Learning Under Structured Uncertainty

BY

Bethany Austhof
BA, Western Michigan University, 2019

DISSERTATION

Submitted as partial fulfillment of the requirements
for the degree of Doctorate of Philosophy in Mathematics, Statistics, and Computer Science
in the Graduate College of the
University of Illinois Chicago, 2026

Chicago, Illinois

Defense Committee:
Lev Reyzin, Chair, Advisor
Dhruv Mubayi
James Freitag
Vishesh Jain
Marcus Michelen, Northwestern University

Accessibility Statement

This document is in an accessible EPUB format. A PDF version of this document is available at [https://indigo.uic.edu/] or can be obtained by contacting [Bethany Austhof]. An accessible EPUB version of this document is available at [https://indigo.uic.edu/] or can be obtained by contacting [Bethany Austhof].

Dedication

To all my loved ones.

Acknowledgements

The past six years have held a great deal of self-discovery and growth for me. I consider myself lucky to have been afforded the opportunity to spend so much time focusing on my education and personal development.

To begin, I would like to thank my advisor, Lev Reyzin. His guidance was crucial in crafting this thesis, and I will always be appreciative of his encouragement and unwavering support of my ambitions. I always felt comfortable showing up as myself in our meetings. A deep mutual respect made all our conversations and work together an absolute joy in my life.

I would also like to express my gratitude to the rest of my committee: Dhruv Mubayi, James Freitag, Vishesh Jain, and Marcus Michelen.

It has been a beautiful experience to embark on my graduate school journey with one of my closest friends, Nick Christo. Seeing each other grow has been a real treasure. I would also like to thank the friends I made in graduate school. Studying with you all and sharing the revelation of understanding a proof has been wonderful. In particular, Lydia Holley, Clay Mizgerd, Jenny Vaccaro, and Patrick Ward.

I would also like to acknowledge that graduate school gave me the opportunity to explore my passions outside of mathematical pursuits. I developed my love of creating art thanks in part to gifted teachers Sabrina Raaf and Jared Kelley. They both encouraged and pushed my artistic skills, emboldening me to pursue a career in the arts. Sabrina, in particular, has taken me under her wing and, with her support, allowed me to embody my artistic dreams and visions. For that, I will always be grateful.

Lastly, I would like to thank my friends and family for their support, especially my mom. My time in graduate school would not have been nearly as joyous without my closest friends

in Chicago, Dante Moore and Alexander Zimmerman. You both have taught me how sweet friendship can be and how to be a better friend and person through your example. Seeing myself reflected through your lens is a beautiful thing. I look forward to continuing to grow with you both. To the rest of my friends, I love you all, and I can't wait to see what the future holds for us.

Contributions of Authors

Chapter 2 represents the paper *Non-Adaptive Learning of Random Hypergraphs with Queries* by Bethany Austhof, Lev Reyzin and Erasmo Tani. All content including the introduction, literature review, formulations of definitions, theorems, and writing of the manuscript was done jointly with the coauthor.

Chapter 3 represents the paper *A Model for Optimizing Recalculation Schedules to Minimize Regret* by Bethany Austhof and Lev Reyzin. All content including the introduction, literature review, formulations of definitions, theorems, and writing of the manuscript was done jointly with the coauthor.

Chapter 4 represents the manuscript *Cross-Validation Error Dynamics in Smaller Datasets* by Bethany Austhof and Lev Reyzin. All content including the introduction, literature review, formulations of definitions, theorems, and writing of the manuscript was done jointly with the coauthor.

Contents

List of Figures

List of Tables

List of Abbreviations

AdaBoost	Adaptive Boosting
CV	Cross-Validation
ML	Machine Learning
NN	Nueral Network
PAC	Probably Approximately Correct
SVM	Support Vector Machine
VC	Vapnik–Chervonenkis

Summary

Modern learning systems frequently operate under constraints that limit how information is observed, updated, or queried. These constraints which arise from finite data, sampling without replacement, costly updates, or restricted query access, introduce systematic statistical effects that are not captured by classical asymptotic analyses. This thesis studies how such constraints shape learning behavior, error dynamics, and achievable performance guarantees.

We begin by studying the problem of learning a random hypergraph under limited, non-adaptive access. Focusing on the task of recovering random hypergraphs via hyperedge-detection queries, we derive information-theoretic lower bounds and develop efficient randomized algorithms that achieve near-optimal query complexity. By exploiting connections to group testing, we show that randomness in both the data-generating process and the query design enables reliable recovery despite strong identifiability constraints.

We then turn to online learning settings in which updating a model is costly, and learning algorithms must balance performance improvement against the expense of recalculation. Assuming per-round regret guarantees that decay polynomially with sample size, we analyze how the timing and frequency of updates affect cumulative regret. We characterize optimal recalculation schedules under fixed and growing update budgets, establishing sharp tradeoffs between the number of updates and achievable regret rates. These results formalize when near-optimal learning is possible despite severe restrictions on adaptivity.

Finally, we investigate cross-validation in small-data regimes, where repeated random train–test splits induce a notable anti-correlation between training and test errors. We show that this phenomenon arises naturally from hypergeometric sampling from a finite population and persists across a range of classical learning algorithms. By introducing a

simple generative model that incorporates both sampling effects and algorithmic overfitting bias, we derive closed-form expressions for the covariances among training, test, and holdout errors, providing a theoretical framework for empirically observed error dynamics.

Taken together, these results demonstrate how structural constraints on information either imposed by data reuse, update costs, or restricted queries can induce nontrivial and sometimes counterintuitive learning phenomena. Rather than being mere obstacles, such constraints often create regularities that can be characterized, exploited, and optimized. This thesis advances a unified perspective on learning under structured uncertainty, highlighting the central role of sampling and information limitations in modern statistical and algorithmic learning theory.

CHAPTER 1

Introduction

Learning in settings where information is constrained in some manner is a common challenge in modern machine learning. In small-data regimes, where the use of cross-validation is natural, such constraints raise fundamental questions about model selection and evaluation. In online learning settings, where updating a model or data-dependent decision rule may be costly, we seek to understand the trade-offs between update frequency and cumulative regret. Finally, in settings where access to data is limited to indirect or restricted queries, such as in the recovery of latent discrete structures, the central questions concern identifiability and the number of queries required for reliable recovery. This thesis investigates how these diverse settings conduce a shared framework of behavior scaffolded by information constraints.

Throughout this thesis, we consider supervised learning problems in which a model is trained on labeled data and evaluated via an error metric on unseen samples. Learning performance is measured primarily in terms of error rates or regret, depending on the setting, and comparisons are made with respect to either a fixed benchmark or an idealized reference model. We distinguish between expected error, defined with respect to the underlying data-generating distribution, and observed error, which arises from finite samples and specific realizations of randomness. Our focus is on finite-sample behavior, where these two quantities may differ substantially and where structural effects induced by sampling and information constraints become most apparent.

The uncertainty studied in this thesis arises from specific mechanisms inherent to the learning process rather than from unmodeled noise. These mechanisms include sampling variability due to finite datasets and sampling without replacement, algorithmic randomness introduced by procedures such as random data splits or randomized query designs, and systematic bias arising from model capacity and overfitting. Importantly, these sources

of uncertainty induce dependencies and correlations that persist across repetitions of the learning process. We therefore view uncertainty in these settings as structured, in the sense that its effects are predictable and amenable to theoretical analysis.

1. Overview of Relevant Topics

1.1. Query Learning and Random Hypergraphs. In query learning, a learner does not observe data directly but instead acquires information through responses to queries posed to an oracle. These queries may be adaptive, depending on previous responses, or non-adaptive, specified in advance. A central challenge in this setting is identifiability: determining whether the available queries contain sufficient information to uniquely recover the underlying structure of interest. In this thesis, we study this problem in the context of recovering latent combinatorial structures, including random hypergraphs, from restricted query access. Randomized query designs often play a crucial role in making recovery feasible, as they impose average-case structure that can be exploited algorithmically and analyzed theoretically.

Query learning models capture settings in which direct observation of data is infeasible or prohibitively expensive. Such scenarios arise in applications including biological testing, network tomography, and combinatorial structure discovery, where information is obtained only through indirect measurements. In these settings, the design of queries and the interpretation of oracle responses are central to the feasibility of learning. The fundamental question is not only how to recover the underlying object, but whether recovery is possible at all given the available query access.

In query learning, the learner seeks to recover an unknown object H by issuing queries to an oracle and observing the corresponding responses. Each query is selected from a prescribed class and returns information determined by both the query and the underlying object. Queries are said to be *adaptive* if they are chosen sequentially based on past responses, and *non-adaptive* if they are fixed in advance. A learning problem is *identifiable* if the oracle responses uniquely determine H within a specified hypothesis class; otherwise, exact

recovery is information-theoretically impossible. In this thesis, we consider settings in which identifiability may only hold under probabilistic assumptions on H, such as when H is a random hypergraph.

In worst-case formulations, query recovery problems often admit strong impossibility results, even when the underlying structure is relatively simple. This motivates the study of average-case models in which the object to be recovered is drawn from a known distribution. In this thesis, we focus on random hypergraphs, where probabilistic structure enables efficient recovery using randomized, non-adaptive query designs. We analyze both information-theoretic limits and algorithmic strategies, demonstrating how randomness can transform otherwise intractable recovery problems into feasible ones.

1.2. Regret and Online Learning. In online learning, data arrive sequentially and decisions must be made before observing future outcomes. Performance in this setting is commonly measured using regret, which quantifies the cumulative difference between the loss incurred by the learner and that of an optimal reference decision chosen in hindsight. Per-round regret captures instantaneous suboptimality, while cumulative regret reflects the long-term cost of learning under uncertainty. Regret provides a natural framework for analyzing learning behavior when decisions are irrevocable and information is revealed over time. In many practical settings, however, updating a model or decision rule is itself costly, motivating the study of learning dynamics under explicit constraints on the frequency of updates.

Online learning models arise naturally in settings where data are revealed sequentially and decisions must be made in real time. Examples include adaptive control, recommendation systems, and resource allocation problems. In such settings, traditional batch performance metrics are insufficient, as decisions cannot be revised retroactively. Regret provides a principled way to evaluate learning performance by comparing the learner's cumulative loss to that of an optimal reference chosen in hindsight.

In an online learning setting, a learner observes a sequence of data points or outcomes $x_1, x_2, \ldots, x_T$ over time and selects actions or predictors $p_1, p_2, \ldots, p_T$ sequentially. At each round t, the learner incurs a loss $\ell(p_t, x_t)$. The *per-round regret* is defined as the difference

between the loss incurred by the learner at round t and the loss of a fixed reference action chosen in hindsight, while the *cumulative regret* is the sum of per-round regrets over all rounds. Regret therefore quantifies the total performance gap between the learner and an ideal benchmark with full knowledge of the data sequence.

Most classical regret analyses implicitly assume that updating the learner's decision rule is computationally or operationally inexpensive. In practice, however, updates may incur significant costs, such as retraining time, system reconfiguration, or financial expense. This thesis examines how explicit constraints on update frequency alter the trade-offs between learning speed and cumulative regret. By analyzing learning dynamics under limited recalculation budgets, we characterize when near-optimal regret rates remain achievable and when such constraints fundamentally limit performance.

1.3. Cross Validation. Cross-validation (CV) is a standard technique for estimating a model's generalization performance by repeatedly partitioning a dataset into training and test subsets. Under idealized assumptions, different CV splits are treated as approximately independent evaluations of the same underlying learning process. In small-data regimes, however, this assumption breaks down: repeated splits reuse a substantial fraction of the same data, inducing dependencies between training and test errors across folds. As a result, the sampling mechanism underlying cross-validation can systematically shape observed error behavior, including the emergence of strong anti-correlations between training and test performance.

Cross-validation is widely used in practice not only as an evaluation tool but also as a mechanism for model selection and comparison. In settings where data are limited and independent test sets are unavailable, cross-validation often serves as the primary basis for choosing between competing models or hyperparameters. As a result, the statistical behavior of cross-validation error estimates directly influences downstream decisions about model complexity and deployment. Understanding how cross-validation behaves under finite-sample constraints is therefore critical for interpreting empirical performance and avoiding systematic bias in model selection.

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ denote a finite dataset drawn from an underlying data-generating distribution $\mathcal{P}$. A cross-validation procedure consists of repeatedly partitioning $\mathcal{D}$ into disjoint training and test sets, training a model on the training portion, and evaluating its empirical error on the corresponding test portion. The resulting test error is used as an estimator of the model's *generalization error*, defined as the expected loss on a fresh data point drawn from $\mathcal{P}$. Because cross-validation splits reuse data points across different partitions, the training and test errors obtained from different splits are generally dependent random variables when N is finite.

This thesis investigates how the finite-sample sampling structure underlying cross-validation induces systematic dependencies between training and test errors across splits. We show that these dependencies can give rise to pronounced anti-correlations that are not artifacts of specific algorithms but rather consequences of sampling without replacement from a fixed dataset. By characterizing these effects analytically and empirically, we provide a principled explanation for observed error behavior in small-data regimes and clarify the limitations of treating cross-validation estimates as independent.

2. Preliminaries

2.1. Notation. Throughout this thesis, random variables are defined on an underlying probability space that captures both the randomness of the data-generating process and any algorithmic randomness. Expectations and probabilities are taken with respect to this space unless stated otherwise. For a distribution $\mathcal{P}$, we write $(x, y) \sim \mathcal{P}$ to denote a sample drawn from $\mathcal{P}$, and we use $\mathbb{E}$ to denote expectation.

2.2. Data and Loss Functions. Let $\mathcal{X}$ denote an input space and $\mathcal{Y}$ an output space. Let $\mathcal{P}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. A dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ consists of N samples drawn without replacement from $\mathcal{P}$.

Let $\mathcal{H}$ denote a hypothesis class. A hypothesis $h \in \mathcal{H}$ incurs loss $\ell(h(x), y)$ on an example (x, y), where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is a fixed loss function.

Given a dataset $\mathcal{S} \subseteq \mathcal{D}$, the empirical error of h on $\mathcal{S}$ is defined as

$$\widehat{L}_{\mathcal{S}}(h) = \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} \ell(h(x), y).$$

The expected (generalization) error of h is defined as

$$L_{\mathcal{P}}(h) = \mathbb{E}_{(x,y)\sim\mathcal{P}}[\ell(h(x), y)].$$

2.3. PAC Learning and VC Dimension. The Probably Approximately Correct (PAC) learning framework (Valiant, 1984) formalizes when a hypothesis class can be learned from finitely many labeled examples. A hypothesis class $\mathcal{H}$ is said to be PAC learnable if there exists a learning algorithm such that, for every distribution $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$ and for all $\varepsilon, \delta > 0$, the algorithm outputs a hypothesis $h \in \mathcal{H}$ satisfying

$$\Pr\left[L_{\mathcal{P}}(h) \leq \inf_{h'\in\mathcal{H}} L_{\mathcal{P}}(h') + \varepsilon\right] \geq 1 - \delta$$

after observing a number of samples polynomial in $1/\varepsilon$, $1/\delta$, and the complexity of $\mathcal{H}$.

The Vapnik–Chervonenkis (VC) dimension (Vapnik and Chervonenkis, 1971) provides a combinatorial measure of the capacity of a hypothesis class. A set $S \subseteq \mathcal{X}$ is said to be *shattered* by $\mathcal{H}$ if for every labeling of S there exists a hypothesis in $\mathcal{H}$ that realizes that labeling. The VC dimension of $\mathcal{H}$, denoted $\mathrm{VC}(\mathcal{H})$, is the size of the largest set that can be shattered by $\mathcal{H}$. Finite VC dimension characterizes PAC learnability for many standard hypothesis classes and governs the sample complexity required to guarantee uniform convergence of empirical error to expected error.

2.4. Query Learning. Let $\mathcal{H}$ be a hypothesis class and let $H \in \mathcal{H}$ be an unknown object. A query learning problem consists of an oracle $\mathcal{O}$ that maps a query $q \in \mathcal{Q}$ to a response $\mathcal{O}(q, H)$.

A query algorithm selects a sequence of queries $(q_1, \ldots, q_m)$. The algorithm is adaptive if q_i may depend on $\{\mathcal{O}(q_j, H)\}_{j<i}$, and non-adaptive otherwise.

The hypothesis class $\mathcal{H}$ is identifiable under query class $\mathcal{Q}$ if for any $H \neq H' \in \mathcal{H}$, there exists a query $q \in \mathcal{Q}$ such that

$$\mathcal{O}(q, H) \neq \mathcal{O}(q, H').$$

2.5. Random Hypergraphs. A k-uniform hypergraph on a vertex set V is a collection $E \subseteq \binom{V}{k}$. A random hypergraph is generated by including each k-subset of V independently with probability q. All probabilistic statements involving hypergraphs are taken with respect to this generative model.

Concentration inequalities structure many probabilistic proofs. We make explicit the following version of Hoeffding's inequality, as it is employed in our proofs in Chapter 2.

