithm for ϵ-approximate recovery of any two k-sparse unknown vectors using $O\Big(\frac{k}{\epsilon}\cdot\log(\frac{nk}{\epsilon})\Big)$ oracle queries with probability at least $1-O(1/n)$.*

When the two vectors have the exact same support, we use a set of sub-Gaussian queries to recover the two vectors. This is slightly tricky, and our algorithms succeeds under some mild assumption on the two unknown vectors (Assumption 2).

Lemma 18. *If* $\mathsf{supp}(\boldsymbol{\beta}^1) = \mathsf{supp}(\boldsymbol{\beta}^2)$*, then there exists an algorithm for* ϵ*-approximate recovery of any two* k*-sparse unknown vectors using* $O(\frac{k^2}{\epsilon^4\delta^2}\log^2(\frac{nk}{\delta}))$ *oracle queries with probability at least* $1 - O(1/n)$.

Algorithm 9 ϵ-APPROXIMATE-RECOVERY

1: Estimate $\mathsf{supp}(\boldsymbol{\beta}^1), \mathsf{supp}(\boldsymbol{\beta}^2)$ using Algorithm 8.
2: **if** $\mathsf{supp}(\boldsymbol{\beta}^1) \neq \mathsf{supp}(\boldsymbol{\beta}^2)$ **then**
3: Return $\hat{\boldsymbol{\beta}^1}, \hat{\boldsymbol{\beta}^2}$ using Algorithm 10.
4: **else**
5: Return $\hat{\boldsymbol{\beta}^1}, \hat{\boldsymbol{\beta}^2}$ using Algorithm 11.
6: **end if**

Proof of Theorem 4. The guarantees of Algorithm 9 prove Theorem 4. The total query complexity after support recovery is the maximum of the query complexities of Algorithm 10 and Algorithm 11, which is $O(\frac{k^2}{\epsilon\delta^2}\log^2(\frac{nk}{\delta}))$.

Moreover from Lemma 17 and Lemma 18, we know that both these algorithms succeed with a probability at least $1-O(1/n)$, therefore, Algorithm 9 is also guaranteed to succeed with probability at least $1 - O(1/n)$. □

We now prove Lemma 17 and Lemma 18.

D.2.1 Case 1: $\mathsf{supp}(\boldsymbol{\beta}^1) \neq \mathsf{supp}(\boldsymbol{\beta}^2)$.

Proof of Lemma 17. Consider a coordinate $p \in \mathsf{supp}(\boldsymbol{\beta}^1) \,\Delta\, \mathsf{supp}(\boldsymbol{\beta}^2)$, where Δ denotes the symmetric difference of the two support sets. Without loss of generality we can assume $p \in \mathsf{supp}(\boldsymbol{\beta}^1)$. We first identify the $\mathsf{sign}(\boldsymbol{\beta}^1_p)$ simply using the query vector $\boldsymbol{e}_p$. For the sake of simplicity let us assume $\mathsf{sign}(\boldsymbol{\beta}^1_p) = +1$.

We use two types of queries to recover the two unknown vectors. The *Type 1* queries are modified Gaussian queries, of the form $\boldsymbol{v} + \mathsf{Inf} \cdot \boldsymbol{e}_p$, where $\boldsymbol{v}$ is a Gaussian query vector. *Type 2* query is the plain Gaussian query $\boldsymbol{v}$.

Since $p \in \mathsf{supp}(\boldsymbol{\beta}^1) \setminus \mathsf{supp}(\boldsymbol{\beta}^2)$, the Type 1 queries will always have a positive response with the unknown vector $\boldsymbol{\beta}^1$. Moreover, they will simulate a Gaussian query with $\boldsymbol{\beta}^2$. Therefore from the responses of the oracle, we can correctly identify the response of $\boldsymbol{\beta}^2$ with a set of $O(k/\epsilon \cdot \log(k/\epsilon))$ Gaussian queries. Now, using Lemma 13, we can approximately recover it.

