Accelerated Experimental Design for Pairwise Comparisons

Yuan Guo[1] Jennifer Dy[1] Deniz Erdogmus[1] Jayashree Kalpathy-Cramer[2]
Susan Ostmo[3] J. Peter Campbell[3] Michael F. Chiang[3] Stratis Ioannidis[1]

Abstract

Pairwise comparison labels are more informative and less variable than class labels, but generating them poses a challenge: their number grows quadratically in the dataset size. We study a natural experimental design objective, namely, D-optimality, that can be used to identify which K pairwise comparisons to generate. This objective is known to perform well in practice, and is submodular, making the selection approximable via the greedy algorithm. A naïve greedy implementation has $O(N^2d^2K)$ complexity, where N is the dataset size, d is the feature space dimension, and K is the number of generated comparisons. We show that, by exploiting the inherent geometry of the dataset–namely, that it consists of pairwise comparisons–the greedy algorithm's complexity can be reduced to $O(N^2(K+d)+N(dK+d^2)+d^2K)$. We apply the same acceleration also to the so-called lazy greedy algorithm. When combined, the above improvements lead to an execution time of less than 1 hour for a dataset with 10^8 comparisons; the naïve greedy algorithm on the same dataset would require more than 10 days to terminate.

1 Introduction

In many supervised learning applications, including medicine and recommender systems, class labels are solicited from (and generated by) human labelers. Datasets constructed thusly are often noisy, to counter this, several recent works [1–5] propose augmenting datasets via *comparisons*. For example, a medical expert can classify patients as, e.g. diseased or normal, but can also order pairs of patients w.r.t. disease severity. Similarly, beyond generating class labels in recommender systems (e.g., stars), labelers can also declare their relative preference between any two items.

Incorporating comparison labels to the training process has two advantages. First, comparisons indeed reveal additional information compared to traditional class labels: this is because they capture both inter *and* intra-class relationships; the latter are not revealed via class labels alone. Second, comparisons are often less noisy than (absolute) class labels. Indeed, human labelers disagreeing when generating class judgments often exhibit reduced variability when asked to compare pairs of items instead. This has been extensively documented in a broad array of domains, including medicine [1, 6], movie recommendations [3, 7], travel recommendations [8], music recommendations [9], and web page recommendations [10], to name a few.

Nevertheless, soliciting comparison labels poses a significant challenge, as the number of potential comparisons is quadratic in the dataset size. It therefore makes sense to solve the following experimental design (i.e., batch active learning) problem: *given a budget K, and a set of existing class labels, identify the K comparison labels the expert should generate that will better augment the existing dataset.* There are several natural ways through which this experimental design problem can be formalized. In this paper, we focus on an objective motivated by D-Optimal design [11, 12]. This objective leads to selections that perform very well in practice against competing methods [5]. Most importantly, it is also *submodular*; as such, a set of comparisons attaining a constant approximation guarantee can be constructed in polynomial time via the so-called *greedy algorithm* [13].

Applying the greedy algorithm naïvely in this experimental design setting leaves a lot to be desired. Given that the set of comparisons is quadratic, a naïve implementation of the algorithm leads to a complexity of $O(N^2d^2K)$, where N is the size of the dataset, d is the dimension of the feature space, and K is the size of the selected set of comparisons. The quadratic nature of this computation makes the use of the algorithm prohibitive for all but the smallest datasets, especially when the samples are high-dimensional. On the other hand, the fact that the same N objects participate in these $O(N^2)$ pairs suggests an underlying structure that can potentially be exploited to improve time performance.

To that end, we make the following contributions:

- We formally study the problem of accelerating the greedy algorithm for learning pairwise comparisons.

[1]{yuanee20, erdogmus, jdy, ioannidis}@ece.neu.edu. ECE Department, Northeastern University, Boston, MA, USA. [2]kalpathy@nmr.mgh.harvard.edu. Department of Radiology, Massachusetts General Hospital, Charlestown, MA, USA. [3]{ostmo, campbelp, chiangm}@ohsu.edu Dept of Ophthalmology, Casey Eye Institute, Oregon Health & Science University, Portland, OR, USA.

To the best of our knowledge, we are the first to study methods of reducing the complexity of greedy by exploiting the inherent geometry of the dataset-namely, that it consists of pairwise comparisons.

- We show that, by exploiting this underlying structure, the greedy algorithm can indeed be accelerated. Using Cholesky factorization [14], the Sherman Morisson formula [15], and the pairwise comparison structure, we reduce the greedy algorithm's complexity from $O(N^2d^2K)$ to $O(N^2(K+d)+N(dK+d^2)+d^2K)$. The $O(N^2(K+d))$ term, which dominates when $N \gg K+d$, consists of an $O(N^2d)$ pre-processing step and an $O(N^2)$ computation per iteration involving only scalar operations.
- We further apply our acceleration techniques to the so-called *lazy-greedy* algorithm [16–19], which is known to perform well experimentally.
- We evaluate the execution time performance of our accelerated algorithms over both synthetic and real-life datasets, demonstrating that they significantly outperform naïve implementations. Our experiments show that we can select comparisons from a dataset involving more than 10^8 comparison pairs, each comprising 400-dimensional features, in less than an hour; a naïve implementation takes more than 10 days.

The remainder of this paper is organized as follows. We discuss related work in Section 2. Our problem formulation and our accelerated greedy algorithm can be found in Sections 3 and 4, respectively. We discuss our accelerated lazy greedy algorithms in Sec. 5, and present our numerical evaluations in Section 6. Finally, we conclude in Section 7.