LEMMA 1. *Let $X \sim Binom(n, p)$. Then, for any $t > 0$:*

$$\Pr\left[\left|\frac{X}{n} - p\right| \geq t\right] \leq 2e^{-2nt^2}.$$

2.6. Online Learning. In the online learning model, a learner interacts with a sequence of outcomes $x_1, x_2, \ldots, x_T$. At each round t, the learner selects an action or prediction p_t from an action set $\mathcal{P}$ and incurs loss $\ell(p_t, x_t)$.

Let $p^\star \in \mathcal{P}$ denote a fixed reference action minimizing cumulative loss in hindsight. The cumulative regret after T rounds is defined as

$$R_T = \sum_{t=1}^{T} \left(\ell(p_t, x_t) - \ell(p^\star, x_t)\right).$$

The per-round regret at time t is defined as the summand inside the above expression.

We say that an algorithm achieves regret rate $r(T)$ if $\mathbb{E}[R_T] = O(r(T))$.

The Littlestone dimension (Littlestone, 1988) provides a combinatorial characterization of online learnability in adversarial environments. Consider a complete binary tree whose internal nodes are labeled by elements of $\mathcal{X}$ and whose edges correspond to binary labels. A hypothesis class $\mathcal{H}$ is said to shatter such a tree if for every root-to-leaf path there exists a hypothesis in $\mathcal{H}$ consistent with all labels along that path. The Littlestone dimension of $\mathcal{H}$, denoted $\mathrm{Ldim}(\mathcal{H})$, is the maximum depth of a tree that can be shattered by $\mathcal{H}$. Finite

Littlestone dimension characterizes the existence of online learning algorithms with bounded regret in adversarial settings.

2.7. Recalculation Schedules. A recalculation schedule is a subset $\mathcal{C} \subseteq \{1, 2, \ldots, T\}$ indicating the rounds at which the learner updates its decision rule. Between recalculation times, the learner continues to use the most recently computed action or hypothesis.

The cost of a schedule is measured by $|\mathcal{C}|$, the number of recalculations performed.

2.8. Cross-Validation. A cross-validation split is a random partition of a dataset, $\mathcal{D}$, into two disjoint subsets $(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$ with fixed cardinalities. Unless otherwise specified, splits are sampled uniformly at random without replacement from $\mathcal{D}$.

Given a split, a learning algorithm produces a hypothesis h_{train} using only $\mathcal{D}_{\text{train}}$. The associated training error and test error are defined as

$$\widehat{L}_{\text{train}} = \widehat{L}_{\mathcal{D}_{\text{train}}}(h_{\text{train}}), \qquad \widehat{L}_{\text{test}} = \widehat{L}_{\mathcal{D}_{\text{test}}}(h_{\text{train}}).$$

Across repeated splits, $\widehat{L}_{\text{train}}$ and $\widehat{L}_{\text{test}}$ are treated as random variables induced by the sampling procedure.

3. Background and Related Foundations

The three settings studied in this thesis, query recovery, online learning with regret, and cross-validation, each arise from well-established traditions in statistical learning theory and theoretical computer science. While these areas have developed largely independently, they share a common concern: understanding how limitations on available information affect the reliability and efficiency of learning.

3.1. Query Learning and Combinatorial Recovery. The study of learning through queries originates in work by (Angluin, 1988) on exact learning from membership and equivalence queries. In this framework, a learner seeks to identify an unknown target object by interacting with an oracle that answers structured questions about the object. Angluin's model established a formal foundation for learning when direct access to labeled data is

unavailable, and it has since been extended to a variety of combinatorial identification problems. These include group testing, decision tree learning, and the recovery of structured objects such as graphs, hypergraphs, and Boolean functions.

Query learning models problems in which the learner cannot directly observe the underlying object but instead obtains information through structured queries. This framework generalizes classical group testing, combinatorial search, and oracle-based identification problems. A fundamental distinction in this literature is between adaptive and non-adaptive query strategies. Adaptive algorithms tailor future queries based on past responses, often achieving stronger guarantees at the cost of sequential interaction. Non-adaptive algorithms fix all queries in advance, enabling parallelization but typically requiring more careful design.

In worst-case formulations, recovery problems frequently admit strong lower bounds, even when the underlying object is sparse or structured. This has motivated the study of average-case models, in which the unknown object is drawn from a known probability distribution. Random graph and hypergraph models provide a natural setting for such analyses. In these models, probabilistic structure enables concentration phenomena and typical-instance behavior that can be exploited algorithmically. The resulting guarantees often balance information-theoretic limits, captured by identifiability conditions with computational considerations such as query complexity and decoding time.

3.2. Online Learning and Regret Minimization. Online learning formalizes sequential decision-making under uncertainty. A central performance metric in this framework is regret, which measures the cumulative excess loss of the learner relative to a benchmark chosen in hindsight. In stochastic settings, where outcomes are drawn from a fixed distribution, classical results establish regret rates that decay sublinearly in time, often of order $O(T^{1-\varepsilon})$ or $O(\log T)$ depending on assumptions. In adversarial settings, regret bounds characterize worst-case guarantees independent of probabilistic assumptions.

Standard formulations implicitly assume that the learner may update its strategy at every round without additional cost. In many realistic systems, however, updating a model requires nontrivial computation, resource allocation, or operational changes. Introducing

explicit constraints on the frequency of updates alters the geometry of regret accumulation: between updates, the learner operates with stale information, potentially incurring additional loss. The resulting trade-off between update frequency and cumulative regret lies at the intersection of learning theory and resource-constrained optimization. While regret minimization is well understood in unconstrained settings, comparatively less work characterizes optimal performance when updates themselves are limited.

3.3. Cross-Validation and Finite-Sample Generalization. Cross-validation is one of the most widely used procedures for estimating generalization error in supervised learning. Classical statistical analyses treat test error as a proxy unbiased estimator of expected loss under suitable independence assumptions. When datasets are large relative to model complexity, asymptotic approximations often justify treating different validation splits as effectively independent.

However, finite-sample settings introduce structural dependencies that are typically ignored in asymptotic analyses. When data are sampled without replacement from a fixed dataset, training and test sets across different splits overlap substantially. As a result, empirical error estimates become correlated random variables whose joint behavior is governed by the combinatorics of sampling rather than by independent draws from the underlying distribution. While variance estimation for cross-validation has been studied extensively, less attention has been given to the systematic covariance structure induced by repeated reuse of finite data. Understanding this dependence structure is central to interpreting cross-validation outcomes in small-data regimes, particularly when model selection decisions hinge on relative performance across splits.

3.4. Common Themes. Despite their differing surface forms, these three areas share structural similarities. Each studies learning under a constraint on accessible information: finite sampling without replacement in cross-validation, delayed updates in online learning, and restricted oracle access in query recovery. In each case, classical results describe idealized or asymptotic performance, while the constrained regimes introduce new dependencies and trade-offs that alter achievable guarantees.

This thesis develops a unified perspective on these phenomena by focusing on how structured limitations, rather than arbitrary noise, govern the joint behavior of recoverability, regret accumulation, and error estimates. By analyzing the combinatorial and probabilistic structure induced by these constraints, we aim to clarify the fundamental limits and opportunities that arise when learning systems operate under realistic informational restrictions.

CHAPTER 2

Non-adaptive Learning of Random Hypergraphs with Queries

This chapter was previously published as Non-adaptive Learning of Random Hypergraphs with Queries by Bethany Austhof, Lev Reyzin and Erasmo Tani (Austhof et al., 2025)

1. Introduction

The problem of learning graphs through edge-detecting queries has been extensively studied due to its many applications, ranging from learning pairwise chemical reactions to genome sequencing problems (Alon and Asodi, 2005; Alon et al., 2004; Angluin and Chen, 2008; Grebinski and Kucherov, 1998; Reyzin and Srivastava, 2007). While significant progress has been made in this area, less is known about efficiently learning hypergraphs, which have now become the de facto standard to model higher-order network interactions (Battiston et al., 2020; Benson et al., 2016; Lotito et al., 2022). In this paper, we take another step toward bridging this gap in the literature.

We study the problem of learning a hypergraph by making hyperedge-detecting queries. In particular, we focus on the *non-adaptive setting* (in which every query needs to be submitted in advance), which is more suited for bioinformatics applications.

A lower bound of Abasi and Nader (2019) shows that it is impossible in general to design algorithms that achieve a query complexity nearly linear in the number of (hyper)edges, even for graphs. A recent paper by Li et al. (2019) shows that this lower bound can be beaten for graphs that are generated from a known Erdős-Rényi model. We extend their results by providing algorithms for learning random hypergraphs that have nearly linear query complexity in the number of hyperedges.

1.1. Background and Related Work.

From Group Testing to Hypergraph Query Learning. In the standard group testing model Aldridge et al. (2019); Dorfman (1943); Du and Hwang (1999), one is given a finite set, containing an unknown set of faulty elements. The main task of interest is to recover the set of faulty elements by repeatedly asking questions of the form:

"Does this subset contain at least one faulty element?"

At its core, the problem of learning a hypergraph via hyperedge-detection queries is a constrained group testing problem. Here, the role of the faulty item is taken by the hyperedges of an unknown k-uniform hypergraph G supported on a known set of vertices V. Note that, if one was allowed to ask whether an arbitrary collection $\mathcal{S}$ of elements of $\binom{V}{k}$ contains a hyperedge[1], then the problem would be entirely analogous to the standard group testing problem. Instead, we require that the collection of hyperedges queried is of the form $\mathcal{S} = \binom{S}{k}$ for some subset $S \subseteq V$. Intuitively, the fact that the queries must be specified by a subset of V, as opposed to a subset of $\binom{V}{k}$, renders the problem more difficult.

For concreteness, consider the special case in which we are learning a graph by making edge-detection queries. We may test whether the subset $\{v_1, v_2, v_3\}$ of vertices contains at least one edge, but there is no way to test whether at least one of the edges v_1v_2 and v_2v_3 is present in the graph in a single test (without also including v_2v_3 in the test).

Recent advances in this model focus on algorithms that achieve low decoding time, i.e. allow to quickly reconstruct the set of faulty elements given the answer to all the queries, and in this paper, we make use of a result of Cheraghchi and Ribeiro (2019) within a reduction used by one of our algorithms.

Learning Hypergraphs. Torney (1999) was the first to generalize group testing to the problem of learning hypergraphs via hyperedge-detection queries, a problem he refers to as testing for *positive subsets*. The problem has since come under different names including *group testing for complexes* (Chodoriwsky and Moura, 2015; Macula et al., 2004) and the *monotone DNF query learning problem* (Angluin, 1988; Gao et al., 2006; Abasi et al., 2014).

[1]As is customary we use the notation $\binom{S}{k}$ to refer to the collection of all k-element subsets of a set S.

Angluin and Chen (2008) show that learning arbitrary (non-uniform) hypergraphs of rank k with m hyperedges requires at least $\Omega\left(\left(\frac{2m}{k}\right)^{k/2}\right)$ queries.[2] The same authors (Angluin et al., 2006) showed that an arbitrary k-uniform hypergraph can be learned with high probability by making at most $O(2^{4k}m\,\mathrm{poly}(m, \log n))$ hyperedge-detection queries. Their algorithm makes use of $O(\min\{2^k(\log m + k)^2, (\log m + k)^3\})$ adaptive rounds. They also relax the uniformity condition by giving algorithms that perform well when the hypergraph is nearly uniform.

Abasi et al. (2014) designed randomized adaptive algorithms for learning arbitrary hypergraphs with hyperedge-detecting queries. They also provide lower bounds for the problem they consider.

Gao et al. (2006) gave the first explicit non-adaptive algorithm for learning k-uniform hypergraphs (exactly and with probability one) from hyperedge-detection queries. Abasi et al. (2018) then give non-adaptive algorithms for learning arbitrary hypergraphs of rank (at most) k in the same setting that run in polynomial time in the optimal query complexity for their version of the problem. This, in general, may not be polynomial in the size of the hypergraph. Abasi (2018) considers the same problem in the presence of errors. In particular, they focus on a model in which up to an α-fraction of the queries made may return the incorrect answer.

Balkanski et al. (2022) study algorithms for learning restricted classes of hypergraphs. They give an $O(\log^3 n)$-adaptive algorithm for learning an arbitrary hypermatching (a hypergraph with maximum degree 1) which makes $O(n \log^5 n)$ hyperedge-detection queries and returns the correct answer with high probability.

1.2. Our Results. In this paper, we generalize the results of Li et al. (2019) to Erdős-Rényi hypergraphs.

In Section 2, we discuss a class of typical instances and use it to derive unconditional lower bounds on the learning problem. In Section 3, we give an algorithm which solves the

[2]We note that in general, one must require the hypergraph to be a *Sperner* hypergraph, i.e. one in which no hyperedge is a subset of another, since otherwise the learning problem is not identifiable (See, e.g. Abasi et al. (2018)).

problem with low query complexity and decoding time. In particular, we prove the following:

THEOREM 2. *There exists an algorithm (Algorithm 1) that on input a hyperedge-detection oracle for an Erdős-Rényi hypergraph, makes* $O(k\bar{m}\log^2\bar{m} + k^2\bar{m}\log\bar{m}\log^2 n)$ *non-adaptive queries to the oracle, and outputs the correct answer with probability* $\Omega(1)$. *Here, the probability is taken over the randomness in both the algorithm and in the hypergraph. The algorithm requires* $O(k\bar{m}\log^2\bar{m} + k^3\bar{m}\log\bar{m}\log^2 n)$ *decoding time.*

In Section 4, we will go over hypergraph adaptions of popular group testing algorithms. Specifically, we adapt the `COMP`, `DD` and `SSS` algorithms (see, e.g. Aldridge et al. (2019)), and establish that they all output the correct hypergraph with probability $\Omega(1)$ with $\Omega(\bar{m}\log n)$ queries, thus achieving a better query complexity than the algorithm in Theorem 2 at the price of a higher decoding time.

1.3. Preliminaries.

Erdős-Rényi Hypergraphs. A k-uniform hypergraph is a tuple $G = (V, E)$ where V is a finite set and $E \subseteq \binom{V}{k}$ is a collection of k-element subsets of V, called hyperedges. We refer to the elements of V as *nodes* or *vertices* and denote by n the number $|V|$ of vertices in G, and by m the number $|E|$ of hyperedges. Whenever the hypergraph G is not clear from context, we may use $m(G)$ to refer to the number of hyperedges in G. We refer to the cardinality k of the hyperedges of G as the *rank* or the *arity* of G. While our guarantees have an explicit dependence on k, we will focus on the regime in which k does not grow with n, i.e. $k = O(1)$.

For any hypergraph G, we define its maximum degree as:

$$\Delta(G) := \max_{v \in V} |\{h \in E \mid v \in h\}|.$$

We will consider hypergraphs generated according to the Erdős-Rényi model $G^{(k)}(n, q)$ in which every k-subset of V is present with probability q. We denote by $\overline{m}$ the expected

number of hyperedges in G under this generative model, i.e.:

$$\overline{m} = q\binom{n}{k}.$$

Note that, under this generative model, m is a random variable, while $\overline{m}$, n and k are deterministic quantities.

Problem Setup. This paper aims to build on the structure of Li et al. (2019), generalizing their results to hypergraphs. With that in mind, we briefly go over the setting; learning an unknown hypergraph generated via the Erdős-Rényi model. We note that after generation, our hypergraph, G, remains fixed, and we try to uncover G through a series of queries to an oracle. Where we ask if a set of vertices contains an edge.

Upon completion of querying, the results create a decoder that forms an estimate of G, $\hat{G}$. The focus of this paper is to find algorithms minimizing the amount of queries, while maintaining the probability the decoder recovers G be arbitrarily close to one.

Sparsity Level. As Li et al. (2019) limit their results to sparse graphs, we limit the scope of this paper to the standard notion of sparse hypergraphs in the following sense. We assume that $q = o(1)$ as $n \to \infty$, and throughout this paper we will set $q = \Theta\left(n^{-k(1-\theta)}\right)$ for some $\theta \in (0, 1)$, so that the average number of hyperedges $\bar{m} = \binom{n}{k}q$ behaves as $\Theta\left(n^{k\theta}\right)$. For efficient decoding results pertaining to Algorithm 1, we use a stronger notion of sparsity, where a superlinear number of edges are still allowed, but we further assume that $m = o(n^{\frac{k}{k-1}})$. We leave the question of tackling less sparse hypergraphs open.

Bernoulli Random Queries. We will often make use of Bernoulli queries, also known as Bernoulli tests. A Bernoulli query on a hypergraph $G = (V, E)$ is one in which the query is selected at random, by including each vertex $v \in V$ to be queried with a fixed probability p, independently of all other vertices. Following Li et al. (2019), we set $p = \sqrt[k]{\frac{k\nu}{qn^k}}$ for some constant $\nu > 0$, and we note that this choice of p gives $p^k = \frac{\nu}{m}(1 + o(1))$, since $\bar{m} = \frac{1}{k}qn^k(1 + o(1))$. Intuitively, this choice is made to guarantee that the probability of any given test being positive is not too close to 0 or to 1, thus making each test as informative as

possible. Given a fixed hypergraph G we will denote by P_G the probability that a Bernoulli test with parameter p as above is positive.