Now since the response of $\boldsymbol{\beta}^2$ with the Type 1 query $\boldsymbol{v} + \mathsf{Inf} \cdot \boldsymbol{e}_p$ and the corresponding Type 2 query $\boldsymbol{v}$, remains the same, we can also obtain correct responses of $\boldsymbol{\beta}^1$ with a set of $O(k/\epsilon \cdot \log(k/\epsilon))$ Gaussian queries. By invoking Lemma 13 again, we can approximately recover $\boldsymbol{\beta}^1$.

The total query complexity of the algorithm is $O(kT/\epsilon \cdot \log(k/\epsilon)) = O(k/\epsilon \cdot \log(nk/\epsilon) \cdot \log(k/\epsilon))$. Also, from Lemma 6, it follows that each oracle query succeeds with probability at least $1 - O(1/mn)$. Therefore by union bound over all $2m$ queries, the algorithm succeeds with probability at least $1 - O(1/n)$. □

D.2.2 Case 2: $\mathsf{supp}(\boldsymbol{\beta}^1) = \mathsf{supp}(\boldsymbol{\beta}^2)$.

We now propose an algorithm for approximate recovery of the two unknown vectors when their supports are exactly the same. Until now for ϵ-recovery, we were using a representative coordinate to generate enough responses to Gaussian queries. However, when the supports are exactly the same, the same trick does not work.

Algorithm 10 ϵ-APPROXIMATE-RECOVERY: CASE 1

Require: $\mathsf{supp}(\boldsymbol{\beta}^1) \neq \mathsf{supp}(\boldsymbol{\beta}^2)$

Set $m = O(k/\epsilon \cdot \log(k/\epsilon))$

Set batchsize $T = 10 \log mn$.

Let Inf be a large positive number.

Let $p \in \mathsf{supp}(\boldsymbol{\beta}^1) \setminus \mathsf{supp}(\boldsymbol{\beta}^2)$, and $s := \mathsf{sign}(\boldsymbol{\beta}^1_p)$.

for $i = 1, \ldots, m$ **do**

  Construct query vector $\boldsymbol{v}$, where $\boldsymbol{v}_j = \mathcal{N}(0,1)$ for all $j \in [n]$.

  Construct query vector $\tilde{\boldsymbol{v}} := \boldsymbol{v} + s \cdot \mathsf{Inf} \cdot \boldsymbol{e}_p$

  $\mathsf{Query}\Big(\boldsymbol{v}, T\Big)$, and $\mathsf{Query}\Big(\tilde{\boldsymbol{v}}, T\Big)$.

  Set $y^i = \begin{cases} +1 & \text{if } \mathsf{poscount}(\tilde{\boldsymbol{v}}) == 2 \\ -1 & \text{if } \mathsf{negcount}(\tilde{\boldsymbol{v}}) == 1 \\ 0 & \text{otherwise} \end{cases}$

  Set $z^i = \begin{cases} +1 & \text{if } y^i = +1 \text{ and } \mathsf{poscount}(\boldsymbol{v}) == 2 \\ -1 & \text{if } y^i = +1 \text{ and } \mathsf{negcount}(\boldsymbol{v}) == 1 \\ +1 & \text{if } y^i = -1 \text{ and } \mathsf{poscount}(\boldsymbol{v}) == 1 \\ -1 & \text{if } y^i = -1 \text{ and } \mathsf{negcount}(\boldsymbol{v}) == 2 \\ +1 & \text{if } y^i = 0 \text{ and } \mathsf{poscount}(\boldsymbol{v}) == 1 \\ -1 & \text{if } y^i = 0 \text{ and } \mathsf{negcount}(\boldsymbol{v}) == 1 \\ 0 & \text{otherwise} \end{cases}$

end for

From $\{y^i \mid i \in [m]\}$ and $\mathsf{supp}(\boldsymbol{\beta}^2)$ recover $\hat{\boldsymbol{\beta}}^2$ by using Lemma 13.

From $\{z^i \mid i \in [m]\}$ and $\mathsf{supp}(\boldsymbol{\beta}^1)$ recover $\hat{\boldsymbol{\beta}}^1$ by using Lemma 13.