Table 1: Summary of Notation

$\mathbb{N}$, $\mathbb{R}$, $\mathbb{S}_+$	sets of naturals, reals, and positive definite matrices
N	number of samples in dataset
d	sample dimension (i.e., number of features)
$\mathcal{N}$	dataset of samples
i, j	sample indices in $\mathcal{N}$
$\mathcal{C}$	set of pairwise comparisons
$\mathcal{A}$	the initial set with absolute labels
K	number of comparisons to be collected
$\mathcal{S}$	subset of $\mathcal{C}$ to be collected
λ	regularization parameter in $\mathbb{R}_+$
$\boldsymbol{x}_i$	feature vector of sample i in $\mathbb{R}^d$
$\boldsymbol{x}_{i,j}$	$\boldsymbol{x}_i - \boldsymbol{x}_j$
$\mathbf{X}$	matrix of feature vectors $\boldsymbol{x}_i$, $i \in \mathcal{N}$
$\mathbf{A}$	matrix used in D-optimality criterion, given by (3.6)
f	submodular objective
y_i	absolute label of sample i
$y_{i,j}$	comparison outcome between i and j
s_i	Bradley-Terry score for sample i
$\boldsymbol{\beta}$	parameter vector/model in $\mathbb{R}^d$
Ω	abstract set in submodular maximization (for us, $\Omega = \mathcal{C}$)
e	abstract element in Ω (for us, $e = (i,j) \in \mathcal{C}$)
$\boldsymbol{x}_e$	$\boldsymbol{x}_i - \boldsymbol{x}_j$, where $e = (i,j)$
$\Delta(e\|\mathcal{S})$	marginal gain $f(\mathcal{S} \cup e) - f(\mathcal{S})$
d_e	proxy for marginal gain of element e
$\mathbf{U}$	Cholesky factor of matrix $\mathbf{A}^{-1} = \mathbf{U}^T\mathbf{U}$
$\boldsymbol{z}_i$	vectors used in Factorization Greedy, equal to $\mathbf{U}\boldsymbol{x}_i$
$\boldsymbol{v}$	auxiliary vector used in Scalar Greedy

2 Related Work

Integrating classification and pairwise comparison labels has received considerable attention recently [2, 20–22]. Integrating regression labels with ranking information was proposed in [2] as a means to improve regression outcomes in label-imbalanced datasets, and similar approaches have been used to incorporate both "pointwise" and "pairwise" labels in image classification tasks [20, 22]. Penalties used in this literature are variants of the MAP estimation we describe in Sec. 3, and are directly related to our Bradley-Terry generative model. None of these works however deal with the problem of how to collect pairwise comparison labels.

Experimental design (a.k.a. batch active learning) is classic [12]. Mutual information is a a commonly used objective [23, 24], which is monotone submodular under certain conditions [25]. Applying this objective to our generative model retains submodularity but, as in other settings [26], both (a) computing the posterior of the model, as well as (b) evaluating the function when having access to this posterior, are intractable. Many natural objectives are submodular, and are thus amenable to approximation via the greedy algorithm by Nemhauser et al. [13]; indeed, submodularity arises in a broad array of active learning problems [27, 28].

Our setting is closest to–and motivated by–work by a series of papers that study experimental design in the context of comparisons. Jamieson and Nowak [29] assume the existence of a total ordering, and which is learned in the absence of features. Grasshof et al. [30] and Glickman et al. [31] study experimental design on the Bradley-Terry model, again without features. They use D-Optimal design and KL-divergence as optimization objectives, respectively. Closer to our setting, Guo et al. [5] study four different submodular experimental design objectives, including D-optimality, Mutual Information, Information Entropy, and Fisher Information, in the high-dimensional setting. The authors establish experimentally that Mutual Information performs best, but is intractable, while D-optimality is a close second. Guo et al. implement only the naïve greedy algorithm, whose complexity is $O(N^2d^2K)$, and do not exploit the underlying structure of the problem to accelerate the algorithm; this limits their experiments to datasets with no more than 10^4 comparisons. We depart in identifying ways to exploit this structure to drastically accelerate the greedy algorithm, enabling us to solve problems with 10^8 comparisons in less than an hour.

3 Problem Formulation

Consider a setting in which data samples are labeled by an expert. Given a sample to label, the expert produces

a binary *absolute* label, indicating the sample's class. Given two different samples, the expert produces a *comparison* label. Comparison labels are also binary and indicate precedence with respect to the classification outcome. For example, for a medical diagnosis problem, absolute labels indicate the existence of disease, while comparison labels indicate the relative severity between two samples. An experimenter has access to noisy absolute labels generated by this expert. At the same time, the experimenter wishes to augment the dataset by adding comparison labels. As comparison labels are numerous (quadratic in the dataset size) and their acquisition is time-consuming, the experimenter collects only a subset of all possible comparison labels.

Formally, the experimenter has access to N samples, indexed by $i \in \mathcal{N} \equiv \{1, \ldots, N\}$. Every sample has a feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$, known to the experimenter; we denote by $\mathbf{X} = [\mathbf{x}_i]_{i \in \mathcal{N}} \in \mathbb{R}^{n \times d}$ the matrix of feature vectors. For some set $\mathcal{A} \subseteq \mathcal{N}$, the experimenter has access to binary absolute labels $y_i \in \{+1, -1\}$, $i \in \mathcal{A}$, generated by the expert. We define $\mathcal{C} \equiv \{(i,j) : i, j \in \mathcal{N}, i < j\}$ to be the set of possible pairwise comparisons.