2. Typical Instances

In this section, we identify a set of typical instance arising from the random hypergraph model. This will allow us to make assumptions about the structure of the specific instance we are learning. We then use this to derive an information-theoretic lower bound on the query complexity of non-adaptively learning hypergraphs.

DEFINITION 1 (ε-typical Hypergraph Set). For any $\varepsilon > 0$, we define the ε-typical hypergraph set as the set $\mathcal{T}(\varepsilon)$ of hypergraphs G satisfying both of the following conditions:

(1) $(1-\varepsilon)\,\bar{m} \leq m(G) \leq (1+\varepsilon)\,\bar{m}$,

(2) $\Delta(G) \leq d_{\max}$,

(3) $(1-\varepsilon)\,(1-e^{-\nu}) \leq P_G \leq (1+\varepsilon)\,(1-e^{-\nu})$.

where:

$$d_{\max} = \begin{cases} kn^{k-1}q & \theta > \frac{1}{k} \\ \log n & \theta \leq \frac{1}{k}. \end{cases}$$

We now show that, for any $\varepsilon > 0$, $\Pr[G \in T(\varepsilon)] \to 1$ as $n \to \infty$, where the probability is taken over the random choice of G from $G^{(k)}(n,q)$. This key result is a hypergraph analogue of a similar result appearing in the paper of Li et al. (2019).

LEMMA 3. *For any $\varepsilon > 0$, we have:*

$$\Pr[G \in T(\varepsilon)] \to 1$$

as $n \to \infty$.

PROOF. We begin by noting that the set $\mathcal{T}(\varepsilon)$ can be written as $\mathcal{T}(\varepsilon) = \mathcal{T}^{(1)}(\varepsilon) \cap \mathcal{T}^{(2)}(\varepsilon) \cap \mathcal{T}^{(3)}(\varepsilon)$, where:

$$\mathcal{T}^{(1)} = \{G : (1-\varepsilon)\,\bar{m} \leq m(G) \leq (1+\varepsilon)\,\bar{m}\}$$

$$\mathcal{T}^{(2)} = \{G : \Delta(G) \leq d_{\max}\}$$

$$\mathcal{T}^{(3)} = \{G : (1-\varepsilon)\left(1-e^{-\nu}\right) \leq P_G \leq (1+\varepsilon)\left(1-e^{-\nu}\right)\}$$

It is then sufficient to show that $\Pr[G \in \mathcal{T}^{(i)}] \to 1$ for every $i = 1, 2, 3$.

Since $m(G)$ follows a binomial distribution with parameters $\binom{n}{k}$ and q, we have:

$$\Pr[(1-\varepsilon)\bar{m} \leq m(G) \leq (1+\varepsilon)\bar{m}] \to 1$$

as $\binom{n}{k} \to \infty$. This yields $\Pr[G \in \mathcal{T}^{(1)}] \to 1$.

We now establish that $\Pr[G \in \mathcal{T}^{(2)}] \to 1$:

(1) If $\theta > \frac{1}{k}$, then the combinatorial degree of each vertex follows a binomial distribution with mean $\binom{n-1}{k-1}q = \Theta(n^c)$ for some $c > 0$. We can then follow the work Li et al. (2019), using the Chernoff bound to show the probability of any degree exceeding $kn^{k-1}q$ goes to zero.

(2) If $\theta \leq \frac{1}{k}$, we note that we need only consider the case $\theta = \frac{1}{k}$, as this is when the probability for exceeding $\log n$ degree is highest. Here, we have that the combinatorial degree for a vertex follows a binomial distribution with $\binom{n-1}{k-1}$ trials and success probability $\Theta(\frac{1}{n^{k-1}})$, so the mean is $\Theta(1)$. From here we can once again follow the argument of Li et al. (2019), using the standard Chernoff bound to show that the probability of any vertex exceeding $\log n$ degree vanishes.

The last and most intensive argument is to establish $\mathcal{T}^{(3)}(\varepsilon)$. However, the hypergraph extension of this result is straightforward, we simply adapt the proof in the paper of Li et al. (2019), making note that the constant, two, used in the graph case becomes k in our new hypergraph setting. □

A simple consequence of Lemma 3 is that the algorithm-independent lower bound for the number of tests needed to obtain asymptotically vanishing probability provided in Li et al. (2019) holds for general hypergraphs.

THEOREM 4. *Under the typical instance setting discussed above, with $q = o(1)$ and an arbitrary non-adaptive test design, to have vanishing error probability we must have at least $\left(\bar{m}\log_2\frac{1}{q}\right)(1-\eta)$ queries, for arbitrarily small $\eta > 0$.*

PROOF. We have the following entropy inequality from Li et al. (2019):

$$P_{\mathrm{e}} \geq \mathbb{P}[\mathcal{A}]\frac{H(G \mid \mathcal{A} = \text{ true }) - I(G;\widehat{G} \mid \mathcal{A} = \text{ true }) - \log 2}{\log|\mathcal{G}_{\mathcal{A}}|},$$

where P_{e} is the probability of outputting the incorrect graph, $\mathcal{A}$ is the event that a graph satisfies condition one of the ε-typical hypergraph set and $\mathcal{G}_{\mathcal{A}}$ is the set of graphs such that $(1-\varepsilon)\bar{m} \leq m \leq \bar{m}(1+\varepsilon)$. From the typicality conditions we have:

- $\mathbb{P}[\mathcal{A}] = 1 - o(1)$
- $\log|\mathcal{G}_{\mathcal{A}}| = \binom{n}{k}H_2(q)(1+o(1))$
- $H(G \mid \mathcal{A} = \text{true }) = \binom{n}{k}H_2(q)(1+o(1))$, where $H_2(q) = q\log\frac{1}{q} + (1-q)\log\frac{1}{1-q}$ is the binary entropy function.

We also have from Li et al. (2019):

- $I(G;\widehat{G} \mid \mathcal{A} = \text{true }) \leq I(G;\mathbf{Y} \mid \mathcal{A} = \text{true }) \leq t\log 2$, where t represents the total amount of queries.

Together, yielding:

$$P_{\mathrm{e}} \geq \left(1 - \frac{t\log 2}{\binom{n}{k}H_2(q)}\right)(1+o(1)).$$

In our setting, $q \to 0$, thus $H_2(q) = \left(q\log\frac{1}{q}\right)(1+o(1))$, and hence

$$P_{\mathrm{e}} \geq \left(1 - \frac{t\log 2}{\frac{1}{k}qn^k\log\frac{1}{q}}\right)(1+o(1)).$$

Since $\bar{m} = \frac{1}{k}qn^k(1+o(1))$, we conclude that to have vanishing error probability we must have at least $\left(\bar{m}\log_2\frac{1}{q}\right)(1-\eta)$ queries, for arbitrarily small $\eta > 0$.

□

3. The HYPERGRAPH-GROTESQUE Algorithm

In this section we give a sublinear-time decoding algorithm for the problem of learning hypergraphs with hyperedge detection queries. As in the previous sections, we assume that the hypergraph is sampled according to the Erdős-Rényi model, and the probabilistic guarantees of the algorithm will depend on the randomness in both the algorithm and the hypergraph generative process.

We prove the main theorem:

THEOREM 2. *There exists an algorithm (Algorithm 1) that on input a hyperedge-detection oracle for an Erdős-Rényi hypergraph, makes $O(k\bar{m}\log^2\bar{m} + k^2\bar{m}\log\bar{m}\log^2 n)$ non-adaptive queries to the oracle, and outputs the correct answer with probability $\Omega(1)$. Here, the probability is taken over the randomness in both the algorithm and in the hypergraph. The algorithm requires $O(k\bar{m}\log^2\bar{m} + k^3\bar{m}\log\bar{m}\log^2 n)$ decoding time.*

Algorithm 1 HYPERGRAPH-GROTESQUE

Input: A hyperedge-detection oracle for a hypergraph G.

Output: A hypergraph $\hat{G} = (V, \hat{E})$.

Let $b = \Theta(\bar{m}\log\bar{m})$ be given as in Section 3.1.

Form bundles $B_1, ..., B_b$ by independently including each vertex v in each bundle B_i with probability $r_{inc} = \frac{1}{\sqrt[k]{2\bar{m}}}$.

Let $\delta^* \leftarrow O(\frac{1}{\bar{m}\log\bar{m}})$

Initialize $\hat{E} = \emptyset$.

for $i = 1, ..., b$ **do**

if Multiplicity_Test(B_i, δ^*) returns 1 **then**

Perform a location test (Section 3.3) on B_i, and add the resulting hyperedge h to $\hat{E}$.

end if

end for

Return $\hat{G} = (V, \hat{E})$

Algorithm 2 Multiplicity Test

Input: A bundle $B \subseteq V$, an error probability δ.
Output: An outcome in $\{0, 1\}$ indicating whether $\mathcal{B}$ contains a single hyperedge.

Let $M = \frac{1}{e}(1 - \frac{1}{\sqrt[k]{e}})$.
Perform $t_{mul} = 2\log(2/\delta)/M^2$ edge detection queries on B chosen according to a Bernoulli design with parameter $r_{mul} = 1/\sqrt[k]{e}$. Let $\hat{p}$ be the fraction of queries that return a positive outcome (i.e. the ones for which the set being queried contains a full hyperedge).
if $\hat{p} \in (0, 1/e + M/2)$ **then**
    Return 1
else
    Return 0
end if

Similarly to the algorithm of Li et al. (2019), our algorithm is inspired by the GROTESQUE procedure first introduced by Cai et al. (2017). In particular, the algorithm is structured according to the following high-level framework:

(1) In the first step, the algorithm produces random sets of vertices (bundles), obtained by including each vertex in each set independently with a fixed probability. This step is successful if each hyperedge is the unique hyperedge in at least one of the bundles. By a coupon-collector argument, one can bound from below the probability of this step succeeding when the number of bundles is sufficiently large (Section 3.1).

(2) Then, the algorithm performs *multiplicity tests* on each of the sets to identify the ones that contain a unique hyperedge. This works by estimating the probability of a Bernoulli test detecting a hyperedge within a bundle $\mathcal{B}$, and then using this estimate to determine whether the bundle really contains a single hyperedge. This step is successful if every multiplicity test correctly identifies whether a bundle contains a single hyperedge. By applying standard sampling results, it can be shown that, if sufficiently many Bernoulli tests are made, this step is successful with high probability (Section 3.2).

(3) Finally, the algorithm performs a *location test* on the sets that passed the multiplicity test, which identifies the unique hyperedge the set contains. This step is successful if every location test correctly identifies the unique hyperedge in a bundle.

We show that this step can be performed by leveraging a reduction to the standard group testing problem (Section 3.3).

It is not hard to see that if all three steps are successful, one can reconstruct the hypergraph G correctly from the result of the queries.

We note that, while the procedure above is described sequentially, all of the tests needed to carry it out can be performed non-adaptively.

We will now analyze each step in detail. After that, we complete the proof of Theorem 2.

3.1. Bundles of Tests. Recall that Algorithm 1 forms a number $b = \Theta(\bar{m} \log \bar{m})$ of bundles of vertices, where each node is placed independently in each bundle with probability $r_{inc} := 1/\sqrt[k]{2m}$.

We say a hyperedge is *fully contained* in a bundle B if all of the vertices in the hyperedge have been placed in B. Intuitively, the random process of forming the bundles is successful if, for every hyperedge h, there exists a bundle B_i such that h is the *unique* hyperedge that is fully contained in B_i. We prove the following lemma:

LEMMA 5. *Let G be any k-uniform hypergraph. Suppose that the vertices of G are placed into bundles according to the procedure described in Algorithm 1. For any fixed hyperedge $h \in G$ and any fixed bundle B_i, let $\mathcal{E}_{h,i}$ be the event that h is the only hyperedge fully contained in B_i. Then:*

$$\Pr[\mathcal{E}_{h,i}] \geq \left(1 - r_{inc} k \Delta(G) - m(G) r^k\right) r_{inc}^k.$$

PROOF. We consider three events A_0, A_1 and A_2 defined as follows:

- A_0 is the event that the hyperedge h is fully contained in B_i,
- A_1 is the event that there exists some hyperedge $h' \neq h$ satisfying $h' \cap h \neq \emptyset$ that is fully contained in B_i,
- A_2 is the event that there exists some hyperedge h' satisfying $h' \cap h = \emptyset$ that is fully contained within B_i.

By definition, we have $\mathcal{E}_{h,i} = A_0 \cap \overline{A}_1 \cap \overline{A}_2$. Note that:

$$\Pr[A_0] = r_{inc}^k,$$

while by the union bound, we have:

$$\Pr[A_1 \mid A_0] \leq \sum_{\substack{h' \in E \setminus \{h\} \\ h' \cap h \neq \emptyset}} r_{inc} \leq r_{inc} k \Delta(G),$$

and:

$$\Pr[A_2 \mid A_0] = \Pr[A_2] \leq \sum_{\substack{h' \in E \\ h' \cap h = \emptyset}} r_{inc}^k = m(G) r_{inc}^k.$$

We then have:

$$\begin{aligned}
\Pr[\mathcal{E}_{h,i}] &= \Pr[A_0 \cap \bar{A}_1 \cap \bar{A}_2] = \Pr[\bar{A}_1 \cap \bar{A}_2 \mid A_0] \Pr[A_0] \\
&= (1 - \Pr[A_1 \cup A_2 \mid A_0])\, r_{inc}^k \\
&\geq (1 - \Pr[A_1 \mid A_0] - \Pr[A_2 \mid A_0])\, r_{inc}^k \\
&= \left(1 - r_{inc} k \Delta(G) - m(G) r^k\right) r_{inc}^k.
\end{aligned}$$

□

This in turn gives the following result.

LEMMA 6. *When the* ***HYPERGRAPH-GROTESQUE*** *algorithm is run on a hypergraph G sampled according to an Erdős-Rényi model for sufficiently large values of n, the probability that every hyperedge h is the unique hyperedge in some bundle of tests satisfies:*

$$\Pr\left[\bigcap_{h \in E} \bigcup_{i \in [b]} \mathcal{E}_{h,i}\right] \geq 1 - \delta.$$

PROOF. Let $m = m(G)$. For every fixed h and i, by Lemma 5:

$$\Pr[\mathcal{E}_{h,i}] \geq \left(1 - r_{inc} k \Delta(G) - m r_{inc}^k\right) r_{inc}^k$$

$$= \left(1 - r_{inc}\frac{k^2 m}{n} - \frac{1}{2}\right)\frac{1}{2m}$$
$$= \frac{1}{4m}(1 + o(1)), \tag{1}$$

where we are using the fact that $n = \omega(m^{1-\frac{1}{k}})$. Hence:

$$\begin{aligned}
\Pr\left[\overline{\bigcap_{h\in E}\bigcup_{i\in[b]}\mathcal{E}_{h,i}}\right] &= \Pr\left[\bigcup_{h\in E}\bigcap_{i\in[b]}\overline{\mathcal{E}_{h,i}}\right] \\
&\leq \sum_{h\in E}\Pr\left[\bigcap_{i\in[b]}\overline{\mathcal{E}_{h,i}}\right] \\
&= \sum_{h\in E}\prod_{i\in[b]}\Pr\left[\,\overline{\mathcal{E}_{h,i}}\,\right] \\
&\leq \sum_{h\in E}\prod_{i\in[b]}\left(1 - \frac{1}{4m}(1 + o(1))\right) \\
&= m\left(1 - \frac{1}{4m}(1 + o(1))\right)^b \\
&\leq me^{-\frac{b}{4m}(1+o(1))},
\end{aligned}$$

where we first applied De Morgan's law, then the union bound, then the fact that for every h the random variables $\{\mathcal{E}_{h,i}\}_{i\in[b]}$ are mutually independent, then Equation (1). The result then follows. □

3.2. Multiplicity Test. We now discuss the guarantees of the multiplicity test.

DEFINITION 2. Given a set $B \subseteq V$, a (r_{mul}, t_{mul})-multiplicity test for B is a collection of t_{mul} tests on the elements of B chosen according to a Bernoulli design with parameter r_{mul}. The test returns 1 if the fraction of positive tests suggests that a single hyperedge is present in the bundle and 0 otherwise.

In order to analyze the multiplicity test (Algorithm 2), we use the following lemma.

LEMMA 7. *Suppose that a set $B \subseteq V$ contains multiple hyperedges. Let S be a subset chosen according to a Bernoulli design with parameter $1/\sqrt[k]{e}$ (i.e. by including each $v \in B$*

into S independently with probability $1/\sqrt[k]{e}$). Then the probability that S contains a full hyperedge is at least: $2/e - 1/e^{(k+1)/k}$.