For the approximate recovery in this case, we use sub-Gaussian queries instead of Gaussian queries. In particular, we consider queries whose entries are sampled uniformly from $\{-1, 1\}$. The equivalent of Lemma 13 proved by [2] for sub-Gaussian queries enables us to achieve similar bounds.

Lemma 19 (Corollary of Theorem 1.1 of [2])**.** *Let $\boldsymbol{x} \in \mathbb{S}^{n-1}$ be a k-sparse unknown vector of unit norm. Let $\boldsymbol{v_1}, \ldots, \boldsymbol{v_m}$ be independent random vectors in $\mathbb{R}^n$ whose coordinates are drawn uniformly from $\{-1, 1\}$. There exists an algorithm that recovers $\hat{\boldsymbol{x}} \in \mathbb{S}^{n-1}$ using the 1-bit sign measurements $\{\mathsf{sign}(\langle \boldsymbol{v_i}, \boldsymbol{x} \rangle)\}_{i \in [m]}$, such that with probability at least $1 - 4e^{-\alpha^2}$ (for any $\alpha > 0$), it satisfies*

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2 \leq O\left(\|x\|_\infty^{\frac{1}{2}} + \frac{1}{2\sqrt{m}} (\sqrt{k \log(2n/k)} + \alpha) \right).$$

In particular, for $m = O(\frac{k}{\epsilon^4} \log n)$, we get $O(\epsilon + \|x\|_\infty^{\frac{1}{2}})$ - approximate recovery with probability at least $1 - O(1/n)$. Therefore, if the unknown vectors are not *extremely* sparse (Assumption 2), we can get good guarantees on their approximate recovery with sufficient number of sub-Gaussian queries.

The central idea of ϵ-recovery algorithm (Algorithm 11) is therefore to identify the responses of a particular unknown vector $\boldsymbol{\beta}$ with respect to a set of sub-Gaussian queries $\boldsymbol{v} \sim \{-1, 1\}^n$. Then using Lemma 19, we can approximately reconstruct $\boldsymbol{\beta}$.

Let us denote by $\mathsf{response}(\boldsymbol{v})$, the set of distinct responses of the oracle with a query vector $\boldsymbol{v}$. Since there are only two unknown vectors, $|\mathsf{response}(\boldsymbol{v})| \leq 2$. If both unknown vectors have the same response with respect to a given query vector $\boldsymbol{v}$, i.e., $|\mathsf{response}(\boldsymbol{v})| = 1$ then we can trivially identify the correct responses with respect both the unknown vectors by setting $\mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta}^2 \rangle) = \mathsf{sign}(\langle \boldsymbol{v}, \boldsymbol{\beta}^2 \rangle) = \mathsf{response}(\boldsymbol{v})$.

However if $|\mathsf{response}(\boldsymbol{v})| = 2$, we need to identify the correct response with respect to a fixed unknown vector. This *alignment* constitutes the main technical challenge in approximate recovery. To achieve this, Algorithm 11 fixes a pivot query say $\boldsymbol{v}_0$ with $|\mathsf{response}(\boldsymbol{v}_0)| = 2$, and aligns all the other queries with respect to it by making some additional oracle queries.

Let W denote the set of queries such that $|\mathsf{response}(\boldsymbol{v})| = 2$. Also, for any pair of query vectors, $\boldsymbol{v}_1, \boldsymbol{v}_2 \in W$, we denote by $\mathsf{align}_{\boldsymbol{\beta}}(\boldsymbol{v}_1, \boldsymbol{v}_2)$ to be an ordered tuple of responses with respect to the unknown vector $\boldsymbol{\beta}$.

$$\mathsf{align}_{\boldsymbol{\beta}}(\boldsymbol{v}_1, \boldsymbol{v}_2) = (\mathsf{sign}(\langle \boldsymbol{v}_1, \boldsymbol{\beta} \rangle), \mathsf{sign}(\langle \boldsymbol{v}_2, \boldsymbol{\beta} \rangle)).$$

We fix a pivot query $\boldsymbol{v}_0 \in W$ to be one that satisfies $\mathsf{response}(\boldsymbol{v}_0) = \{-1, 1\}$. We can assume without loss of generality that there always exists one such query, otherwise all queries $\boldsymbol{v} \in W$ have $0 \in \mathsf{response}(\boldsymbol{v})$, and Proposition 20 aligns all such responses using $O(\log n)$ additional oracle queries.