3.1 Experimental Design. The experimenter wishes to augment the existing dataset of absolute labels by adding comparison labels $y_{i,j} \in \{+1, -1\}$, where $(i,j) \in \mathcal{C}$. It is expensive and time consuming to collect all $|\mathcal{C}| = \frac{N(N-1)}{2}$ comparison labels. The experimenter thus collects K labels from a subset $\mathcal{S} \subseteq \mathcal{C}$, where $|\mathcal{S}| = K$. To determine the optimal such set $\mathcal{S}^*$, the experimenter solves:

$$\begin{aligned} &\text{Maximize} \quad f(\mathcal{S}) - f(\emptyset), \\ &\text{subj. to} \quad \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| = K. \end{aligned} \tag{3.1}$$

where objective $f : 2^{|\mathcal{C}|} \to \mathbb{R}$ captures how informative samples in $\mathcal{S}$ are. We use the objective:

$$f(\mathcal{S}) = \log\det(\lambda \mathbf{I}_d + \sum_{i \in \mathcal{A}} \boldsymbol{x}_i \boldsymbol{x}_i^T + \sum_{(i,j) \in \mathcal{S}} \boldsymbol{x}_{i,j} \boldsymbol{x}_{i,j}^T) \tag{3.2}$$

where $\boldsymbol{x}_{i,j} = \boldsymbol{x}_i - \boldsymbol{x}_j$, $\lambda > 0$ is a positive value, and $I_d \in \mathbb{R}^{d \times d}$ is the d-dimensional identity matrix. As above, sets $\mathcal{A}$ and $\mathcal{S}$ represent the set of absolute labels observed already and the set of comparisons to be collected, respectively.

Objective (3.2) is motivated by D-optimal design [11], assuming a Bradley-Terry generative model for comparison labels [32]. In particular, (3.2) is the negative log entropy of a linear model learned under Gaussian noise [11], and has been observed to have excellent performance as an experimental design objective compared to a broad array of competitors, including Mutual Information and Fisher Information [5, 33]. Before elaborating on how to solve (3.1), we briefly discuss how (3.2) arises under the Bradley-Terry model below.

3.2 D-Optimal Design Under the Bradley-Terry Model. Assume that absolute and comparison labels are generated according to the following probabilistic model. First, there exists a parameter vector $\boldsymbol{\beta} \in \mathbb{R}^d$, sampled from a Gaussian prior $\mathtt{N}(0, \sigma^2 \mathbf{I})$, such that for all $i \in \mathcal{N}$ and all $(i,j) \in \mathcal{C}$ the absolute labels y_i and comparison labels $y_{i,j}$ are independent conditioned on $\boldsymbol{\beta}$. Second, the conditional distribution of y_i given $\boldsymbol{x}_i$ and $\boldsymbol{\beta}$ is given by a logistic model, i.e.,

$$\mathbf{P}(y_i = +1 | \boldsymbol{x}_i, \boldsymbol{\beta}) = \tfrac{1}{1+\exp(-\boldsymbol{\beta}^T \boldsymbol{x}_i)}, \quad i \in \mathcal{N}. \tag{3.3}$$

Finally, the conditional distribution of $y_{i,j}$ given $\boldsymbol{x}_i, \boldsymbol{x}_j$ and $\boldsymbol{\beta}$ is given by the following Bradley-Terry model [32]: every sample $i \in \mathcal{N}$ is associated with a parameter $s(\boldsymbol{x}_i, \boldsymbol{\beta}) = \exp(\boldsymbol{\beta}^T \boldsymbol{x}_i) \in \mathbb{R}_+$ such that, for all $(i,j) \in \mathcal{C}$,

$$\mathbf{P}(y_{i,j} = +1 | \boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{\beta}) = \tfrac{s(\boldsymbol{x}_i, \boldsymbol{\beta})}{s(\boldsymbol{x}_i, \boldsymbol{\beta}) + s(\boldsymbol{x}_j, \boldsymbol{\beta})} \tag{3.4}$$

Intuitively, score $s(\boldsymbol{x}, \boldsymbol{\beta})$ captures the propensity of input $\boldsymbol{x}$ to receive a positive absolute label, *as well as* to be selected when compared to other objects.

The advantage of the generative model (3.3)-(3.4) is that it leads to a tractable Maximum A-Posteriori (MAP) estimation procedure for learning $\boldsymbol{\beta}$. Indeed, after collecting both absolute *and* comparison labels, the experimenter learns $\boldsymbol{\beta}$ by minimizing the following negative log-likelihood loss function:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}; \mathcal{A}, \mathcal{S}) =& \lambda ||\boldsymbol{\beta}||_2^2 + \textstyle\sum_{i \in \mathcal{A}} \mathbf{log}(1 + e^{-y_i \boldsymbol{\beta}^T \boldsymbol{x}_i}) \\ &+ \textstyle\sum_{(i,j) \in \mathcal{S}} \mathbf{log}(1 + e^{-y_{i,j} \boldsymbol{\beta}^T (\boldsymbol{x}_i - \boldsymbol{x}_j)}), \end{aligned} \tag{3.5}$$

where the coefficient λ equals $1/\sigma^2$. The loss $\mathcal{L}(\boldsymbol{\beta}; \mathcal{A}, \mathcal{S})$ is convex in $\boldsymbol{\beta}$; in fact, it can be seen as a special form of logistic regression, in which the covariates of comparison labels $y_{i,j}$ are given precisely by $\boldsymbol{x}_{i,j} = \boldsymbol{x}_i - \boldsymbol{x}_j$. Matrix

$$\mathbf{A}(\mathcal{S}) = \lambda \mathbf{I}_d + \textstyle\sum_{i \in \mathcal{A}} \boldsymbol{x}_i \boldsymbol{x}_i^T + \sum_{(i,j) \in \mathcal{S}} \boldsymbol{x}_{i,j} \boldsymbol{x}_{i,j}^T \tag{3.6}$$

used in our objective (3.2) is the Fisher information matrix resulting from (3.5), when the underlying logistic regression is approximated by a linear regression.