PROOF. By assumption, the set $\mathcal{B}$ contains at least two distinct hyperedges. Fix two distinct hyperedges h and h'. Then let:

- D be the event that the set S contains a full hyperedge $\mathcal{B}$,
- D_h be the event that S contains h, i.e. $h \subseteq S$
- $D_{h'}$ be the event that S contains h', i.e. $h' \subseteq S$,
- $D_{h\cap h'}$ be the event that S satisfies $(h \cap h') \subseteq S$,
- a be the cardinality of the intersection of h and h', i.e. $a := |h \cap h'|$. Note that a is some integer between 0 and $k-1$.

Since each element of $\mathcal{B}$ is included in S independently, the events D_h and $D_{h'}$ are conditionally independent given $D_{h\cap h'}$. We then have:

$$\begin{aligned}
\Pr[D] &\geq \Pr[D_h \cup D_{h'}] = \Pr[D_h \cup D_{h'} \mid D_{h\cap h'}] \Pr[D_{h\cap h'}] \\
&= \left(1 - \Pr[\overline{D_h} \cap \overline{D_{h'}} \mid D_{h\cap h'}]\right) \Pr[D_{h\cap h'}] \\
&= \left(1 - \Pr[\overline{D_h} \mid D_{h\cap h'}] \cdot \Pr[\overline{D_{h'}} \mid D_{h\cap h'}]\right) \left(\frac{1}{\sqrt[k]{e}}\right)^a \\
&= \left[1 - \left(1 - \left(\frac{1}{\sqrt[k]{e}}\right)^{k-a}\right)^2\right] \left(\frac{1}{\sqrt[k]{e}}\right)^a \\
&= \frac{2}{e} - \left(\frac{1}{e}\right)^{2k-\frac{a}{k}} \geq \frac{2}{e} - \left(\frac{1}{e}\right)^{k+\frac{1}{k}},
\end{aligned}$$

as needed. □

This then yields the following guarantee on correctness:

LEMMA 8. *Suppose we run a multiplicity test on a bundle B with error probability parameter δ. Then:*

(1) if B contains no hyperedge, the answer is always 0,

(2) if B contains a single hyperedge, the test returns 1 with probability at least $1 - \delta$,

(3) and if B contains more than one hyperedge, the test returns 0 with probability at least $1 - \delta$.

PROOF. If the bundle contains no hyperedge, then the fraction of positive tests will be zero and the algorithm will return 0 every time. Otherwise, Let R_0, R_1 be the events that the multiplicity test returns 0 and 1 respectively. If the bundle contains a single hyperedge, the probability that any given edge-detection test returns 1 is $p_{single} = (e^{-k})^k = e^{-1}$. Applying Lemma 1, we obtain:

$$\Pr[R_1] \geq \Pr\left[\left|\hat{p} - \frac{1}{e}\right| < \frac{M}{2}\right] \geq 1 - 2e^{-t_{mul}M^2/2} = 1 - \delta.$$

On the other hand, if the bundle contains at least two hyperedges, by Lemma 7 the probability $p_{multiple}$ that any individual test detects a hyperedge satisfies:

$$p_{multiple} \geq \frac{2}{e} - \frac{1}{e^{(k+1)/k}}. \tag{2}$$

Hence, in this case:

$$\Pr[R_0] \geq \Pr\left[|\hat{p} - p_{multiple}| < M\right] \leq e^{-t_{mul}M^2/2} = 1 - \delta.$$

as needed. □

We also obtain the following guarantee on the efficiency of the multiplicity tests:

LEMMA 9. *The number of queries made by a multiplicity test with error parameter δ is at most $e^3 k \log \frac{2}{\delta}$, and the decoding time for each multiplicity test is $O(k \log \frac{1}{\delta})$.*

PROOF. The number of queries made is:

$$t_{mul} = 2 \log \frac{2}{\delta} e^2 \frac{e^{2/k}}{\sqrt[k]{e} - 1} \leq e^3 k \log \frac{2}{\delta},$$

The decoding time is proportional to the number of queries. □

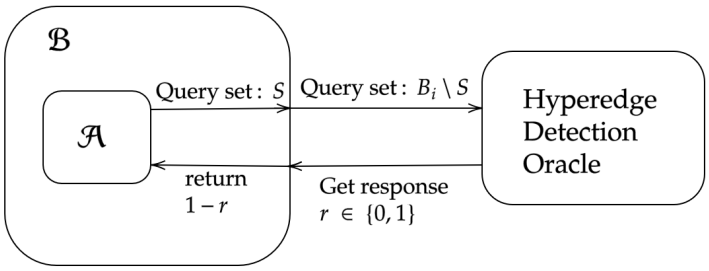

FIGURE 1. The structure of the reduction in the proof of Lemma 10. Here, the algorithm $\mathcal{B}$ is given access to a hyperedge-detection oracle. $\mathcal{B}$ simulates algorithm $\mathcal{A}$ and converts $\mathcal{A}$'s queries into hyperedge detection queries.

3.3. Location Test via Reduction to Group Testing. Once the algorithm has performed all the multiplicity tests, it runs location tests on the bundles that passed the multiplicity tests. Executing a location test on a bundle that contains a single hyperedge h allows the algorithm to *discover* h and add it to the estimate hypergraph $\hat{H}$.

We obtain a location test by highlighting an equivalence between the problem of learning a single hyperedge of arity k using a hyperedge-detection oracle, and that of group testing with k defective items.

LEMMA 10. *Any algorithm for the group testing problem with k faulty items yields an algorithm for the problem of learning a hypergraph known to have a single hyperedge of arity k by making edge-detection queries. Conversely, any algorithm for the latter problem yields an algorithm for the former. These reductions preserve query complexity, error probability, and decoding time.*

PROOF. Consider the following reduction from the latter problem to the former. Suppose $\mathcal{A}$ is an algorithm that solves the group testing problem: i.e. given a finite set V which contains a subset K of defective items, $\mathcal{A}$ submits queries of the form $S \subseteq V$ to an oracle that determines whether $S \cap K \neq \emptyset$, and based on the answer to those queries, it recovers K.

Now, consider the problem of learning a cardinality-k hyperedge h on V by making hyperedge-detection queries. We design an algorithm $\mathcal{B}$ for the latter problem as follows: $\mathcal{B}$ simulates $\mathcal{A}$ and whenever $\mathcal{A}$ submits a query $S \subseteq V$, $\mathcal{B}$ instead submits the query $\overline{S}$ to the hyperedge-detection oracle, and then returns to $\mathcal{A}$ the opposite $(1 - r)$ of the answer r it receives. When $\mathcal{A}$ terminates, outputting a set S^*, $\mathcal{B}$ outputs the same set.

For each query S made by $\mathcal{A}$ the value of $1 - r$ is equal to 1 if and only if the set S contains at least one element of the hidden hyperedge h. Hence, from the perspective of $\mathcal{A}$, $\mathcal{B}$ is implementing a group testing oracle for an instance in which $K = h$. In particular, if $\mathcal{A}$ correctly solves the group testing problem, the output S^* of $\mathcal{B}$ is equal to h.

It is easy to see that an analogous reduction can be used to reduce from the group testing problem to that of learning a single hyperedge, and hence the two problems are entirely equivalent. □

Note that this reduction preserves query complexity, adaptivity, and runtime guarantees.

The group testing problem is well-studied in the literature and in particular the following result is known.

THEOREM 11 (Paraphrasing Theorem 11 from Cheraghchi and Ribeiro (2019)). *Consider the standard group testing problem on n element with k defective elements. There exists a(n explicitly constructable) collection of $O(k^2 \log^2 n)$ group tests and an algorithm $\mathcal{A}$ which, given the results of the tests as input, outputs the set of defective items in $O(k^3 \log^2 n)$ time.*

The result of Cheraghchi and Ribeiro is based on a construction of linear codes with fast decoding time described in the same paper.

By Lemma 10, the above result implies:

COROLLARY 12. *Consider the problem of learning a hypergraph known to consist of a single hyperedge of arity k by non-adaptively making queries to a hyperedge-detection oracle. There exists an algorithm for this problem which makes $O(k^2 \log^2 n)$ queries and requires decoding time $O(k^3 \log^2 n)$.*

The algorithm guaranteed by Corollary 12 is simply the group testing algorithm of Cheraghchi and Ribeiro (2019) run through the reduction used in the proof of Lemma 10.

3.4. Proof of Theorem 2.

Proof of Theorem 2. By Lemma 6, if we create $b = \Theta(\bar{m} \log \bar{m})$ bundles of vertices, and assign each vertex to a bundle at random with probability $r_{inc} = 1/\sqrt[k]{2\bar{m}}$, every hyperedge is the unique hyperedge in some bundle with constant probability.

The algorithm then runs a multiplicity test with error probability $\delta^* = \Theta\left(\frac{1}{\bar{m} \log \bar{m}}\right)$ on every bundle. By Lemma 8 and the union bound, there is a constant probability that every multiplicity test succeeds.

By Lemma 9, this requires $O(k \log \bar{m})$ queries and $O(k \log \bar{m})$ decoding time for every bundle, which amounts to a total of $O(k\bar{m} \log^2 \bar{m})$ queries and decoding time to establish which bundles contain a single hyperedge. We then need to run $\bar{m} \log \bar{m}$ location tests[3], each of which requires $O(k^2 \log^2 n)$ queries and $O(k^3 \log^2 n)$ decoding time. The total then equals:

$$O(k\bar{m} \log^2 \bar{m} + k^2 \bar{m} \log \bar{m} \log^2 n))$$

queries and:

$$O(k\bar{m} \log^2 \bar{m} + k^3 \bar{m} \log \bar{m} \log^2 n))$$

decoding time as needed (Recall that $m = \Theta(\bar{m})$ with probability that tends to 1 as $n \to \infty$). □

While the results above are stated with success probability $\Omega(1)$, this is not a fundamental limitation of our algorithms. The success probability of Algorithm 1 can be boosted to $1 - \delta$ for any $\delta > 0$ by adjusting the algorithm's parameters. Concretely,

[3]Note that prior to running a location test on a bundle B, the algorithm can check whether B contains a previously discovered hyperedge h. If that is the case (i.e. if $h \in B$), then B could be ignored, and the algorithm would not run a location test on it. This allows one to guarantee that in a successful run of the algorithm, no more than m location tests are run. However performing this check comes at an extra computational cost, and it is not clear that it can be carried out efficiently. In the paper of Li et al. (2019) the authors do not discuss this issue and rather just assume that one is able to run at most m location tests through the run of the algorithm. By doing this, they remove the factor of $\log m$ from the second term in the above bound

Lemma 6 holds for arbitrary δ, requiring $b = \Theta(\bar{m}\log(\bar{m}/\delta))$ bundles rather than $\Theta(\bar{m}\log\bar{m})$. Running a multiplicity test with error parameter $\delta^* = \Theta(\delta/(\bar{m}\log(\bar{m}/\delta)))$ on each bundle and union bounding gives overall failure probability at most δ from the multiplicity tests. Tracing through Lemmas 6, 7, 8, and 9, the total query complexity becomes $O(k\bar{m}\log^2(\bar{m}/\delta) + k^2\bar{m}\log(\bar{m}/\delta)\log^2 n)$. In particular, setting $\delta = 1/\text{poly}(\bar{m})$ yields high probability guarantees with no asymptotic overhead, since $\log(\bar{m}/\delta) = O(\log\bar{m})$ in this case and the query complexity reduces to $O(k\bar{m}\log^2\bar{m} + k^2\bar{m}\log\bar{m}\log^2 n)$, matching the original bound. An analogous argument applies to the `COMP`, `DD`, and `SSS` algorithms, where increasing t by an $O(\log(1/\delta))$ factor drives the failure probability to δ.

4. Other Algorithmic Results

This section presents hypergraph analogues of popular group testing algorithms, building upon the results given by Li et al. (2019) in the context of graphs. We also provide formal guarantees on the query complexity and success probability of the algorithms we describe, showing that these algorithms have a better query complexity ($O(k\bar{m}\log n)$) than Algorithm 1 (at the price of longer decoding time).

The three algorithms we adapt are "Combinatorial Orthogonal Matching Pursuit" `COMP`, "Definite Defectives" `DD`, and "Smallest Satisfying Set" `SSS`. The `COMP` algorithm for group testing simply rules out all of the elements that have appeared in any negative tests and returns the remaining elements. The `DD` algorithm, first rules out all elements that appear in any negative test, then outputs all the elements that must be defective out of the remaining elements. `SSS` simply returns a satisfying assignment of minimum cardinality. We refer the reader to the survey of Aldridge et al. (2019) for a review of how these algorithms are used in group testing.

All three of these algorithms produce an estimate $\hat{G}$ of the hypergraph G based on the result of a single batch of Bernoulli queries. In particular, we will assume that each algorithm takes as input a collection $\{X^{(i)}\}_{i\in[t]}$ of hyperedge-detection queries, where each $X^{(i)} \subseteq V$ is chosen according to a Bernoulli design with parameter $p = \sqrt[k]{\frac{k\nu}{qn^k}}$ (for some ν to be defined).

We also assume the algorithms have access to the results $\{Y^{(i)}\}_{i\in[t]}$ of the queries, where:

$$Y^{(i)} = \begin{cases} 1 & \text{if there exists } h \in H \text{ s.t. } h \subseteq X^{(i)} \\ 0 & \text{otherwise.} \end{cases}$$

Since the algorithms themselves are deterministic, all of the probabilistic guarantees are based on the randomness in both the choice of Bernoulli queries and the hypergraph generation process.

The COMP Algorithm. The first algorithm we examine is COMP (Algorithm 3). The key observation behind this algorithm is the following: no collections h of k vertices can be a hyperedge in G if all the vertices in h appear in some query $X^{(i)}$ with $Y^{(i)} = 0$. The algorithm then simply assumes each hyperedge h is present in G unless it satisfies this condition.

Algorithm 3 COMP

Input: A hyperedge-detection oracle for a hypergraph G, t hyperedge-detection queries $\{X^{(1)}, \ldots, X^{(t)}\}$, and the results $\{Y^{(1)}, \ldots, Y^{(t)}\}$ of the queries.
Output: A hypergraph $\widehat{G} = (V, \widehat{E})$.
Initialize $\widehat{E}$ to contain all $\binom{n}{k}$ edges
for each i such that $Y^{(i)} = 0$ **do**
  Remove all h from $\widehat{E}$ satisfying $h \subseteq X^{(i)}$
end for
return $\widehat{G} = (V, \widehat{E})$

We obtain the following guarantees on the performance of COMP.

THEOREM 13. *If COMP is given as input an unknown hypergraph G sampled from $G^{(k)}(n, q)$ that is in the typical instance setting, where we have $q = \Theta\left(n^{k(\theta-1)}\right)$ for some $\theta \in (0, 1)$, as well as at least $t = ke \cdot \bar{m} \log n$ Bernoulli queries with parameter $\nu = 1$, then it outputs $\hat{G} = G$ with probability $\Omega(1)$.*

PROOF. We will adapt the proof of Li et al. (2019) of the analogous theorem for graphs. We note that conditioning on the random graph being in the typical set of graphs with high probability, we need only show that using the above stated amount of tests yields error probability approaching zero. So we examine the probability of failing to identify a non-edge,

say $(i_1, \ldots, i_k) \notin E$, in the hypergraph setting this probability changes slightly. Recall, that there are two ways a test could fail to identify a non-edge:

(1) at least one vertex in $(i_1, \ldots, i_k)$ isn't present in the test
(2) $(i_1, \ldots, i_k)$ is contained in the test, but another edge of G, our hypergraph, is also present in the test.