Proposition 20. *Suppose for all queries $\boldsymbol{v} \in W$, $0 \in \mathsf{response}(\boldsymbol{v})$. There exists an algorithm that estimates $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v})$ and $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v})$ for any $\boldsymbol{v}, \boldsymbol{v}_0 \in W$ using $O(\log n)$ oracle queries with probability at least $1 - O(1/n)$.*

For a fixed pivot query $\boldsymbol{v}_0 \in W$ such that $\mathsf{response}(\boldsymbol{v}_0) = \{-1, 1\}$, Proposition 21 and Proposition 22 compute $\mathsf{align}_{\boldsymbol{\beta}}(\boldsymbol{v}_0, \boldsymbol{v})$ for all queries $\boldsymbol{v} \in W$ such that $0 \in \mathsf{response}(\boldsymbol{v})$ and $0 \notin \mathsf{response}(\boldsymbol{v})$ respectively.

Proposition 21. *Let $\boldsymbol{v}_0 \in W$ such that $\mathsf{response}(\boldsymbol{v}_0) = \{-1, 1\}$. For any query vector $\boldsymbol{v} \in W$ such that $0 \in \mathsf{response}(\boldsymbol{v})$, there exists an algorithm that computes $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v})$ and $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v})$ using $O(\log n)$ oracle queries with probability at least $1 - O(1/n)$.*

Proposition 22. *Let $\delta > 0$, be the largest real number such that $\boldsymbol{\beta}^1, \boldsymbol{\beta}^2 \in \delta\mathbb{Z}^n$. Let $\boldsymbol{v}_0 \in W$ such that $\mathsf{response}(\boldsymbol{v}_0) = \{-1, 1\}$. For any query vector $\boldsymbol{v} \in W$ such that $\mathsf{response}(\boldsymbol{v}) = \{-1, 1\}$, there exists an algorithm that computes $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v})$ and $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v})$ using $O(\frac{k}{\delta^2} \log(\frac{nk}{\delta}))$ oracle queries with probability at least $1 - O(1/n)$.*

Using the alignment process and Lemma 19, we can now approximately recover both the unknown vectors.

Proof of Lemma 18. Consider Algorithm 11, which basically collects enough responses of an unknown vector for a set of sub-Gaussian queries by aligning all responses.

Without loss of generality, we fix $\boldsymbol{v}_0$ such that $\mathsf{response}(\boldsymbol{v}_0) = \{+1, -1\}$, and also enforce that $\mathsf{sign}(\boldsymbol{v}_0, \boldsymbol{\beta}^1) = +1$. Now, we align all other responses with respect to $\boldsymbol{v}_0$. The proof of Lemma 18 then follows from the guarantees of Lemma 19. For $m = O(\frac{k}{\epsilon^4} \log n)$, along with the assumptions that $\|\boldsymbol{\beta^1}\|_\infty, \|\boldsymbol{\beta^2}\|_\infty = o(1)$, the algorithm approximately recovers $\boldsymbol{\beta^1}, \boldsymbol{\beta^2}$.

The number of queries made by Algorithm 11 is at most mT to generate responses and $O(m\frac{k}{\delta^2} \log(\frac{nk}{\delta}))$ to align all the m responses with respect to a fixed pivot query $\boldsymbol{v}_0$. Therefore the total query complexity of Algorithm 11 is $O(\frac{k^2}{\epsilon^4 \delta^2} \log^2(\frac{nk}{\delta}))$.

All parts of the algorithm succeed with probability at least $1 - O(1/n)$, and therefore the algorithm succeeds with probability at least $1 - O(1/n)$. □

Finally, we prove Proposition 20, Proposition 21 and Proposition 22.

Proof of Proposition 20. For the proof of Proposition 20, we simply use the query vector $\boldsymbol{v}_0 + \boldsymbol{v}$ to reveal whether the 0's in the two response sets correspond to the same unknown vector or different ones. The correctness of Algorithm 12 follows from the fact that there will be a 0 in the response set of $\boldsymbol{v}_0 + \boldsymbol{v}$ if and only if both the 0's correspond to the same unknown vector.