3.3 Greedy Optimization. Unfortunately, problem (3.1) is NP hard both for the D-optimality objective (3.2) as well as for many other objective functions of interest [5, 25, 27]. However, we can produce an approximation algorithm using the theory of submodular functions. A set function $f : 2^\Omega \to \mathbb{R}$ is *submodular* if $f(\mathcal{T} \cup \{z\}) - f(\mathcal{T}) \geqslant f(\mathcal{D} \cup \{z\}) - f(\mathcal{D})$ for all

Algorithm 1 Greedy Algorithm

procedure GREEDY(f, Ω)
  PREPROCESSING( )
  while $|\mathcal{S}| < K$ **do**
    $e^* = \text{FINDMAX}(\mathcal{S})$
    UPDATES($\mathcal{S}, e^*$)
  end while
  return $\mathcal{S}$
end procedure
procedure PREPROCESSING( )
  Set $\mathcal{S} = \emptyset$
end procedure
procedure FINDMAX($\mathcal{S}$)
  return $e^* = \underset{e \in \Omega \setminus \mathcal{S}}{\textbf{argmax}}\ \Delta(e|\mathcal{S})$ ▷ $\Delta(e|\mathcal{S})$ given by (3.8).
end procedure
procedure UPDATES($\mathcal{S}, e^*$)
  Set $\mathcal{S} = \mathcal{S} \cup e^*$
end procedure

Algorithm 2 Naïve Greedy

Naïve greedy algorithm, as described in Sec. 4.1. The main GREEDY procedure is the same as in Alg. 1.

procedure PREPROCESSING($\mathbf{X}$)
  Compute $\mathbf{A}_0^{-1} = (\lambda I_d + \sum_{i \in \mathcal{A}} \boldsymbol{x}_i \boldsymbol{x}_i^T)^{-1}$; Set $\mathbf{A}^{-1} = \mathbf{A}_0^{-1}$; Set $\mathcal{S} = \emptyset$
end procedure
procedure FINDMAX($\mathcal{S}$)
  Compute $d_e = \boldsymbol{x}_e \mathbf{A}^{-1} \boldsymbol{x}_e$ for $e \in \mathcal{C} \setminus \mathcal{S}$
  return $e^* = \underset{e \in \mathcal{C} \setminus \mathcal{S}}{\textbf{argmax}}\ d_e$
end procedure
procedure UPDATES($\mathcal{S}, e^*$)
  Set $\mathcal{S} = \mathcal{S} \cup e^*$; Set $\mathbf{A}^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1} \boldsymbol{x}_{e^*} \boldsymbol{x}_{e^*}^T \mathbf{A}^{-1}}{1 + \boldsymbol{x}_{e^*}^T \mathbf{A}^{-1} \boldsymbol{x}_{e^*}}$
end procedure

$\mathcal{T} \subseteq \mathcal{D} \subseteq \Omega$ and $z \in \Omega$. Function f is called *monotone* if $f(\mathcal{D} \cup \{z\}) - f(\mathcal{D}) \geqslant 0$ for all $\mathcal{D} \subseteq \Omega$ and $z \in \Omega$. The *greedy* algorithm, summarized in Alg. 1, solves problem

$$\begin{aligned} &\text{Maximize} \quad f(\mathcal{S}), \\ &\text{s.t.} \quad |\mathcal{S}| \leqslant K, \mathcal{S} \subseteq \Omega, \end{aligned} \tag{3.7}$$

where f is monotone submodular over set Ω. Starting from $\mathcal{S} = \emptyset$, the algorithm iteratively adds the element e to the present set $\mathcal{S}$ that maximizes the *marginal gain*:

$$\Delta(e|\mathcal{S}) = f(\mathcal{S} \cup e) - f(\mathcal{S}), \tag{3.8}$$

among all elements $e \in \Omega \setminus \mathcal{S}$; this is repeated until $|\mathcal{S}| = K$. The following guarantee holds:

THEOREM 3.1. *Nemhauser et al. [13] If f is monotone submodular, the set $\mathcal{S}$ returned by Alg. 1 satisfies: $f(\mathcal{S}) - f(\emptyset) \geqslant (1 - 1/e)(f(\mathcal{S}^*) - f(\emptyset))$, where $\mathcal{S}^*$ is the optimal solution to Eq.* (3.7).

Objective (3.2) is indeed monotone submodular. However, set $\Omega = \mathcal{C}$ is *quadratic* in the number of inputs N. This is prohibitive for large datasets, particularly when marginal gains $\Delta(e|\mathcal{S})$ are themselves expensive to compute. As described in the next section, for f given by (3.2), each marginal gain computation is $O(d^2)$; this motivates us to accelerate Alg. 1.

4 Accelerating The Greedy Algorithm

In this section, we describe how to accelerate the greedy algorithm, improving its complexity from $O(N^2 d^2 K)$ to $O(N^2(d + K) + N(dK + d^2) + Kd^2)$. In doing so, we exploit the inherent structure of set $\Omega = \mathcal{C}$, namely, that it comprises pairwise comparisons. When $N \gg d + K$, the dominant term is $O(N^2(d + K))$; constants in this term amount to the time to compute 1 scalar multiplication and 1 scalar addition; as such, the algorithm scales very well in practice (see Sec. 6).