This results in the probability of failing to identify $(i_1, \ldots, i_k)$ as a non-edge as

$$p_{ne} = (1 - p^k) + p^k P_G[\{i_1, \ldots, i_k\} \subseteq \mathcal{L}],$$

where we recall that $\mathcal{L}$ is the set of nodes in the test and p is the probability of inclusion in the test, and $P_G[\{i_1, \ldots, i_k\} \subseteq \mathcal{L}]$ is the probability of a positive Bernoulli test, given $\{i_1, \ldots, i_k\}$ is included in the test. If we are to have a positive test, we have either

(1) An edge $e = (e_1, \ldots e_k) \in E$ such that $e \cap (i_1, \ldots, i_k) \neq \emptyset$,
(2) An edge $e = (e_1, \ldots e_k) \in E$ such that $e \cap (i_1, \ldots, i_k) = \emptyset$,

included in the test. In case 1, we can examine the situation $|e \cap (i_1, \ldots, i_k)| = 1$, noting that if we examine all possible neighbors, the rest of $(i_1, \ldots, i_k)$ is included in that set.

$$\begin{aligned} P_G[\{i_1, \ldots, i_k\} \subseteq \mathcal{L}] &\leq \mathbb{P}\left[e \cap (i_1, \ldots, i_k) = 1 \mid \{i_1, \ldots, i_k\} \subseteq \mathcal{L}\right] + \mathbb{P}\left[e \cap (i_1, \ldots, i_k) = \emptyset \mid \{i_1, \ldots, i_k\} \subseteq \mathcal{L}\right] \\ &\leq p^{k-1} k \Delta(G) + P_G \end{aligned}$$

In the first term, there are at most k choices of vertices to intersect with, then $\Delta(G)$ possible edges containing that vertex. Since the second event is independent of conditioning on $\{i_1, \ldots, i_k\} \subseteq \mathcal{L}$, we get that the probability is just P_G. We assumed our graph is in the typical set so we can substitute in $P_G = (1 - e^{\nu})(1 + o(1))$, we also have that $\Delta(G) k p^{k-1} = o(1)$, so our probabilities in the hypergraph case align with those in Li et al. (2019). We note that in the hypergraph case our bound changes slightly over union-bounding over a possible $\binom{n}{k}$ non-edges, resulting in

$$\mathbb{P}[\text{ error }] \leq n^k e^{-\frac{t}{em}(1+o(1))}.$$

After re-arranging we have that $\mathbb{P}[\text{ error }] \to 0$ as long as

$$t \geq kem \log n(1+\eta)$$

for arbitrarily small $\eta > 0$. Since $m = \bar{m}(1+o(1))$ for all typical graphs, and the probability that G is typical tends to one, we obtain the condition in our statement. □

The DD Algorithm. COMP's method of assuming edges are present until proven otherwise may be rather inefficient since we are looking for a sparse graph The DD algorithm reverses this assumption and starts with all edges as non-edges, making use of COMP to preclude non-edges.

Algorithm 4 DD

Input: A hyperedge-detection oracle for a hypergraph G, t sets of vertices, $\{X^{(1)}, \ldots, X^{(t)}\}$, to be queried, with oracle given binary responses, $\{Y^{(1)}, \ldots, Y^{(t)}\}$.
Output: A hypergraph $\widehat{G} = (V, \widehat{E})$.
Initialize $\widehat{E} = \emptyset$, and initialize a potential edge set, PE, to contain all $\binom{n}{k}$ edges
for each i such that $Y^{(i)} = 0$ **do**
  Remove all edges from PE whose nodes are all in $X^{(i)}$
end for
for each i such that $Y^{(i)} = 1$ **do**
  If the nodes from $X^{(i)}$ cover exactly one edge in PE, add that edge to $\widehat{E}$
end for
return $\widehat{G} = (V, \widehat{E})$.

Proof. We again adapt the proof in Li et al. (2019) of an analogous theorem for graphs. Once again, we have a hypergraph in the typical set, so we need only show that with the stated number of queries our error probability goes to zero.

There are two steps to the DD algorithm, the first in which we find a set of 'potential' edges, this set could include non-edges. Then the second where we find the set of edges from our set of potential edges. The argument examines how large t, the number of queries, must be for each of these events to occur with high probability. We adapt slightly the two events in the first step of the proof in Li et al. (2019).

- H_0 is the total number of non-edges in the potential edge set, PE

- H_1 is the number of non-edges in PE such that at least one of its vertices form a part of at least one true edge.

We have that the amount of non-edges must be less than n^k, and then utilizing the probability of failing to identify a non-edge recorded in the `COMP` algorithm proof above, we have

$$\mathbb{E}\left[H_0\right] \leq n^k e^{-\frac{t}{em}(1+o(1))}.$$

The amount of non-edges sharing a vertex with an edge is upper-bounded by mkn^{k-1}, but still must be less than n^k, so we have

$$\mathbb{E}\left[H_1\right] \leq \min\left\{mkn^{k-1}, n^k\right\} e^{-\frac{t}{em}(1+o(1))}.$$

Applying Markov's inequality, we have for any $\xi_0 > 0$ and $\xi_1 > 0$ that

$$\mathbb{P}\left[H_0 \geq n^{k\xi_0}\right] \leq n^{k(1-\xi_0)} e^{-\frac{t}{em}(1+o(1))} \tag{3}$$

$$\mathbb{P}\left[H_1 \geq n^{k\xi_1}\right] \leq \min\left\{mkn^{k-1}, n^k\right\} n^{-k\xi_1} e^{-\frac{t}{em}(1+o(1))}. \tag{4}$$

After re-arranging we find that these two probabilities go to zero as $n \to \infty$ as long as

$$t \geq \left(k\left(1-\xi_0\right) em \log n\right)(1+\eta),$$

$$t \geq (1+\eta) em \log n \times \begin{cases} k\left(1-\xi_1\right) & \frac{1}{k} \leq \theta < 1 \\ k\left(1+\theta-\xi_1\right) - 1 & 0 < \theta \leq \frac{1}{k} \end{cases}$$

for arbitrarily small $\eta > 0$. The first case uses the n^k term in the $\min\{\cdot\}$ term in 4. The second case uses the mkd term in the $\min\{\cdot\}$ term and that $m = \Theta(n^{k\theta})$ and when $\theta > \frac{1}{k}$, $d_{max} \leq kn^{k-1}q = kn^{k\theta-1}$, the last case we look at when $theta \leq \frac{1}{k}$, so $d_{max} = O(\log n)$.

We show that $H_0 = o(m)$ and $H_1 = o(\sqrt{m})$ (with high probability). By setting ξ_0 to be arbitrarily close to (but still less than) θ, and similarly ξ_1 arbitrarily close to $\theta/2$, the above

requirements simplify to

$$t \geq (k(1-\theta)em\log n)(1+\eta),$$

$$t \geq (1+\eta)em\log n \times \begin{cases} k - k\theta/2 & \frac{1}{k} \leq \theta < 1 \\ k + k\theta/2 - 1 & 0 < \theta \leq \frac{1}{k} \end{cases}$$

for arbitrarily small $\eta > 0$.

The second step of the algorithm can be shown to succeed as long as

$$t \geq (k\theta em\log n)(1+\eta)$$

for arbitrarily small $\eta > 0$. This follows directly from the proof of the second step written by Li et al. (2019).

□

THEOREM 14. *If we have an unknown Erdős-Rényi hypergraph that is in the typical instance setting, where we have* $q = \Theta\left(n^{k(\theta-1)}\right)$ *for some* $\theta \in (0,1)$*, and Bernoulli testing with parameter* $\nu = 1$*, then with at least* $k\max\{\theta, 1-\theta, 1-\theta/2, 1+\theta/2-1/k\}e \cdot \bar{m}\log n$ *non-adaptive queries* `DD` *outputs the correct answer with probability* $\Omega(1)$*.*

The `SSS` Algorithm. The `SSS` algorithm works by finding the smallest set of edges such that the output is consistent with the Bernoulli test results, i.e. $\{Y^{(i)}\}_{i\in[t]}$. Since `SSS` searches for the minimal satisfying graph, it gives a lower bound to the size of the output of any Bernoulli-queries-based decoding algorithm.

Algorithm 5 `SSS`

Input: A hyperedge-detection oracle for a hypergraph G, t sets of vertices, $\{X^{(1)}, \ldots, X^{(t)}\}$, to be queried, with oracle given binary responses, $\{Y^{(1)}, \ldots, Y^{(t)}\}$.
Output: A hypergraph $\widehat{G} = (V, \widehat{E})$.
Find $\widehat{E}$ such that $|\widehat{E}|$ is minimized while satisfying $\{Y^{(1)}, \ldots, Y^{(t)}\}$
return $\widehat{G} = (V, \widehat{E})$.

THEOREM 15. *If we have an unknown Erdős-Rényi hypergraph that is in the typical instance setting, where we have* $q = \Theta\left(n^{k(\theta-1)}\right)$ *for some* $\theta \in (0,1)$*, and Bernoulli testing with an arbitrary choice of* $\nu > 0$*, then with at least* $k\theta e \cdot \bar{m} \log n$ *non-adaptive queries the* `SSS` *algorithm outputs the correct answer with probability* $\Omega(1)$*.*

PROOF. The proof of the `SSS` algorithm bound in the hypergraph case follows extremely closely to the graph case, where the main difference is simply replacing the number of vertices with general arity term k. We will just verify assumptions hold that are slightly altered in our setup. In the hypergraph case, event A_1 is just the event that another hyperedge intersects with the hyperedge we're seeking to find, say $(i_1, \dots i_k)$, and masking it. Therefore,

$$\mathbb{P}\left[A_1^{(i_1,\dots i_k)} \mid \{i_1, \dots i_k\} \subseteq \mathcal{L}\right] \leq \Delta(G)(1+p)^k$$

and defines $\xi' = \Delta(G)(1+p)^k$. We recall that the converse bound we are trying to prove is $t = \Omega(m \log n)$, so we can assume without loss of generality that $t = \Theta(m \log n)$, as additional tests only improve the `SSS` algorithm. From Li et al. (2019) we can assume without loss of generality that $p^k = \Theta\left(\frac{1}{m}\right)$, since if p^k behaves as $o\left(\frac{1}{m}\right)$ or $\omega\left(\frac{1}{m}\right)$ then the probability of a positive test tends to 0 or 1 as $n \to \infty$, and it follows from a standard entropy-based argument that $\omega(m \log n)$ tests are needed. We claim that these conditions imply that

$$\frac{e^{-2t\left(p^k\xi + O\left(p^{2k}\right)\right)}}{e^{2tp^k\xi'}} \to 1$$

We note that ξ' still behaves as $O\left(n^{-c}\right)$ for sufficiently small c. We also note that we have the same behavior for $\xi = (1 + k\Delta(G))p^k$. This is seen by noting that $tp^k = \Theta(\log n)$ by the above-mentioned behavior of t and p^k. This behavior fall inline with the $O\left(tp^{2k}\right)$ term by the above-mentioned behavior of t and p^k, and is seen to also hold for ξ and ξ' by noting that $\Delta(G)p^{k-1} = \Theta\left(\frac{\Delta(G)\sqrt[k]{m}}{m}\right)$, along with $\Delta(G) = O(\max\{\log n, n^{k-1}q\})$, $m = \Theta(n^k q)$, and the behavior of q. Thus, nothing of consequence changes when generalizing to hypergraphs.

□

5. Open Problems

The main open problem remains to improve the sparsity level of the low decoding time hypergraph learning `HYPERGRAPH-GROTESQUE` or to show that the sparsity assumption is necessary. Another direction is to improve its decoding time, which seems very likely to at least be possible with respect to logarithmic factors.

CHAPTER 3

Optimizing Recalculation Schedules to Minimize Regret

This chapter was previously published as Optimizing Recalculation Schedules to Minimize Regret by Bethany Austhof and Lev Reyzin (Austhof and Reyzin, 2024)

1. Introduction to Online Regret Schedules

In many online settings, one faces the problem on when to recalculate a given solution based on new data. This phenomenon occurs in various settings by different names: in the bandit framework, "sticky" decisions prevent the learner from switching arms without paying a cost Dekel et al. (2014); Kash et al. (2022); Machado et al. (2018), in the online clustering setting every time one switchs solutions, one may need to pay for new centers Bhattacharjee et al. (2023); Bhattacharjee and Moshkovitz (2021); Hess et al. (2021); Moshkovitz (2021), in facility location, adding a new facility adds cost Fotakis (2008); Meyerson (2001).

The specific details of different problems require the analysis of their respective structures and often have their own involved solutions. Here, we attempt to take a broader view, and we adopt the learning language of additive regret (instead of "competitive ratios," etc. from the streaming literature, though the results apply just as well there).

We assume we have a black-box algorithm $\mathcal{A}$ that produces an estimator with expected **per-round regret** of $\mathcal{R}_t(x) \in [0,1]$ on a new data-point x when given samples $S_t = \{x_1 \dots x_t\}$ and when point x is drawn from the same distribution as $x_1, \dots x_t$, which we will generically refer to as D. Given the assumption above, we will drop the argument from the function $\mathcal{R}$. We also define $\mathcal{R}_0 = 1$, which means that until samples are given to $\mathcal{A}$, full regret is suffered by its (lack of) solution.

We call giving samples $\{x_1, \dots x_t\}$ to $\mathcal{A}$ a **recalculation** at round t. We work in the setting where calls to $\mathcal{A}$ are expensive and need to be traded off against the benefits to

improving regret. A call to $\mathcal{A}$ involves recalculating, and possibly committing to, a new solution to a given problem. For example, in the case of facility location, it involves the costly opening of new facilities (or the moving of old ones).

Hence, we are interested in understanding the optimal recalculation strategy as minimize cumulative regret for various budgets on calls to $\mathcal{A}$. Given a strategy that recalculates at rounds $C = \{t_1, t_2, \ldots\}$, where each t_i represents a sequential point drawn from the distribution, the **expected cumulative regret** $\mathcal{R}^T$ would be

$$\mathcal{R}^T = \sum_{i=1}^{T} \max_{t \in C \text{ s.t. } t<i} \mathcal{R}_t \tag{5}$$

In this paper, we will consider per-round regret guarantees of the form $t^{-\varepsilon}$

$$\mathcal{R}_t = O(1/t^{\varepsilon}), \tag{6}$$

which is known to be the asymptotically optimal regret for many problems in bandit and online learning in the stochastic setting Auer et al. (2002). Rephrasing this in our terminology, we have that for the recalculation schedule $C = \{1, \ldots, T\}$, the expected cumulative regret for $0 \leq \varepsilon < 1$

$$\mathcal{R}^T = \sum_{t=1}^{T} \mathcal{R}_{t-1} = O\left(\sum_{t=1}^{T} \frac{1}{t^{\varepsilon}}\right) = O(T^{1-\varepsilon}),$$

So even when recalculating at every round, a regret guarantee smaller than $O\left(T^{1-\varepsilon}\right)$ is not possible, and we therefore call this rate **optimal**. This is the quantity we will compare to while attempting to do many fewer recalculations.

To simplify notation further, we will henceforth use $\mathcal{R}$ for expected regret. Therefore, all of our results will be on bounding regret in expectation.

Before proceeding, we note that another interesting setting $\varepsilon = 1$ for $\mathcal{R}_t = O(1/t)$, which is, for example, relevant for problems of estimation of parameters to minimize mean squared error (MSE), where the squared error of the empirical average of t samples versus the true

estimate scales as σ^2/t DeGroot (1986). Here, we would have

$$\mathcal{R}^T = O\left(\sum_{t=1}^{T} \frac{1}{t}\right) = O(\log T).$$

2. Warm up

We begin by analyzing the special case of $\mathcal{R}_t = 1/\sqrt{t}$ The argument in the previous section shows that the optimum regret in this situation is $O(\sqrt{T})$. However, this was the setting in which we recalculated every time we drew a point, which in the online setting isn't always practicable. Thus, inducing a trade-off between the amount of times we recalculate and the regret formed in our calculation.

PROPOSITION 16. *Let D be a probability distribution from which T data points are sampled online and let $\mathcal{R}_t = O(1/\sqrt{t})$. Using one recalculation, one can achieve an expected regret of $O(T^{2/3})$.*

PROOF. We first begin by calculating the optimal expected regret of calculating the algorithm just once. Let c be a constant such that $0 \leq c \leq 1$. We have that the expected regret must be:

$$\sum_{t=1}^{T^c} \mathcal{R}_0 + \sum_{t=T^c+1}^{T} \mathcal{R}_{T^c} = O\left(T^c + T^{1-c/2}\right).$$

We note that this is just a simple optimization of the right hand side, so we equate the exponents and get $c = 2/3$. This recovers the well-known bound of ε-first, (ε-greedy, and epoch-greedy) sampling Langford and Zhang (2007); Sutton and Barto (2018); Tran-Thanh et al. (2010). □

3. Uniform recalculations

One naive idea is to create a sampling schedule that solves our problem is to sample uniformly, so after observing a set ℓ amount of points we sample again and continue in this manner. However, we find this to at best require $O(T^{\varepsilon})$ recalculations to get asymptotically optimum regret.

LEMMA 17. *Let D be a probability distribution from which T data points are sampled online and let $\mathcal{R}_t = O(1/t^\varepsilon)$ for $0 \leq \varepsilon < 1$. If we recalculate uniformly after observing every ℓ points (and therefore recalculate T/ℓ times), then we incur an expected regret of $O\left(\ell + T^{1-\varepsilon}\right)$.*

PROOF. We calculate the expected regret for recalculating after every ℓ points, we carry out the calculation by using the general formula for a uniform schedule.

$$\begin{aligned}\sum_{i=0}^{T/\ell}\sum_{t=1}^{\ell}\mathcal{R}_{\ell i} &= \ell + O\left(\sum_{i=1}^{T/\ell}\sum_{t=1}^{\ell}\frac{1}{(i\ell)^\varepsilon}\right)\\ &= \ell + O\left(\sum_{i=1}^{T/\ell}\frac{\ell^{1-\varepsilon}}{i^\varepsilon}\right)\\ &= O\left(\ell + T^{1-\varepsilon}\right),\end{aligned}$$

which completes the proof. □ □

We note briefly, that this proof holds in the $\varepsilon = 1$ case, we just get the result $\mathcal{R} = O(\ell + \log(T))$.