To obtain the complete response set for the query $\boldsymbol{v}_0 + \boldsymbol{v}$ with probability at least $1 - 1/n$, Algorithm 12 makes at most $O(\log n)$ queries.

□

Proof of Proposition 21. In this case, we observe that the response set corresponding to the query $\mathsf{Inf} \cdot \boldsymbol{v} + \boldsymbol{v}_0$ can reveal the correct alignment. To see this, let the response of $\boldsymbol{v}_0$ and $\boldsymbol{v}$ be $\{+1, -1\}$ and $\{\mathsf{s}, 0\}$ respectively for some $\mathsf{s} \in \{\pm 1\}$. The response set corresponding to $\mathsf{Inf} \cdot \boldsymbol{v} + \boldsymbol{v}_0$ will be the set (or multi-set) of the form $\{\mathsf{s}, \mathsf{t}\}$. Since we know $\mathsf{s} = \mathsf{response}(\boldsymbol{v}) \setminus \{0\}$, we can deduce t from the $\mathsf{poscount}(\mathsf{Inf} \cdot \boldsymbol{v} + \boldsymbol{v}_0)$, and $\mathsf{negcount}(\mathsf{Inf} \cdot \boldsymbol{v} + \boldsymbol{v}_0)$.

Algorithm 11 ϵ-APPROXIMATE RECOVERY: CASE 2

Require: $\mathsf{supp}(\boldsymbol{\beta}^1) = \mathsf{supp}(\boldsymbol{\beta}^2)$, Assumption 2.
Set $m = O(\frac{k}{\epsilon^4}\log(n))$
Set batchsize $T = O(\log mn)$
for $i = 1, \ldots, m$ **do**
  Sample query vector $\boldsymbol{v}$ as: $\boldsymbol{v}_j = \begin{cases} +1 & \text{w.p. } 1/2 \\ -1 & \text{w.p. } 1/2 \end{cases}$
  $\mathsf{Query}(\boldsymbol{v}, T)$, and store $\mathsf{response}(\boldsymbol{v})$.
  if $|\mathsf{response}(\boldsymbol{v})| == 1$ **then**
    Set $y^{\boldsymbol{v}} = \mathsf{response}(\boldsymbol{v})$.
    Set $z^{\boldsymbol{v}} = \mathsf{response}(\boldsymbol{v})$.
  else
    Add $\boldsymbol{v}$ to W.
  end if
  Let $\boldsymbol{v}_0$ be an arbitrary $\boldsymbol{v} \in W$.
  for every $\boldsymbol{v} \in W$ **do**
    Set $(y^{\boldsymbol{v}_0}, y^{\boldsymbol{v}}) = \mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v})$.
    Set $(z^{\boldsymbol{v}_0}, z^{\boldsymbol{v}}) = \mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v})$.
  end for
end for
Using $\{y^{\boldsymbol{v}}\}_{\boldsymbol{v}}$, estimate $\hat{\boldsymbol{\beta}^1}$.
Using $\{z^{\boldsymbol{v}}\}_{\boldsymbol{v}}$, estimate $\hat{\boldsymbol{\beta}^2}$.

Algorithm 12 ALIGN QUERIES, CASE 1

Require: $\boldsymbol{v}_0, \boldsymbol{v} \in \{-1, 1\}^n$, $0 \in \mathsf{response}(\boldsymbol{v}_0) \cap \mathsf{response}(\boldsymbol{v})$.
Set batchsize $T = O(\log n)$.
$\mathsf{Query}(\boldsymbol{v}_0 + \boldsymbol{v}, T)$.
if $0 \in \mathsf{response}(\boldsymbol{v}_0 + \boldsymbol{v})$ **then**
  $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (0, 0)$
  $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v}) = (\mathsf{response}(\boldsymbol{v}_0) \setminus \{0\}, \mathsf{response}(\boldsymbol{v}) \setminus \{0\})$
else
  $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (0, \mathsf{response}(\boldsymbol{v}) \setminus \{0\})$
  $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v}) = (\mathsf{response}(\boldsymbol{v}_0) \setminus \{0\}, 0)$
end if

Now, if $\mathsf{t} = +1$, then $(+1, 0)$ are aligned together (response of the same unknown vector) and $(\mathsf{s}, -1)$ are aligned together. Similarly, if $\mathsf{t} = -1$, then $(-1, 0)$ and $(+1, \mathsf{s})$ are aligned together respectively.