Before presenting our accelerated method, we first review a naïve implementation (*Naïve Greedy*) of Alg. 1 applied to our problem (3.1). We also construct an intermediate algorithm (*Factorization Greedy*), with slightly improved complexity ($O(Nd^2K + N^2dK)$) over Naïve Greedy. Finally, we present our fastest algorithm (*Scalar Greedy*), that attains the aforementioned guarantee. We present Factorization Greedy both for the sake of clarity, but also because its lazy implementation, presented in Section 5, has advantages over the corresponding Scalar Greedy algorithm. All algorithms receive the sample feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ as input.

4.1 Naïve Greedy.

Our first "naïve" implementation slightly improves upon the abstract greedy algorithm (Alg. 1), which operates on the value oracle model, by (a) computing a simpler version of gains $\Delta(e|\mathcal{S})$, and (b) speeding-up matrix inversion via the Sherman-Morisson formula [15]. For f given by Eq. (3.2), by the matrix determinant lemma [34]:

$$\Delta(e|\mathcal{S}) = \log(1 + \boldsymbol{x}_e^T \mathbf{A}^{-1} \boldsymbol{x}_e), \tag{4.9}$$

where $\mathbf{A} = \mathbf{A}(\mathcal{S}) \equiv (\lambda \mathbf{I}_d + \sum_{i \in \mathcal{A}} \boldsymbol{x}_i \boldsymbol{x}_i^T + \sum_{e \in \mathcal{S}} \boldsymbol{x}_e \boldsymbol{x}_e^T) \in \mathbb{S}_+^d$ and $\boldsymbol{x}_e = \boldsymbol{x}_{i,j} = \boldsymbol{x}_i - \boldsymbol{x}_j$ for all $e \equiv (i, j) \in \mathcal{C} \setminus \mathcal{S}$. As $\log(1 + s)$ is monotone in s, to implement FINDMAX in Alg. 1, it suffices to compute the maximum among

$$d_e = d_e(S) \equiv \boldsymbol{x}_e^T \mathbf{A}^{-1} \boldsymbol{x}_e, \quad e = (i, j) \in \mathcal{C} \setminus \mathcal{S}. \tag{4.10}$$

We call d_e, $e \in \mathcal{C}$, the *proxy* marginal gain. We further reduce computation costs using the fact that

$$\mathbf{A}^{-1}(\mathcal{S} \cup e^*) = \mathbf{A}^{-1}(\mathcal{S}) - \frac{\mathbf{A}^{-1}(\mathcal{S}) \boldsymbol{x}_{e^*} \boldsymbol{x}_{e^*}^T \mathbf{A}^{-1}(\mathcal{S})}{1 + \boldsymbol{x}_{e^*}^T \mathbf{A}^{-1}(\mathcal{S}) \boldsymbol{x}_{e^*}}, \tag{4.11}$$

by the Sherman Morrison formula [15]. These two observations lead to the implementation of the naïve greedy algorithm presented in Alg. 2. The algorithm uses the same main Greedy procedure as Alg. 1. In preprocessing, we initialize matrix $\mathbf{A}^{-1}$. At each iteration, we find element e^* that maximizes d_e rather than $\Delta(e|\mathcal{S})$, and subsequently update $\mathbf{A}^{-1}$ via (4.11).

Algorithm 3 Factorization Greedy

Factorization greedy algorithm, as described in Sec. 4.2. The main GREEDY procedure is the same as in Alg. 1, while the PREPROCESSING and UPDATES are the same as Alg. 2.

procedure FINDMAX($\mathcal{S}$)
  Factorize the matrix $\mathbf{A}^{-1}$ into $\mathbf{A}^{-1} = \mathbf{U}^T\mathbf{U}$ by Cholesky factorization.
  Compute and save $\boldsymbol{z}_i = \mathbf{U}\boldsymbol{x}_i$ for all $i \in \mathcal{N}$.
  Compute and save $d_e = ||\boldsymbol{z}_i - \boldsymbol{z}_j||_2^2$ for all $e \in \mathcal{C} \setminus \mathcal{S}$.
  return $e^* = \underset{e \in \mathcal{C}\setminus\mathcal{S}}{\mathbf{argmax}}\ d_e$
end procedure

Inverting matrix $\mathbf{A}_0$ has complexity[1] $O(d^{2.37})$, though for small $\mathcal{A}$ the Sherman-Morisson formula can be used again to reduce this to $O(d^2|\mathcal{A}|)$. Computing d_e and updating $\mathbf{A}^{-1}$ via the Sherman Morrison formula have complexity $O(d^2)$. Hence, Alg. 2 has a total complexity $O(N^2d^2K)$, which scales poorly for high N and d. Note that the $O(d^{2.37})$ term in pre-processing is dominated by higher order terms and therefore ignored; this holds for all algorithms in this section.