As the lemma above shows, a constant number of recalculations would imply linearly many recalculations between rounds and therefore linear regret. Therefore, we need a smarter recalculation strategy if we want to recalculate only a constant number of times. This is presented in the following section.

We see in the previous section that uniform recalculation scheduling fails to guarantee small expected regret with a small number of recalculations. So we pivot to non-uniform schedules. Taking, first, a look at what we can do with a constant number of recalculations.

4. Constant number of recalculations

We now consider the case when we are allowed a constant number of recalculations, but they need not be uniformly spaced.

PROPOSITION 18. *Let D be a probability distribution from which T data points are sampled online and let $\mathcal{R}_t = O(1/t^\varepsilon)$ for $0 \leq \varepsilon \leq 1$. Using one recalculation, one can achieve an expected cumulative regret of*

$$\mathcal{R}^T = O\left(T^{1/(1+\varepsilon)}\right).$$

PROOF. Consider recalculating after T^c rounds for $c = \frac{1}{1+\varepsilon}$. We suffer $O\left(T^{1/(1+\varepsilon)}\right)$ cumulative regret on those rounds. For the remaining $T - T^c \leq T$ rounds, we suffer at most

$$\begin{aligned} O\left(T/T^{c\varepsilon}\right) =& O\left(T^{1-c\varepsilon}\right) \\ =& O\left(T^{1-\varepsilon/(1+\varepsilon)}\right) \\ =& O\left(T^{1/(1+\varepsilon)}\right) \end{aligned}$$

regret. □ □

PROPOSITION 19. *Let D be a probability distribution from which T data points are sampled online and let $\mathcal{R}_t = O(1/t^\varepsilon)$ for $0 \leq \varepsilon \leq 1$. Using two recalculations, one can achieve an expected cumulative regret of*

$$\mathcal{R}^T = O\left(T^{1/(1+\varepsilon+\varepsilon^2)}\right).$$

PROOF. Consider recalculating after T^{c_1} and T^{c_2} for $0 \leq c_1 \leq c_2 \leq 1$. Extending the argument in Lemma 18, we incur a total regret of $\mathcal{R}^T \leq T^{c_1} + T^{c_2 - c_1\varepsilon} + T^{1-c_2\varepsilon}$. Equalizing the three terms we get the equations

$$c_1 = c_2 - c_1\varepsilon \text{ and } c_1 = 1 - c_2\varepsilon.$$

Solving the above yields $c_1 = \frac{1}{1+\varepsilon+\varepsilon^2}$ and yields an expected cumulative regret of $O(T^{c_1})$.

□ □

THEOREM 20. *Let D be a probability distribution from which T data points are sampled online and let $\mathcal{R}_t = O(1/t^\varepsilon)$ for $0 \leq \varepsilon < 1$. Using n recalculations, one can achieve an*

expected cumulative regret of

$$\mathcal{R}^T = O\left(nT^{\frac{1-\varepsilon}{1-\varepsilon^{n+1}}}\right).$$

PROOF. Generalizing from the above, we divide into n recalculations and bound $\mathcal{R}^T \leq \sum_{i=1}^{n+1} T^{c_i - c_{i-1}\varepsilon}$ setting $c_0 = 0$ and $c_{n+1} = 1$.

We argue that solving for the system of equations leads to the following recursive formula:

$$c_j = c_{j+1}\frac{\sum_{i=0}^{j-1}\varepsilon^i}{\sum_{i=0}^{j}\varepsilon^i}.$$

We proceed inductively, we first see that $c_1 = c_2 - \varepsilon c_1$, which gets us $c_1 = c_2(1+\varepsilon)^{-1}$, as desired. Moving on, we assume this holds for c_{j-1} and solve for c_j.

$$\begin{aligned}
c_j - \varepsilon c_{j-1} &= c_{j+1} - \varepsilon c_j \\
c_j(1+\varepsilon) &= c_{j+1} + \varepsilon c_{j-1} \\
c_j(1+\varepsilon) &= c_{j+1} + c_j\varepsilon\frac{\sum_{i=0}^{j-2}\varepsilon^i}{\sum_{i=0}^{j-1}\varepsilon^i} \\
c_j\left(1+\varepsilon - \frac{\sum_{i=1}^{j-1}\varepsilon^i}{\sum_{i=0}^{j-1}\varepsilon^i}\right) &= c_{j+1} \\
c_j\left(1+\varepsilon - 1 + \frac{1}{\sum_{i=0}^{j-1}\varepsilon^i}\right) &= c_{j+1} \\
c_j\frac{\sum_{i=0}^{j}\varepsilon^i}{\sum_{i=0}^{j-1}\varepsilon^i} &= c_{j+1} \\
c_j &= c_{j+1}\frac{\sum_{i=0}^{j-1}\varepsilon^i}{\sum_{i=0}^{j}\varepsilon^i}.
\end{aligned}$$

So we have a recursive formula and now we can see that if we terminate after n recalculations we have a telescoping product, and we have that

$$c_1 = \frac{1}{\sum_{i=0}^{n}\varepsilon^i} = \frac{1-\varepsilon}{1-\varepsilon^{n+1}}.$$

Since the $n+1$ phases have equal regret, the total regret is $O(nT_1^c)$, producing the bound of $\mathcal{R}^T = O\left(nT^{\frac{1-\varepsilon}{1-\varepsilon^{n+1}}}\right)$. This bound tends to T^ε for arbitrarily high constant n. □ □

We observe that this proof does hold for $\varepsilon = 1$, we just leave c_1 unsimplified and have that the expected cumulative regret in this case is $O(nT^{1/n})$.

5. Schedules with increasing recalculations in T

After seeing some extreme examples and results on constants, we wish to discover how to improve the amount of times needed to recalculate while still maintaining the optimum expected regret. By employing a "doubling trick" argument, we see that we can easily reduce to $\log(T)$ recalculations. We note that other settings have also been studied that achieve optimal regret using logarithmically many policy "switches," Abbasi-Yadkori et al. (2011); Jaksch et al. (2010).

LEMMA 21. *Let D be a probability distribution from which T points are sampled online and let $\mathcal{R}_t = O(1/t^{\varepsilon})$ for $0 \leq \varepsilon < 1$. There exists a sequence of $\log(T)$ recalculations for which we can achieve optimal expected regret of $O(T^{1-\varepsilon})$.*

PROOF. Let $T = 2^m$, with m sufficiently large, and let $\mathcal{L} = \{2^0, 2^1, \dots 2^{m-1}\}$. If we recalculate the algorithm after seeing each point in $\mathcal{L}$, then we have the following expected cost:

$$\begin{aligned}
\sum_{i=0}^{m}\sum_{t=1}^{2^i}\mathcal{R} &\leq O\left(\sum_{i=0}^{m}\sum_{t=1}^{2^i}\left(\frac{1}{2^{(i-1)\varepsilon}}\right)\right) \\
&= O\left(\sum_{i=0}^{m}\left(2^{i-(i-1)\varepsilon}\right)\right) \\
&= O\left(\sum_{i=0}^{m}\left(2^{i(1-\varepsilon)}\right)\right) \\
&= O\left(2^{(1-\varepsilon)m}\right) \\
&= O\left(T^{(1-\varepsilon)}\right).
\end{aligned}$$

This establishes that we can recalculate $\log(T)$ times and achieve optimal expected regret of $T^{1-\varepsilon}$. □ □

We observe that if the regret is of the form $\mathcal{R}_t = O(1/t)$ (the $\varepsilon = 1$ case), then the regret will be bounded by $O(\log T)$ in the above schedule.

We've established that we can achieve optimal expected regret with only $\log(T)$ recalculations, we go on to establish that we cannot improve this to $\log\log(T)$, by showing that the expected regret when taking $\log\log(T)$ recalculations is $O(\log\log(T)\sqrt{T})$.

LEMMA 22. *Let D be a probability distribution from which T are sampled online and let $\mathcal{R}_t = O(1/t^{\varepsilon})$ for $0 < \varepsilon < 1$, where $c = 1/\varepsilon$. There exists a sequence of $\log\log(T)$ recalculations such that we can achieve an expected regret of $O(\log\log(T)T^{1-\varepsilon})$.*

PROOF. Let $T = c^{c^m}$, with m sufficiently large. Let $\mathcal{L} = \{\ell_0, \ell_1, \dots \ell_m\}$. We recalculate after seeing each point in $\mathcal{L}$.

We wish to find a schedule that will cover all T with $\log\log(T)$ recalculations. We do this with the following schedule: at epoch i, once we have seen a total of $T^{1-c^{-i}}$ points we recalculate. We show that if we cover half of T at epoch $\log\log(T)$, then in the next epoch we will have seen all of T. So we establish this recalculation schedule achieves this. Let y be the amount of times needed to see half of T under the above scheduling scheme, so:

$$\begin{aligned} 1/c &= T^{-c^{-y}} \\ -\log_c(c) &= (-c^{-y})\log_c(T) \\ \frac{1}{\log_c(T)} &= c^{-y} \\ y &= \log_c\log_c(T). \end{aligned}$$

We note that we found the exact point at which we would have half, and that this makes $\log\log(T)$ the minimum amount of rounds needed to cover all of T with this particular schedule. We can be sure of this since there were no assumptions made on the size of T, so we couldn't reduce to $\log\log\log(T)$ rounds. So now, we can sum our per-round cost over

the amount of rounds we must run:

$$\begin{aligned}\sum_{i=0}^{m}\sum_{t=1}^{T^{1-c^{-i}}}\mathcal{R}_{T^{1-c^{-(i-1)}}} &\leq \sum_{i=1}^{m}\sum_{t=1}^{T^{1-c^{-i}}} T^{-\frac{c^{i-1}-1}{c^i}} \\ &= \sum_{i=1}^{m} T^{1-c^{-i}-c^{-1}+c^{-i}} \\ &= \log\log(T)T^{1-c^{-1}} \\ &= \log\log(T)T^{1-\varepsilon}.\end{aligned}$$

Computing the inner sum, we see that at each round, we will have $O(T^{1-\varepsilon})$ expected regret.

□ □

We note this proof does not extend to the $\varepsilon = 1$ case, since the argument hinges on the expected regret being bounded by $O(T^{1-\varepsilon})$.

Now we establish a lower bound, in particular that we cannot achieve an expected regret of $O(T^{1-\varepsilon})$ with $\log\log(T)$ recalculations.

LEMMA 23. *Any schedule that performs $O(\log\log T)$ recalculations using an algorithm that suffers per-round regret of $\mathcal{R} = O(1/t^{\varepsilon})$, $0 < \varepsilon < 1$, must suffer $\omega(T^{1-\varepsilon})$ cumulative regret.*

PROOF. Given $\log\log(T)$ recalculations, we are lower-bounded by a regret of $O(T^{1-\varepsilon})$. Specifically, we show that we cannot achieve a regret of $O(T^{1-\varepsilon})$ given $\log\log(T)$ recalculations. So assume to the contrary, that there exists a schedule that does this. Now, to achieve this clearly the regret incurred at each recalculation has to be $O(T^{1-\varepsilon})$. Based on this we craft an inductive argument on the size of each recalculation round. At round $i-1$, the size, $T^{c_{i-1}}$ must be $O(T^{1-\varepsilon^i})$. As a base case, we see that $T^{c_0} = O(T^{1-\varepsilon})$. We move onto the inductive step: We note that the regret for round i is

$$\begin{aligned}T^{c_i}T^{-\varepsilon c_{i-1}} &= O(T^{1-\varepsilon}) \\ T^{c_i} &= O(T^{1-\varepsilon+\varepsilon c_{i-1}}).\end{aligned}$$

This implies,

$$c_i \leq 1 - \varepsilon + \varepsilon(1 - \varepsilon^{i-2})$$

$$c_i \leq 1 - \varepsilon^{i-1}.$$

This, then results in the total portion of T witnessed as $\sum_{n=1}^{\log\log(T)} T^{1-\varepsilon^n}$. Now, since

$$\sum_{n=1}^{\log\log(T)} T^{1-\varepsilon^n} \leq \log\log(T) T^{1-\varepsilon^{\log\log(T)}} = o(T),$$

we have failed to witness all of T in $\log\log(T)$ rounds and have reached a contradiction. □

□

We note this proof does not extend to the $\varepsilon = 1$ case, since the argument hinges on the expected regret being bounded by $O(T^{1-\varepsilon})$.

CHAPTER 4

Cross-Validation Error Dynamics in Smaller Datasets

1. Introduction

Cross-validation (CV) is a long-standing widely-used (Kohavi, 1995; Stone, 1974) technique to estimate and optimize a model's generalization performance by repeatedly splitting data into training and test sets (see Arlot and Celisse (2010) for a survey). While cross-validation has some known pitfalls (Dietterich, 1998), we herein discuss an underappreciated phenomenon that occurs especially in smaller datasets (containing, say, thousands, not millions of examples): that, across CV splits, training error and test error tend to be negatively correlated—splits where a model fits the training data especially well often yield higher test error, and vice versa. We have also observed a more moderate negative correlation between training and holdout error, as well as almost no correlation between test error and holdout error. These phenomena are of course related to others that have previously been studied, especially in work accounting for the variance in the test and training splits (Nadeau and Bengio, 2003) and on correlations across splits Bates et al. (2024); Bayle et al. (2020).

In this chapter, we introduce a simple model that accounts for these patterns. We show how hypergeometric sampling (sampling from a finite population without replacement) of these types induces the observed anti-correlation between training and test splits, and we show how a model's tendency to overfit to its training data can also account for the negative correlation observed between test and holdout error.

2. An empirical finding

In experimenting with cross-validation on smaller datasets, we repeatedly observed a strong anti-correlation between training and test errors across runs. This phenomenon

seemed surprising at first, though it is easily explainable (as we discuss in the next section). Moreover, we saw a moderate anti-correlation between training and holdout errors and a negligible anti-correlation between test and holdout.

Our experiments first chose a holdout set and then repeatedly split the remaining data into training and test sets, with the holdout set fixed. Figure 3 shows a graphical display of these phenomena on the 'Adult' UCI dataset, run on AdaBoost. The aim of this short paper is to discuss this experimental phenomenon and come up with a convincing model to explain it. We also investigate the ramifications of such an anti-correlation between test and train performance. Running multiple experiments 1 on 11 different datasets testing model selection strategy's effect on mean accuracy. Models were selected based on lowest train error and the lowest test error performance measured against a randomly selected model. We saw a general tendency for models selected based on lowest train error to slightly out-perform both randomly selected models and models selected based on lowest test error.

2.1. Experimental Methods. All experiments were conducted within a single, fully reproducible notebook that implemented data loading, preprocessing, model training, cross-validation, and evaluation. For each dataset, we first sampled a fixed holdout set, which remained unchanged throughout the experiment. The remaining data were repeatedly split into training and test subsets using uniformly random partitions without replacement. Unless otherwise specified, each experiment consisted of 100 independent train–test splits. This design isolates variance arising from cross-validation itself, rather than from fluctuations in the holdout set.

To ensure comparability across models and datasets, we used a standardized preprocessing pipeline based on `scikit-learn`'s `ColumnTransformer`: continuous features were scaled using standardization, and categorical features were one-hot encoded. We evaluated five commonly used classification algorithms:

- AdaBoost with 50 decision-stump base learners,
- Random Forests with 100 trees and maximum depth 10,
- RBF-kernel SVM with $C = 1.0$,

- Logistic Regression with a maximum of 1000 iterations,
- A feedforward neural network (MLP) with hidden layers of sizes (100, 50), early stopping, and maximum 300 training iterations.

For each train–test split, we recorded the training error, test error, and holdout error, producing a triple of error quantities for every model instantiation. These values were used to compute empirical correlations, generate scatterplots, and evaluate model-selection strategies.

To study the effect of the observed train–test anti-correlation on model choice, we repeated the procedure across 11 datasets. For each dataset and algorithm, we compared three selection rules: choosing the model with the lowest training error, choosing the model with the lowest test error, and selecting a model uniformly at random. The holdout accuracy of the selected model was then used as the evaluation metric. Aggregated results across datasets are reported in Figure 1. Our methodological goal here was not hyperparameter optimization but rather to examine the structural behavior of cross-validation under repeated random splits in small data regimes.

2.2. Analysis of Results. Repeated experimentation revealed mild patterns in test–train anti-correlation and best model performance that we examine here. In Figure 2 we see that outside of cross-validation trained with a neural network, there is a consistent pattern of anti-correlation between test and train error rates. This deviation in anti-correlation cannot be explained solely by differences in training error, i.e. anti-correlation wasn't lost due to no variability in training errors, as several of the other models also consistently achieve near-zero training error. Instead, we hypothesize that anti-correlation relies on a form of bias that links the difficulty of the training fold to the effective complexity of the learned classifier. For models such as AdaBoost or SVMs, a "hard" fold forces the learner toward a simpler or more biased decision boundary, which tends to generalize better; conversely, an "easy" fold allows the model to fit more idiosyncratic structure, often harming test performance. This coupling creates the systematic inversion of training and testing errors.