The alignment algorithm is presented in Algorithm 13. It makes $O(\log n)$ queries and succeeds with probability at least $1 - 1/n$.

Algorithm 13 ALIGN QUERIES, CASE 2

Require: $\boldsymbol{v}_0, \boldsymbol{v} \in \{-1, 1\}^n$, $0 \in \mathsf{response}(\boldsymbol{v})$, $\mathsf{response}(\boldsymbol{v}_0) = \{\pm 1\}$.
Set batchsize $T = O(\log n)$.
Set Inf to be a large positive number.
$\mathsf{Query}(\boldsymbol{v}_0 + \mathsf{Inf} \cdot \boldsymbol{v}, T)$.
if $\mathsf{response}(\boldsymbol{v}_0 + \mathsf{Inf} \cdot \boldsymbol{v}) = \{\mathsf{response}(\boldsymbol{v}) \setminus \{0\}, +1\}$ **then**
  $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (+1, 0)$
  $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v}) = (-1, \mathsf{response}(\boldsymbol{v}) \setminus \{0\})$
else
  $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (+1, \mathsf{response}(\boldsymbol{v}) \setminus \{0\})$
  $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v}) = (-1, 0)$
end if

□

Proof of Proposition 22. The objective of Proposition 22 is to align the responses of queries $\boldsymbol{v}_0$ and $\boldsymbol{v}$ by identifying which among the following two hypotheses is true:

- $\mathbb{H}_1$: The response of the unknown vectors with both the query vectors $\boldsymbol{v}_0$ and $\boldsymbol{v}$ is same. Since we fixed the $\mathsf{sign}(\langle \boldsymbol{v}_0, \boldsymbol{\beta}^1 \rangle) = 1$, this corresponds to the case when $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (+1, +1)$ and $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (-1, -1)$.

  In this case, we observe that for any query of the form $\eta \boldsymbol{v}_0 + \zeta \boldsymbol{v}$ with $\eta, \zeta > 0$, the response set will remain $\{+1, -1\}$.

- $\mathbb{H}_2$: The response of each unknown vector with both the query vectors $\boldsymbol{v}_0$ and $\boldsymbol{v}$ is different, i.e., $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (+1, -1)$ and $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (-1, +1)$.

  In this case, we note that the response for the queries of the form $\eta \boldsymbol{v}_0 + \zeta \boldsymbol{v}$ changes from $\{-1, 1\}$ to either $\{+1\}, \{-1\}$, or $\{0\}$ for an appropriate choice of $\eta, \zeta > 0$. In particular, the cardinality of the response set for queries of the form $\eta \boldsymbol{v}_0 + \zeta \boldsymbol{v}$ changes from 2 to 1 if $\frac{\eta}{\zeta} \in \left[-\frac{\langle \boldsymbol{\beta}^1, \boldsymbol{v} \rangle}{\langle \boldsymbol{\beta}^1, \boldsymbol{v}_0 \rangle}, -\frac{\langle \boldsymbol{\beta}^2, \boldsymbol{v} \rangle}{\langle \boldsymbol{\beta}^2, \boldsymbol{v}_0 \rangle} \right] \cup \left[-\frac{\langle \boldsymbol{\beta}^2, \boldsymbol{v} \rangle}{\langle \boldsymbol{\beta}^2, \boldsymbol{v}_0 \rangle}, -\frac{\langle \boldsymbol{\beta}^1, \boldsymbol{v} \rangle}{\langle \boldsymbol{\beta}^1, \boldsymbol{v}_0 \rangle} \right]$.