4.2 Factorization Greedy. Naïve Greedy requires $O(N^2d^2)$ operations per iteration. To avoid this, we exploit the pairwise comparison structure of $\boldsymbol{x}_e = \boldsymbol{x}_i - \boldsymbol{x}_j$, for $e = (i, j) \in \mathcal{C}$. Note that positive definite matrix $\mathbf{A}^{-1}$ can be factorized into $\mathbf{A}^{-1} = \mathbf{U}^T\mathbf{U}$ by Cholesky factorization , where matrix $\mathbf{U}$ is an upper triangular matrix. Then, d_e satisfies:

$$d_e = \boldsymbol{x}_e^T\mathbf{A}^{-1}\boldsymbol{x}_e = ||\mathbf{U}\boldsymbol{x}_e||_2^2 = ||\mathbf{U}\boldsymbol{x}_i - \mathbf{U}\boldsymbol{x}_j||_2^2. \tag{4.12}$$

This gives rise to the following algorithm, summarized in Alg. 3. PREPROCESSING and UPDATES are as in the Naïve Greedy algorithm (Alg. 2). For FINDMAX, in each iteration, we first factorize the matrix $\mathbf{A}^{-1}$ into $\mathbf{U}^T\mathbf{U}$ and calculate and save $\boldsymbol{z}_i = \mathbf{U}\boldsymbol{x}_i$ for all $i \in \mathcal{N}$. Then we calculate d_e via Eq. (4.12) for all $e \in \mathcal{C} \setminus \mathcal{S}$, and return the maximal element. Cholesky factorization has $O(d^{2.37})$ complexity [35]. Computing $\mathbf{U}\boldsymbol{x}_i$ for all $i \in \mathcal{N}$ involves $O(Nd^2)$ computations, while computing all d_e, $e \in \mathcal{C} \setminus \mathcal{S}$, via Eq. (4.12) requires $O(N^2d)$ computations. Hence, the complexity of FINDMAX in Alg. 3 is $O(Nd^2 + N^2d)$, and the entire Factorization Greedy algorithm has complexity $O(Nd^2K + N^2dK)$.

4.3 Scalar Greedy. In both previous algorithms, d_e is computed from scratch, not taking advantage of the previous iteration's computation. Let d_e, d_e' be the values of the (proxy) marginal gain for e at iterations k and $k + 1$, respectively. By the Sherman Morrison formula:

$$d_e^{'} = d_e - \boldsymbol{x}_e^T \frac{\mathbf{A}^{-1}\boldsymbol{x}_{e^*}\boldsymbol{x}_{e^*}^T\mathbf{A}^{-1}}{1+\boldsymbol{x}_{e^*}^T\mathbf{A}^{-1}\boldsymbol{x}_{e^*}}\boldsymbol{x}_e = d_e - (\boldsymbol{x}_e^T\boldsymbol{v})^2, \tag{4.13}$$

[1] As matrix inversion has the same complexity as matrix multiplication.

Algorithm 4 Scalar Greedy

Scalar greedy algorithm, as described in Sec. 4.3. The main GREEDY procedure is the same as in Alg. 1.

procedure PREPROCESSING($\mathbf{X}$)
  Compute $\mathbf{A}_0^{-1} = (\lambda I_d + \sum_{i \in \mathcal{A}} \boldsymbol{x}_i\boldsymbol{x}_i^T)^{-1}$; Set $\mathbf{A}^{-1} = \mathbf{A}_0^{-1}$; Set $\mathcal{S} = \emptyset$.
  Compute $\mathbf{U}$: factorize the matrix $\mathbf{A}_0^{-1}$ into $\mathbf{U}^T\mathbf{U}$ by Cholesky factorization.
  Compute and save $\boldsymbol{z}_i = \mathbf{U}\boldsymbol{x}_i$ for all $i \in \mathcal{N}$.
  Compute and save $d_e = ||\boldsymbol{z}_i - \boldsymbol{z}_j||_2^2$ for all $e \in \mathcal{C}$.
end procedure

procedure FINDMAX($\mathcal{S}$)
  return $e^* = \underset{e \in \mathcal{C}\setminus\mathcal{S}}{\mathbf{argmax}}\ d_e$
end procedure

procedure UPDATES($\mathcal{S}, e^*$)
  $\mathcal{S} = \mathcal{S} \cup e^*$
  Compute $\boldsymbol{v} = \frac{\mathbf{A}^{-1}\boldsymbol{x}_{e^*}}{\sqrt{1+\boldsymbol{x}_{e^*}^T\mathbf{A}^{-1}\boldsymbol{x}_{e^*}}}$.
  Compute and save $z_i = \boldsymbol{v}^T\boldsymbol{x}_i$ for all $i \in \mathcal{N}$.
  Compute and save $d_e = d_e - (z_i - z_j)^2$ for all $e \in \mathcal{C} \setminus \mathcal{S}$.
  $\mathbf{A}^{-1} = \mathbf{A}^{-1} - \boldsymbol{v}\boldsymbol{v}^T$.
end procedure

where $\boldsymbol{v} \equiv \frac{\mathbf{A}^{-1}\boldsymbol{x}_{e^*}}{\sqrt{1+\boldsymbol{x}_{e^*}^T\mathbf{A}^{-1}\boldsymbol{x}_{e^*}}}$. Exploiting the pairwise structure $\boldsymbol{x}_e = \boldsymbol{x}_i - \boldsymbol{x}_j$, we get:

$$d_e^{'} = d_e - (z_i - z_j)^2, \quad e = (i, j) \in \Omega \setminus \mathcal{S}, \tag{4.14}$$

where $z_i \equiv \boldsymbol{v}^T\boldsymbol{x}_i$. This gives rise to our final greedy implementation, summarized in Alg. 4. In PREPROCESSING, we factorize matrix $\mathbf{A}^{-1}$ into $\mathbf{A}^{-1} = \mathbf{U}^T\mathbf{U}$ and calculate d_e for all $e \in \mathcal{C}$ via Eq. (4.12). In UPDATES, we compute vector $\boldsymbol{v}$ and scalars $z_i = \boldsymbol{v}^T\boldsymbol{x}_i$ for all $i \in \mathcal{N}$. Then we update every d_e through Eq. (4.14), and $\mathbf{A}^{-1}$ via Eq. (4.11), using $\boldsymbol{v}$ again.