Neural networks, by contrast, do not exhibit such behavior in the small-data regime. One plausible hypothesis is that when faced with harder folds, they do not systematically revert to simpler effective hypotheses, but instead rely on high-dimensional representations whose effective complexity does not contract in a predictable way. As a result, variations in training-set difficulty do not induce correspondingly structured changes in generalization performance, and the train–test anti-correlation mechanism breaks down. This interpretation is consistent wit a broader body of work documenting counterintuitive and stability-related phenomena in neural networks.Belkin et al. (2019)

We now address where this observed anti-correlation points to in terms of optimal model selection. Unfortunately, we failed to consistently find a method of model selection that yielded a statistically significant hold-out performance. However, we note that there is an observed mild improvement on hold-out performance in models selected based on yielding lowest train error. While often not statistically significantly different at $n = 75$ trials of cross-validation, under a chi-square test of significance lowest train error models wins at holdout performance most often among tested models at a statistically significant $\alpha = .05$ level. Thus, when there is no obvious model choice, our recommendation is to select a model based on lowest train error performance.

FIGURE 1. Results from 100-fold AdaBoost Cross-Validation ran on Census data, 'Adult', from UCI's Machine Learning Repository (Kohavi and Becker, 1996). We used decision stumps with 50 base learners in our AdaBoost model. We then performed 100-fold cross validation. There are slightly over 48 thousand instances and 15 features in this data set. The data was partitioned into train, test, and hold-out as follows: initially 10% was reserved as the hold-out set, then 67.5% of the data was devoted to the training set and the remaining 22.5% to the test set.

FIGURE 2. Visualization of the mean and standard deviation of correlation between test and train error rates by dataset and algorithm. This is the average of 75 separate runs of cross-validation run on different random seeds, in attempt to capture the true pattern of correlation. We see aside from CV with Neural Network, that we consistently see our acclaimed anti-correlation.

Table 1: Mean accuracy performance ± standard deviation across 75 runs for each dataset, algorithm, and selection strategy. Datasets all pulled from UCI's Machine Learning Repository. Bolded entries indicate the best performing strategy, asterisks indicate a statistically significant result based on an $\alpha = .05$ confidence level.

Dataset	Algorithm	Lowest Train Error	Lowest Test Error	Random Baseline
	AdaBoost	**0.580 ± 0.008***	0.578 ± 0.008	0.577 ± 0.008

Dataset	Algorithm	Lowest Train Error	Lowest Test Error	Random Baseline
Adult Income	Logistic Regression	0.575 ± 0.009	0.576 ± 0.009	**0.576 ± 0.008**
	Neural Network	0.553 ± 0.011	**0.566 ± 0.009***	0.565 ± 0.011
	Random Forest	0.581 ± 0.007	0.580 ± 0.007	**0.581 ± 0.008**
	SVM	0.572 ± 0.009	**0.573 ± 0.007**	0.572 ± 0.008
Bank Marketing	AdaBoost	**0.895 ± 0.005**	0.894 ± 0.005	0.894 ± 0.006
	Logistic Regression	**0.898 ± 0.006**	0.897 ± 0.005	0.897 ± 0.005
	Neural Network	0.897 ± 0.007	**0.900 ± 0.006**	0.898 ± 0.007
	Random Forest	**0.897 ± 0.004**	0.896 ± 0.004	0.896 ± 0.004
	SVM	**0.898 ± 0.006**	0.897 ± 0.005	0.897 ± 0.005
Heart Disease	AdaBoost	0.803 ± 0.059	**0.810 ± 0.055**	0.807 ± 0.060
	Logistic Regression	0.821 ± 0.057	**0.823 ± 0.055**	0.820 ± 0.051
	Neural Network	**0.816 ± 0.047**	0.808 ± 0.057	0.802 ± 0.055
	Random Forest	0.810 ± 0.057	0.807 ± 0.057	**0.815 ± 0.058**
	SVM	0.816 ± 0.056	0.808 ± 0.055	**0.821 ± 0.057**
Ionosphere	AdaBoost	0.883 ± 0.039	0.890 ± 0.038	**0.892 ± 0.041**
	Logistic Regression	0.878 ± 0.041	**0.887 ± 0.045**	0.881 ± 0.043
	Neural Network	**0.894 ± 0.048***	0.886 ± 0.049	0.860 ± 0.051
	Random Forest	0.934 ± 0.033	0.936 ± 0.031	**0.938 ± 0.034**
	SVM	0.938 ± 0.035	0.938 ± 0.033	**0.939 ± 0.033**
Landsat	AdaBoost	**0.933 ± 0.012**	0.932 ± 0.011	0.931 ± 0.011
	Logistic Regression	**0.930 ± 0.011**	**0.930 ± 0.011**	0.929 ± 0.010
	Neural Network	**0.950 ± 0.010***	0.946 ± 0.010	0.945 ± 0.012
	Random Forest	**0.954 ± 0.009**	**0.954 ± 0.009**	**0.954 ± 0.009**
	SVM	**0.952 ± 0.009**	0.951 ± 0.010	0.951 ± 0.010
Letter Recognition	AdaBoost	**0.851 ± 0.008***	0.847 ± 0.009	0.844 ± 0.008
	Logistic Regression	**0.808 ± 0.007***	0.805 ± 0.007	0.805 ± 0.007
	Neural Network	**0.948 ± 0.009***	0.945 ± 0.009	0.940 ± 0.010
	Random Forest	**0.919 ± 0.008***	0.916 ± 0.009	0.914 ± 0.010

Dataset	Algorithm	Lowest Train Error	Lowest Test Error	Random Baseline
	SVM	**0.922 ± 0.009**	0.919 ± 0.009	0.919 ± 0.009
Online Shoppers	AdaBoost	**0.896 ± 0.007**	0.895 ± 0.007	0.894 ± 0.007
	Logistic Regression	0.887 ± 0.007	0.886 ± 0.006	**0.887 ± 0.006**
	Neural Network	0.898 ± 0.007	**0.900 ± 0.007**	0.899 ± 0.007
	Random Forest	**0.905 ± 0.007**	**0.905 ± 0.007**	**0.905 ± 0.006**
	SVM	**0.894 ± 0.006**	**0.894 ± 0.006**	**0.894 ± 0.006**
Spambase	AdaBoost	0.900 ± 0.041	**0.908 ± 0.036**	0.898 ± 0.044
	Logistic Regression	**0.926 ± 0.010**	0.924 ± 0.010	0.924 ± 0.010
	Neural Network	**0.941 ± 0.008**	0.939 ± 0.009	0.938 ± 0.008
	Random Forest	0.940 ± 0.008	0.939 ± 0.009	**0.940 ± 0.008**
	SVM	**0.930 ± 0.009**	**0.930 ± 0.010**	**0.930 ± 0.009**
Statlog Shuttle	AdaBoost	**0.997 ± 0.002**	**0.997 ± 0.002**	**0.997 ± 0.002**
	Logistic Regression	**0.962 ± 0.007**	0.961 ± 0.007	0.961 ± 0.007
	Neural Network	**0.997 ± 0.002***	0.996 ± 0.002	0.996 ± 0.003
	Random Forest	**0.998 ± 0.001**	**0.998 ± 0.001**	**0.998 ± 0.001**
	SVM	**0.996 ± 0.002**	**0.996 ± 0.002**	**0.996 ± 0.002**
Wine Quality	AdaBoost	**0.818 ± 0.008**	**0.818 ± 0.008**	0.816 ± 0.008
	Logistic Regression	**0.818 ± 0.008**	**0.818 ± 0.008**	**0.818 ± 0.008**
	Neural Network	**0.839 ± 0.010***	0.835 ± 0.010	0.831 ± 0.010
	Random Forest	**0.866 ± 0.008**	0.864 ± 0.008	0.864 ± 0.008
	SVM	**0.834 ± 0.007**	0.832 ± 0.008	0.833 ± 0.007
Yeast	AdaBoost	**0.772 ± 0.025**	0.765 ± 0.028	0.762 ± 0.027
	Logistic Regression	**0.761 ± 0.022**	0.756 ± 0.021	0.755 ± 0.020
	Neural Network	**0.772 ± 0.027**	0.766 ± 0.023	0.766 ± 0.025
	Random Forest	**0.777 ± 0.024**	0.776 ± 0.025	**0.777 ± 0.025**
	SVM	**0.771 ± 0.025**	**0.771 ± 0.023**	**0.771 ± 0.023**

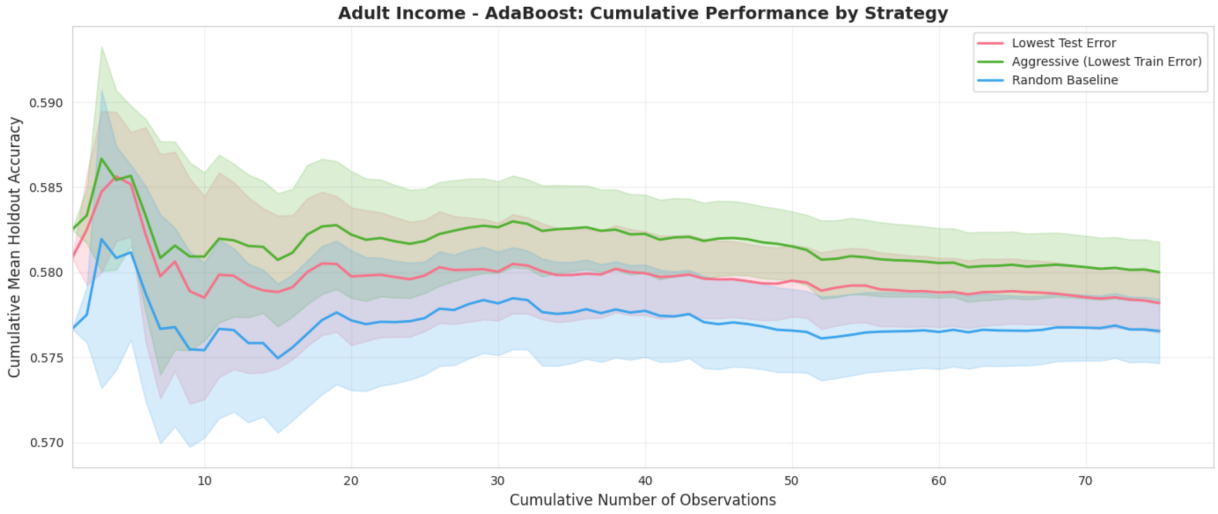

FIGURE 3. Results from 75 runs of AdaBoost 100-fold Cross-Validation ran on Census data, 'Adult', from UCI's Machine Learning Repository (Kohavi and Becker, 1996). We used the same setup as above, and ran on different random seeds. This is exemplifying the process of how we attained the final numbers in Table 1.

3. A theoretical explanation

This section presents a minimal probabilistic model consistent with our experimental findings: (i) training and CV-test errors are strongly negatively correlated across splits, (ii) training and holdout errors exhibit a nontrivial (typically moderate) correlation, and (iii) CV-test and holdout errors are only weakly correlated. The key point is that these three phenomena arise from *distinct* sources of variability that affect the observed errors in different ways.

We model three conceptually separate contributors to across-split variability:

(1) **Fold composition (difficulty) variability** induced by sampling without replacement from a finite population, which affects *train* and *test* in opposite directions and is absent from the fixed holdout set.

(2) **Overfitting variability**, summarized by a scalar δ, which lowers training error but increases out-of-sample (test and holdout) error.

(3) **Overall generalization-quality variability**, summarized by a scalar γ, which captures split-to-split fluctuations in how well the learned hypothesis generalizes across *all* evaluation sets.

This separation allows training–test error to be strongly anti-correlated while training–holdout and test–holdout correlations remain weaker.

3.1. Setup and latent "difficulty" field. Let N be the total dataset size. We first draw a fixed holdout set of size H, leaving $M = N - H$ points for repeated random train–test splitting. On each replicate, we draw a training set of size n uniformly at random without replacement from these M points, and the remaining $m = M - n$ points form the CV test fold.

To model heterogeneity in example difficulty, assign each of the M CV points a fixed (but unobserved) difficulty score u_i, $i = 1, \ldots, M$, with finite-population mean zero:

$$\frac{1}{M}\sum_{i=1}^{M} u_i \;=\; 0.$$

Let the finite-population variance be

$$s_u^2 \;=\; \frac{1}{M-1}\sum_{i=1}^{M} u_i^2.$$

For a given split, define the average difficulty in the training and test folds as

$$\bar{u}_{\text{train}} \;=\; \frac{1}{n}\sum_{i\in\text{Train}} u_i, \qquad \bar{u}_{\text{test}} \;=\; \frac{1}{m}\sum_{i\in\text{Test}} u_i.$$

Because Train and Test are complementary subsets of the same centered finite population,

$$n\,\bar{u}_{\text{train}} \;+\; m\,\bar{u}_{\text{test}} \;=\; 0 \qquad\Longrightarrow\qquad \bar{u}_{\text{test}} \;=\; -\frac{n}{m}\,\bar{u}_{\text{train}}. \tag{7}$$

Thus, fold-composition effects are perfectly negatively aligned between training and test splits.

Standard sampling-without-replacement identities yield

$$\mathrm{Var}(\bar{u}_{\mathrm{train}}) = \Big(1 - \frac{n}{M}\Big)\frac{s_u^2}{n} = \frac{m}{M} \cdot \frac{s_u^2}{n}, \tag{8}$$

$$\mathrm{Var}(\bar{u}_{\mathrm{test}}) = \Big(1 - \frac{m}{M}\Big)\frac{s_u^2}{m} = \frac{n}{M} \cdot \frac{s_u^2}{m}, \tag{9}$$

$$\mathrm{Cov}(\bar{u}_{\mathrm{train}}, \bar{u}_{\mathrm{test}}) = -\frac{s_u^2}{M}. \tag{10}$$

3.2. Model-level variability. We introduce two split-level latent variables.

Overfitting variability. Let δ capture split-to-split variation in effective model complexity or overfitting. Larger δ lowers training error but increases out-of-sample error. Assume

$$\mathbb{E}[\delta] = 0, \qquad \mathrm{Var}(\delta) = \sigma_\delta^2.$$

Generalization-quality variability. Let γ capture variation in the overall quality of the learned hypothesis across splits, reflecting factors such as how informative or representative the training fold is. Larger γ corresponds to uniformly better generalization. Assume

$$\mathbb{E}[\gamma] = 0, \qquad \mathrm{Var}(\gamma) = \sigma_\gamma^2.$$

For simplicity, we treat δ, γ, and the fold-composition averages $(\bar{u}_{\mathrm{train}}, \bar{u}_{\mathrm{test}})$ as mutually uncorrelated. The assumption of independence is made for analytical clarity; allowing moderate dependence does not change the qualitative conclusions.

3.3. Observed errors. We model the observed errors as additive combinations of baseline error, fold difficulty, model-level effects, and evaluation noise:

$$\hat{p}_{\mathrm{train}} = \mu \;+\; a\,\bar{u}_{\mathrm{train}} \;-\; \alpha\,\delta \;-\; \gamma \;+\; \varepsilon_{\mathrm{train}}, \tag{11}$$

$$\hat{p}_{\mathrm{test}} = \mu \;+\; a\,\bar{u}_{\mathrm{test}} \;+\; \beta\,\delta \;-\; \gamma \;+\; \varepsilon_{\mathrm{test}}, \tag{12}$$

$$\hat{p}_{\mathrm{hold}} = \mu_{\mathrm{hold}} \;+\; \beta\,\delta \;-\; \gamma \;+\; \varepsilon_{\mathrm{holdout}}. \tag{13}$$

Here $a > 0$ scales the effect of fold difficulty; $\alpha, \beta > 0$ control the strength of overfitting effects (often $\alpha > \beta$, reflecting stronger sensitivity of training error to overfitting); and

$\varepsilon_{\text{train}}, \varepsilon_{\text{test}}, \varepsilon_{holdout}$ are mean-zero noise terms, mutually uncorrelated and uncorrelated with all latent variables.

3.4. Predicted covariances and correlations. Using (11)–(13) and (10), we obtain:

Train vs. Test.

$$\text{Cov}(\hat{p}_{\text{train}}, \hat{p}_{\text{test}}) = -\,a^2 \frac{s_u^2}{M} \;-\; \alpha\beta\,\sigma_\delta^2 \;+\; \sigma_\gamma^2. \tag{14}$$

In small datasets, the finite-population term dominates, yielding a strong negative correlation despite the shared γ component.

Train vs. Holdout.

$$\text{Cov}(\hat{p}_{\text{train}}, \hat{p}_{hold}) = -\,\alpha\beta\,\sigma_\delta^2 \;+\; \sigma_\gamma^2. \tag{15}$$

Thus, training and holdout errors can exhibit a moderate correlation whose sign and magnitude depend on the balance between overfitting variability and overall generalization quality.