Algorithm 14 ALIGN QUERIES, CASE 3

Require: $\boldsymbol{v}_0, \boldsymbol{v} \in \{0, -1, 1\}^n$, $\mathsf{response}(\boldsymbol{v}) = \mathsf{response}(\boldsymbol{v}_0) = \{\pm 1\}$.
1: Set batchsize $T = O(\log nk/\delta)$.
2: **for** $\eta \in \{\frac{c}{d} \mid c, d \in \mathbb{Z} \setminus \{0\}, |c|, |d| \leq \frac{\sqrt{k}}{\delta}\}$ **do**
3: Query$(\eta \boldsymbol{v}_0 + \boldsymbol{v}, T)$.
4: **if** $|\mathsf{response}(\eta \boldsymbol{v}_0 + \boldsymbol{v})| == 1$ **then**
5: Return $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (+1, -1)$, $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v}) = (-1, +1)$
6: **end if**
7: **end for**
8: Return $\mathsf{align}_{\boldsymbol{\beta}^1}(\boldsymbol{v}_0, \boldsymbol{v}) = (+1, +1)$, $\mathsf{align}_{\boldsymbol{\beta}^2}(\boldsymbol{v}_0, \boldsymbol{v}) = (-1, -1)$

In order to distinguish between these two hypotheses, Algorithm 14 makes sufficient queries of the form $\eta \boldsymbol{v}_0 + \zeta \boldsymbol{v}$ for varying values of $\eta, \zeta > 0$. If for some η, ζ the cardinality of the response set changes from 2 to 1, then we claim that $\mathbb{H}_2$ holds, otherwise $\mathbb{H}_1$ is true. Algorithm 14 then returns the appropriate alignment.

Note that for any query vector $\boldsymbol{v} \in \{-1, 1\}^n$, and any k-sparse unknown vector $\boldsymbol{\beta} \in \mathbb{S}^{n-1}$ the inner product $\langle \boldsymbol{\beta}, \boldsymbol{v} \rangle \in [-\sqrt{k}, \sqrt{k}]$. Moreover, if we assume that the unknown vectors have precision δ, the ratio $\frac{\langle \boldsymbol{\beta}^2, \boldsymbol{v} \rangle}{\langle \boldsymbol{\beta}^2, \boldsymbol{v}_0 \rangle}$ can assume at most $4k/\delta^2$ distinct values. Algorithm 14 therefore iterates through all such possible values of η/ζ in order to decide which among the two hypothesis is true.

The total number of queries made by Algorithm 14 is therefore $4kT/\delta^2 = O(\frac{k}{\delta^2} \log(\frac{nk}{\delta}))$. From Lemma 6, all the responses are recovered correctly with probability $1 - O(1/n)$. □

E Experiments

Similar to the mixed regression model, the problem of learning mixed linear classifiers can be used to model heterogenous data with categorical labels. We provide some simulation results to show the efficacy of our proposed algorithms to reconstruct the component classifiers in the mixture.

Moreover, the algorithm suggested in this work can be used to learn the set of discriminative features of a group of people in a crowd sourcing model using simple queries with binary responses. Each person's preferences represents a sparse linear classifier, and the oracle queries here correspond to the crowdsourcing model. To exemplify this, we provide experimental results using the MovieLens [18] dataset to recover the movie genre preferences of two different users (that may use the same account, thus generating mixed responses) using small number of queries.

E.1 Simulations

We perform simulations that recover the support of $\ell = 2$, k-sparse vectors in $\mathbb{R}^n$ using Algorithm 8. We use random sparse matrices with sufficient number of rows to construct an RUFF. Error is measured in terms of relative hamming distance between the actual and the reconstructed support vectors.

The simulations show an improvement in the accuracy with increasing number of rows allocated to construct the RUFF for different values of $n = 1000, 2000, 3000$ with fixed $k = 5$. This is evident since the increasing number of rows improve the probability of getting an RUFF.

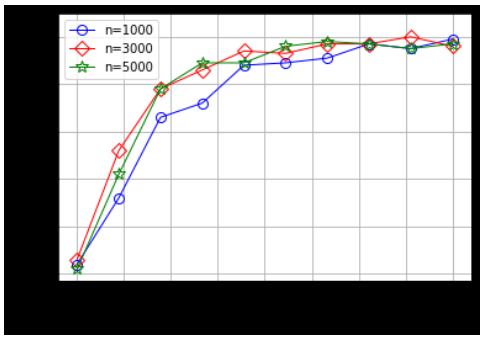

Figure 2: Support Recovery for $\ell = 2, k = 5$ and $n = 1000, 2000, 3000$.