Preprocessing requires $O(d^{2.37})$ time for the matrix inversion and Cholesky factorization, $O(Nd^2)$ for computing all $\boldsymbol{z}_i$, $i \in \mathcal{N}$, and $O(N^2d)$ for computing all d_e, $e \in \mathcal{C}$. Computing $\boldsymbol{v}$ and updating $\mathbf{A}^{-1}$ via the Sherman Morrison formula have complexity $O(d^2)$ for one iteration. Computing $\boldsymbol{v}^T\boldsymbol{x}_i$, for all $i \in \mathcal{N}$ involves $O(Nd)$ computations, while updating d_e, for all $e \in \Omega \setminus \mathcal{S}$ via Eq. (4.14) requires $O(N^2)$ scalar computations. Hence, the total complexity is $O(N^2(K + d) + N(dK + d^2) + d^2K)$. The $O(N^2(K+d))$ term, due to the computation of d_e, $e \in \mathcal{C}$ in preprocessing and at each iteration, dominates the rest when $N \gg K, d$. The constant in this term thus involves only the time to perform 1 scalar subtraction and 1 scalar multiplication; as such, it remains tractable even for large datasets.

5 Accelerating the Lazy Greedy Algorithm

The *lazy greedy* algorithm [16, 36, 37] is a well-known variant of the standard greedy algorithm; it reduces execution time by avoiding the computation of all $|\Omega \setminus \mathcal{S}|$ marginal gains $\Delta(e|\mathcal{S})$ at each iteration. This is accomplished via a "lazy" evaluation of each inner loop in FINDMAX in Alg. 1; though no bounds exist on

the worst-case amortized complexity of lazy greedy, it performs quite well in practice [28, 38].

We employ the same optimizations we describe in Sec. 4 to also accelerate the lazy greedy algorithm. Each of the three accelerations we mentioned in the previous section yield corresponding "lazy" versions, namely *Naïve Lazy Greedy*, *Factorization Lazy Greedy*, and *Scalar Lazy Greedy*, respectively. A full description of these three versions can be found in the extended version of this paper [39].

In the latter two cases (Factorization and Scalar Lazy Greedy), an additional form of accelaration can be used. Due to lazy evaluation, not all quantities such as, e.g., z_i (in Line 3 of Alg. 3 and Line 4 of Alg. 4) are used throughout an iteration. Such quantities can either be pre-computed at each iteration, or computed on the spot, as needed. Though the latter appears to be a faster approach, in practice, it is not always the case: pre-computation can be faster, as matrix-vector multiplication is more efficient than for-loops in many languages. As discussed in Sec. 6, we implement both variants in python, and refer to them as *with-precomputation* and *with-memoization*, respectively.

6 Evaluation

We use synthetic and real datasets to evaluate the performance of different greedy and lazy greedy algorithms.[2] We evaluate these algorithms both in terms of execution time and classification performance w.r.t. accuracy of predictions, after labels are collected.

6.1 Evaluation Setup.

We begin by describing our evaluation setup.

Datasets. In our synthetic dataset, the absolute feature vectors $\boldsymbol{x}_i \in \mathbb{R}^d$, $i \in \mathcal{N}$, are sampled from a Gaussian distribution $\mathtt{N}(\mathbf{0}, \sigma_x I_d)$ with feature dimension d ranging from 20 to 400 and dataset size N ranging from 500 to 15000. We also sample a parameter vector $\widetilde{\boldsymbol{\beta}}$ from Gaussian distribution $\mathtt{N}(\mathbf{0}, \sigma_{\boldsymbol{\beta}} I_d)$. We generate absolute labels y_i, $i \in \mathcal{N}$, using Eq. (3.3) with $\boldsymbol{\beta} = \widetilde{\boldsymbol{\beta}}/C_a$, where C_a is a positive scalar. Finally, we generate $|\mathcal{C}|$ comparison labels via Eq. (3.4), with $\boldsymbol{\beta} = \widetilde{\boldsymbol{\beta}}$. Parameter C_a allows us to control the relative noise ratio between absolute and comparison labels; we set it to $C_a = 1.2$ in our experiments.

We also use seven real-life datasets, summarized in Fig. 1(a). The first four (ROP, Sushi, Netflix, Camra) contain comparison labels; the remaining (ROP5K, SIFT, and Microsoft URL) do not, and are used only for measuring the execution time of our algorithms. A detailed description of all datasets is in [39].

Algorithms. We implement eight greedy algorithms: Naïve Greedy (NG), Factorization Greedy (FG), Scalar Greedy (SG), Naïve Lazy Greedy (NL), Factorization Lazy Greedy with Pre-Computation (FLP), Factorization Lazy Greedy with Memoization (FLM), Scalar Lazy Greedy with Pre-Computation (SLP), and Scalar Lazy Greedy with Memoization (SLM). In each dataset, we set λ in (3.2) to about 10^{-5} the average norm of feature vectors (see Fig 1(a)).

We also implement the greedy algorithm with Mutual Information (Mut), Fisher Information (Fisher), and Entropy (Ent) objectives, as described in [5] (also reviewed in [39]). Finally, we implement a Random (Ran) baseline method, in which the set $\mathcal{S}$ is selected uniformly at random from $\mathcal{C}_{trn}$.