Test vs. Holdout.

$$\text{Cov}(\hat{p}_{\text{test}}, \hat{p}_{hold}) = \beta^2\,\sigma_\delta^2 \;+\; \sigma_\gamma^2 \;>\; 0. \tag{16}$$

The correlation between test and holdout errors will usually remain weak because $\hat{p}_{\text{test}}$ includes an additional variance term from fold composition:

$$\text{Var}(\hat{p}_{\text{test}}) = a^2\,\text{Var}(\bar{u}_{\text{test}}) + \beta^2\sigma_\delta^2 + \sigma_\gamma^2 + \text{Var}(\varepsilon_{\text{test}}),$$

which is absent from $\hat{p}_{hold}$. This extra variability dilutes the test–holdout correlation even when the covariance is positive.

3.5. Interpretation and "small data" behavior. The model isolates three distinct mechanisms: exact complementarity of finite-population sampling (which drives strong train–test anti-correlation), overfitting variability (which weakly couples training to out-of-sample performance), and generalization-quality variability (which induces shared movement across all error estimates). In small datasets, fold-composition effects dominate test-set variance,

naturally explaining why test error is strongly anti-correlated with training error yet only weakly informative about holdout performance. As dataset sizes increase and learning algorithms become more stable, these effects diminish and the correlations collapse toward zero.

4. Conclusions

We documented an underappreciated empirical pattern in repeated cross-validation on smaller datasets: training and test errors across splits tend to be strongly negatively correlated, training and holdout errors exhibit a nontrivial (often moderate) correlation, and test and holdout errors are only weakly correlated. Beyond being a curiosity, this behavior matters because it complicates the interpretation of cross-validation outcomes and can subtly affect model selection when one repeatedly trains models on different random splits.

To explain these observations, we introduced a minimal generative model in Section 3 that separates three sources of across-split variability: (i) finite-population sampling effects from drawing complementary train/test folds without replacement, captured by the latent fold-difficulty field u_i; (ii) split-to-split variation in effective overfitting, summarized by δ, which reduces training error but worsens out-of-sample error; and (iii) split-to-split variation in overall generalization quality, summarized by γ, which shifts errors on train, test, and holdout in a shared direction. In this model, the strong train–test anti-correlation is primarily a consequence of complementarity in finite-population sampling, while the weaker relationships involving holdout error arise from the shared model-level terms δ and γ, with test–holdout correlation additionally diluted by test-fold composition variance.

The model is intentionally coarse and leaves several natural refinements open. Most notably, it does not explicitly model how particular "hard" examples migrate between folds and influence the learned hypothesis, nor does it attempt to predict when particular learning algorithms (e.g., neural networks in our experiments) will fail to exhibit the train–test anti-correlation pattern. A more detailed treatment might allow δ and γ to depend on fold difficulty, incorporate heterogeneity in example-level noise, or directly model algorithmic

stability. Finally, it would be interesting to extend these ideas beyond classification to regression and other loss functions, where analogous finite-sample correlation effects may arise but interact differently with estimation error.

APPENDIX

Cross-Validation Experiment Code

This appendix contains the full Python source code used to run the cross-validation experiments described in Chapter 4. The code loads UCI benchmark datasets, partitions each into holdout, test, and training sets (15%/25%/60%), and records training and test errors across cross-validation folds to study the anti-correlation phenomenon.

The original experiments were run in Google Colab; the `ucimlrepo` package can be installed via `pip install ucimlrepo`. After loading and processing datasets from ICML's repo, noting conversion to binary for some multiclass datasets, we then used the stated function to run out CV experiements. We also note that the following function relies on popular modules like `numpy`, `sci-kit learn` and `pandas`, and these dependencies would have to be loaded, but this is omitted from the code for brevity.

```
def get_algorithm_instance(algo_name, seed):
    """Get a fresh instance of the specified algorithm with given seed"""
    base_learner = DecisionTreeClassifier(max_depth=1)

    algorithms = {
        'AdaBoost': AdaBoostClassifier(
            estimator=base_learner,
            n_estimators=50,
            learning_rate=1.0,
            random_state=seed
        ),
        'Random Forest': RandomForestClassifier(
            n_estimators=100,
            max_depth=10,
            random_state=seed
        ),
        'SVM': SVC(
            kernel='rbf',
            C=1.0,
            random_state=seed
        ),
        'Logistic Regression': LogisticRegression(
            max_iter=1000,
            random_state=seed
        ),
        'Neural Network': MLPClassifier(
            hidden_layer_sizes=(100, 50),
            max_iter=300,
            random_state=seed,
            early_stopping=True,
            validation_fraction=0.1
        )
    }

```

```
    return algorithms[algo_name]



def run_single_experiment(loop_id, X, y, n_runs=100, algorithm_names=None):
    """Run a single experiment loop for all algorithms - INDIVIDUAL MODELS ONLY"""
    start_time = time.time()

    if algorithm_names is None:
        algorithm_names = ['AdaBoost', 'Random Forest', 'SVM',
                           'Logistic Regression', 'Neural Network']

    # Initialize holdout data (consistent across all runs for this loop)
    X_train_val, X_holdout, y_train_val, y_holdout = train_test_split(
        X, y, test_size=0.15, random_state=42 + loop_id, stratify=y
    )

    # Setup preprocessor
    numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
    categorical_features = X.select_dtypes(include=['object']).columns

    numeric_transformer = StandardScaler()
    categorical_transformer = OneHotEncoder(handle_unknown='ignore')

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numeric_features),
            ('cat', categorical_transformer, categorical_features)
        ])

    all_loop_results = []
    all_model_metrics = []

    # Process each algorithm
```

```
    for algo_name in algorithm_names:

        models = []
        train_errors = []
        test_errors = []
        holdout_errors = []

        for seed in range(n_runs):
            if seed % 20 == 0:

            # Stratified split
            X_train, X_test, y_train, y_test = train_test_split(
                X_train_val, y_train_val,
                test_size=0.25,
                random_state=seed,
                stratify=y_train_val
            )

            # Create pipeline with algorithm
            current_algo = get_algorithm_instance(algo_name, seed)
            pipeline = Pipeline([
                ('preprocessor', preprocessor),
                ('classifier', current_algo)
            ])

            # Train
            pipeline.fit(X_train, y_train)

            # Calculate errors
            train_error = 1 - accuracy_score(y_train, pipeline.predict(X_train))
            test_error = 1 - accuracy_score(y_test, pipeline.predict(X_test))
            holdout_error = 1 - accuracy_score(y_holdout, pipeline.predict(X_holdout))

            models.append(pipeline)
```

```
            train_errors.append(train_error)
            test_errors.append(test_error)
            holdout_errors.append(holdout_error)

        # Convert to arrays
        train_errors = np.array(train_errors)
        test_errors = np.array(test_errors)
        holdout_errors = np.array(holdout_errors)

        # Store individual model metrics
        for i in range(n_runs):
            all_model_metrics.append({
                'experiment_id': EXPERIMENT_ID,
                'loop_id': loop_id + 1,
                'algorithm': algo_name,
                'model_id': i,
                'train_error': train_errors[i],
                'test_error': test_errors[i],
                'holdout_error': holdout_errors[i]
            })

        pure_test_idx = np.argmin(test_errors)
        aggressive_idx = np.argmin(train_errors)
        np.random.seed(loop_id)
        random_idx = np.random.choice(n_runs)

        selection_strategies = {
                "Lowest Test Error": pure_test_idx,
                "Aggressive (Lowest Train Error)": aggressive_idx,
                "Random Baseline": random_idx
            }

        for strategy_name, idx in selection_strategies.items():
                best_model = models[idx]
```

```
                                                                                
            y_pred = best_model.predict(X_holdout)
            accuracy = accuracy_score(y_holdout, y_pred)
                                                                                
            all_loop_results.append({
                'experiment_id': EXPERIMENT_ID,
                'loop_id': loop_id + 1,
                'algorithm': algo_name,
                'strategy': strategy_name,
                'holdout_accuracy': accuracy,
                'train_error': train_errors[idx],
                'test_error': test_errors[idx],
                'model_id': idx,
            })
                                                                                
    return all_loop_results, pd.DataFrame(all_model_metrics)
```

Bibliography

Hasan Abasi. Error-tolerant non-adaptive learning of a hidden hypergraph. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Hasan Abasi and Bshouty Nader. On learning graphs with edge-detecting queries. In *Algorithmic Learning Theory*, pages 3–30. PMLR, 2019.

Hasan Abasi, Nader H Bshouty, and Hanna Mazzawi. On exact learning monotone DNF from membership queries. In *Algorithmic Learning Theory: 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings 25*, pages 111–124. Springer, 2014.

Hasan Abasi, Nader H Bshouty, and Hanna Mazzawi. Non-adaptive learning of a hidden hypergraph. *Theoretical Computer Science*, 716:15–27, 2018.

Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 2312–2320, 2011.

Matthew Aldridge, Oliver Johnson, Jonathan Scarlett, et al. Group testing: an information theory perspective. *Foundations and Trends® in Communications and Information Theory*, 15(3-4):196–392, 2019.

Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics*, 18(4):697–712, 2005.

Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing*, 33(2):487–501, 2004.

Dana Angluin. Queries and concept learning. *Machine learning*, 2:319–342, 1988.

Dana Angluin and Jiang Chen. Learning a hidden graph using $O(\log n)$ queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008.

Dana Angluin, Jiang Chen, and Manfred Warmuth. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7(10), 2006.

Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002.

Bethany Austhof and Lev Reyzin. A model for optimizing recalculation schedules to minimize regret. In *Proceedings of the 18th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, pages 1–9, 2024.

Bethany Austhof, Lev Reyzin, and Erasmo Tani. Non-adaptive learning of random hypergraphs with queries. In *Proceedings of the 58th IEEE International Symposium on*

Information Theory (ISIT), 2025.

Eric Balkanski, Oussama Hanguir, and Shatian Wang. Learning low degree hypergraphs. In *Conference on Learning Theory*, pages 419–420. PMLR, 2022.

Stephen Bates, Trevor Hastie, and Robert Tibshirani. Cross-validation: What does it estimate and how well does it do it? *Journal of the American Statistical Association*, 119 (546):1434–1445, 2024.

Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. Networks beyond pairwise interactions: Structure and dynamics. *Physics reports*, 874:1–92, 2020.

Pierre Bayle, Alexandre Bayle, Lucas Janson, and Lester Mackey. Cross-validation confidence intervals for test error. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

Robi Bhattacharjee and Michal Moshkovitz. No-substitution k-means clustering with adversarial order. In Vitaly Feldman, Katrina Ligett, and Sivan Sabato, editors, *Algorithmic Learning Theory, 16-19 March 2021, Virtual Conference, Worldwide*, volume 132 of *Proceedings of Machine Learning Research*, pages 345–366. PMLR, 2021.

Robi Bhattacharjee, Jacob Imola, Michal Moshkovitz, and Sanjoy Dasgupta. Online k-means clustering on arbitrary data streams. In Shipra Agrawal and Francesco Orabona, editors, *International Conference on Algorithmic Learning Theory, February 20-23, 2023, Singapore*, volume 201 of *Proceedings of Machine Learning Research*, pages 204–236. PMLR, 2023.

Sheng Cai, Mohammad Jahangoshahi, Mayank Bakshi, and Sidharth Jaggi. Efficient algorithms for noisy group testing. *IEEE Transactions on Information Theory*, 63(4):2113–2136, 2017.

Mahdi Cheraghchi and João Ribeiro. Simple codes and sparse recovery with fast decoding. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 156–160. IEEE, 2019.

Jacob Chodoriwsky and Lucia Moura. An adaptive algorithm for group testing for complexes. *Theoretical Computer Science*, 592:1–8, 2015.

Morris H. DeGroot. *Probability and Statistics*. 1986.

Ofer Dekel, Jian Ding, Tomer Koren, and Yuval Peres. Bandits with switching costs: $T^{2/3}$ regret. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 459–467. ACM, 2014.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

Robert Dorfman. The detection of defective members of large populations. *The Annals of mathematical statistics*, 14(4):436–440, 1943.

Ding-Zhu Du and Frank Kwang-ming Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 1999.

Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1): 1–57, 2008.

Hong Gao, Frank K Hwang, My T Thai, Weili Wu, and Taieb Znati. Construction of d (h)-disjunct matrix for group testing in hypergraphs. *Journal of Combinatorial Optimization*, 12(3):297–301, 2006.

Vladimir Grebinski and Gregory Kucherov. Reconstructing a Hamiltonian cycle by querying the graph: Application to dna physical mapping. *Discrete Applied Mathematics*, 88(1-3): 147–165, 1998.

Tom Hess, Michal Moshkovitz, and Sivan Sabato. A constant approximation algorithm for sequential random-order no-substitution k-median clustering. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 3298–3308, 2021.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600, 2010.

Ian A. Kash, Lev Reyzin, and Zishun Yu. Slowly changing adversarial bandit algorithms are provably efficient for discounted mdps. *CoRR*, abs/2205.09056, 2022.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1145, 1995.

Ron Kohavi and Barry Becker. Adult data set, 1996. URL `https://archive.ics.uci.edu/ml/datasets/adult`.

John Langford and Tong Zhang. The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in neural information processing systems*, 20(1):96–1, 2007.

Zihan Li, Matthias Fresacher, and Jonathan Scarlett. Learning Erdos-Renyi random graphs via edge detecting queries. *Advances in Neural Information Processing Systems*, 32, 2019.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988. doi: 10.1023/A:1022869011914. URL `https://link.springer.com/article/10.1023/A:1022869011914`.

Quintino Francesco Lotito, Federico Musciotto, Alberto Montresor, and Federico Battiston. Higher-order motif analysis in hypergraphs. *Communications Physics*, 5(1):79, 2022.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents (extended abstract). In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5573–5577. ijcai.org, 2018.

Anthony J Macula, Vyacheslav V Rykov, and Sergey Yekhanin. Trivial two-stage group testing for complexes using almost disjunct matrices. *Discrete Applied Mathematics*, 137 (1):97–107, 2004.

Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431. IEEE Computer Society, 2001.

Michal Moshkovitz. Unexpected effects of online no-substitution k-means clustering. In Vitaly Feldman, Katrina Ligett, and Sivan Sabato, editors, *Algorithmic Learning Theory, 16-19 March 2021, Virtual Conference, Worldwide*, volume 132 of *Proceedings of Machine Learning Research*, pages 892–930. PMLR, 2021.

Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003.

Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Algorithmic Learning Theory: 18th International Conference, ALT 2007, Sendai, Japan, October 1-4, 2007. Proceedings 18*, pages 285–297. Springer, 2007.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B*, 36(2):111–147, 1974.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

David C Torney. Sets pooling designs. *Annals of Combinatorics*, 3(1):95–101, 1999.

Long Tran-Thanh, Archie Chapman, Enrique Munoz De Cote, Alex Rogers, and Nicholas R Jennings. Epsilon–first policies for budget–limited multi-armed bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1211–1216, 2010.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971. doi: 10.1137/1116025.

Vita

EDUCATION

University of Illinois Chicago, Chicago, Illinois
Ph.D. in Mathematics, May 2026, GPA: 3.8/4.0

Western Michigan University, Kalamazoo, Michigan
B.A. in Mathematics, April 2019, GPA: 4.0/4.0

TEACHING

University of Illinois Chicago: 2020 – 2026

- Pre-Calculus, Applied Calculus, Differential Equations, Intro to Statistics and Intro to Python.
- Youth Scholar Program Lead Instructor – Summer 2022

WORK EXPERIENCE

Data Science Intern

- Lakril Technologies, Chicago, Illinois: June 2024 – August 2024
- CCC Intelligent Solutions, Chicago, Illinois: June 2021 – August 2021

PRE-PRINT

- Austhof, B., Reyzin, L. *Cross-Validation Error Dynamics in Smaller Datasets.* MOSS@ICML, 2025.

PUBLICATIONS

- Bethany Austhof, Lev Reyzin, Erasmo Tani. *Non-Adaptive Learning of Random Hypergraphs with Queries.* In the Proceedings of the 58th IEEE International Symposium on Information Theory (ISIT), 2025.
- Bethany Austhof, Lev Reyzin. *A Model for Optimizing Recalculation Schedules to Minimize Regret.* In the Proceedings of the 18th International Symposium on Artificial Intelligence and Mathematics (ISAIM), 2024, pp. 1–9.
- Bethany Austhof, Patrick Bennett, Nick Christo. *The set of ratios of derangements to permutations in digraphs is dense in* $[0, 1/2]$. Electronic Journal of Combinatorics (2021), Vol. 29(1), p. 1.8.
- Bethany Austhof, Sean English. *Nearly-Regular Hypergraphs and Saturation of Berge Stars.* Electronic Journal of Combinatorics (2019), Vol. 26(4), p. 4.49.