E.2 Movie Lens

The MovieLens [18] database contains the user ratings for movies across various genres. Our goal in this set of experiments is to learn the movie genre preferences of two $(\ell = 2)$ unknown users using a small set of commonly rated movies.

We first preprocess the set of all movies from the dataset to obtain a subset that have an average rating between 2.5 to 3.5. This is done to avoid biased data points that correspond to movies that are liked (or not liked at all) by almost everyone. For the rest of the experiment, we work with this pre-processed set of movies.

We consider $n = 20$ movie genres in some arbitrary, but predetermined order. The genre preference of each user i is depicted as an (unknown) indicator vector $\boldsymbol{\beta}^i \in \{0, 1\}^n$, i.e., $\boldsymbol{\beta}^i_j = 1$ if and only if user i likes the movies in genre j. We assume that a user *likes* a particular movie if they rate it 3 or above. Also, we assume that the user likes a genre if they like at least half the movies they rated in a particular genre.

We consider two users, say U_1, U_2 who have commonly rated at least 500 movies. The preference vectors for both the users is obtained using Algorithm 8. We query the *oracle* with a movie, and obtain its rating from one of the two users at random. For the algorithm, we consider each query to correspond to the indicator of genres that the queried movie belongs to. Using small number of such randomly chosen movie queries, we show that Algorithm 8 approximately recovers the movie genre preference of both the users.

First, we pick a random subset of m movies that were rated by both the users, and partition them into two subsets of size m_1, and m_2 respectively. The first set of m_1 movies are used to partition the list of genres into three classes - genres liked by exactly one of the users, genres liked by both the users, and the genres liked by neither user. These set of m_1 randomly chosen movies essentially correspond to the rows of a RUFF used in Algorithm 8.

We then align the genres liked by exactly one of the users, we use the other set of m_2 randomly chosen movies and obtain two genre preference vectors $\boldsymbol{s_1}, \boldsymbol{s_2}$. Since we do not know whether $\boldsymbol{s_1}$ corresponds the preference vector of U_1 or U_2, we validate it against both, i.e., we validate $\boldsymbol{s_1}$ with U_1, $\boldsymbol{s_2}$ with U_2 and vice versa and select the permutation with higher average accuracy.

Validation: In order to validate our results, we use our recovered preference vectors to predict the movies that U_1 and U_2 will like. For each user U_i, we select the set of movies that were rated by U_i, but were not selected in the set of m movies used to recover their preference vector. The accuracy

of our recovered preference vectors are measured by correctly predicting whether a user will like a particular movie from the test set.

Results: We obtain the accuracy, precision and recall for three random user pairs who have together rated at least 500 movies. The results show that our algorithm predicts the movie genre preferences of the user pair with high accuracy even with small m. Each of the quantities are obtained by averaging over 100 runs.

id: (U_1, U_2)	m_1	m_2	$\mathsf{A}(U_1)$	$\mathsf{P}(U_1)$	$\mathsf{R}(U_1)$	$\mathsf{A}(U_2)$	$\mathsf{P}(U_2)$	$\mathsf{R}(U_2)$
	0	0	0.300	0.000	0.000	0.435	0.000	0.000
(68, 448)	10	20	0.670	0.704	0.916	0.528	0.550	0.706
	30	60	0.678	0.700	0.944	0.533	0.548	0.791
	0	0	0.269	0.000	0.000	0.107	0.000	0.000
(274, 380)	10	20	0.686	0.733	0.902	0.851	0.893	0.946
	30	60	0.729	0.737	0.982	0.872	0.891	0.976
	0	0	0.250	0.000	0.000	0.197	0.000	0.000
(474, 606)	10	20	0.665	0.752	0.827	0.762	0.804	0.930
	30	60	0.703	0.750	0.910	0.787	0.806	0.970