Experiment Setup. In each experiment, we partition the dataset $\mathcal{N}$ into three datasets: a training set $\mathcal{N}_{\mathtt{trn}}$, a test set $\mathcal{N}_{\mathtt{tst}}$, and a validation set $\mathcal{N}_{\mathtt{val}}$. Wherever available, we denote by $\mathcal{C}_{\mathtt{trn}} \subset \mathcal{C}$ the corresponding comparison set restricted to pairs of objects in $\mathcal{N}_{\mathtt{trn}}$. We select a random subset $\mathcal{A}$ from $\mathcal{N}_{\mathtt{trn}}$ whose absolute labels y_i, $i \in \mathcal{A}$ are presumed revealed to the experimenter. Then we use our greedy algorithms to select $\mathcal{S} \subset \mathcal{C}_{\mathtt{trn}}, |\mathcal{S}| = K$. We record the running time t_K of each algorithm for different values of $K \in \mathbb{N}$ executed on the training set. For synthetic data, we repeat each experiment 150 times, each time with a different randomly generated dataset; we report average t_k values, as well as standard deviations. For real datasets including absolute labels (ROP, Sushi, Netflix, Camra), we also repeat experiments 150 times, each time with a different randomly selected set $\mathcal{A}$.

For both synthetic and real datasets for which we have comparison labels (ROP, Sushi, Netflix, Camra), we collect the K comparison labels from $\mathcal{S}$ and train a model $\boldsymbol{\beta} \in \mathbb{R}^d$ using the labels in $\mathcal{A}$ and $\mathcal{S}$ via MAP estimation (3.5), and predict both comparison and absolute labels in the test set. In doing so, we select the parameter λ in (3.5) as the value that maximizes AUC on the validation set. Especially, for ROP, we measure the performance w.r.t. the reference standard diagnosis (RSD) label prediction rather than absolute labels, even though the model is trained on (noisier) absolute labels. For each dataset, we perform cross validation, repeating the partition to training and test datasets and keeping the validation set fixed. To produce confidence intervals, each 4-fold cross validation is repeated 150 times, i.e., over 150 different random data shuffles (for the Netflix dataset, the experiment is executed for 150 users).

6.2 Execution Time Performance.

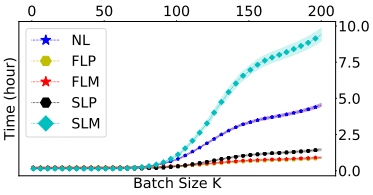
We first study the execution time in terms of N, d, and K.

Dependence on N. In Fig. 2 we plot the running time

[2] Our code is publicly available at: `https://github.com/neu-spiral/AcceleratedExperimentalDesign`

Dataset	N	d	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{C}_{\texttt{trn}}\rvert$	$T_n(s)$	$A_{\texttt{ab}}$	$A_{\texttt{ac}}$	λ
ROP	100	156	30	4950	1770	6.30	0.938	0.858	0.0001
Sushi	100	20	15	4821	1560–1762	0.878	0.932	0.682	0.0001
Netflix	833–1198	30	20	180K–540K	160K–450K	88.4	0.811	0.871	0.0001
CAMRa	896–3300	10	20	400K–5M	400K-5M	459	0.77	0.79	0.0001
SIFT	3000	128	30	4.5M	4.5M	12K	N/A	N/A	0.001
ROP5K	3000	143	30	4.5M	4.5M	14K	N/A	N/A	0.0001
MSLR	325-996	134	30	52K–500K	52K–500K	252	N/A	N/A	0.0001

(a) Dataset Summary



(b) Scalability

Figure 1: *(a)* Summary of real datasets. Columns N, d, $|\mathcal{A}|$, and $|\mathcal{C}|$ indicate the number of samples, the dimension, the number of absolute labels, and the number of comparisons, respectively. Column $|\mathcal{C}_{\texttt{trn}}|$ is size of train comparison set, while $T_{\texttt{n}}$ is the execution time under NG for $\mathcal{S} \subseteq \mathcal{C}_{\texttt{trn}}, |\mathcal{S}| = 200$. Columns $A_{\texttt{ab}}, A_{\texttt{ac}}$, indicate the test set AUC of absolute and comparison labels, respectively, when $\mathcal{S} = \mathcal{C}_{\texttt{trn}}$; we report these only for datasets for which we have comparisons. Finally, λ is the positive value in Eq. (3.2). For Netflix and Camra, we indicate value ranges across 150 users as appropriate; for MSLR, we report ranges for 150 queries. *(b)* Scalability of Lazy Greedy Algorithm. Time execution result for large synthetic dataset with $N = 15000$, $d = 400$, and $|\mathcal{C}| = 1.125 \times 10^8$.

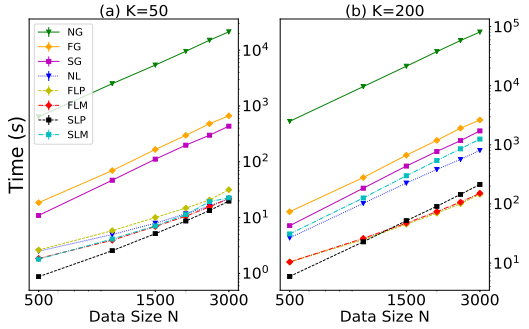


Figure 2: Average execution time for synthetic data under different sample sizes N, with feature dimension $d = 400$ and K set to $K = 50$ in subfigure (a) and $K = 200$ in subfigure (b).

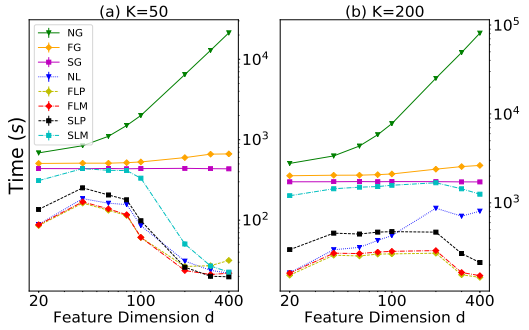


Figure 3: Average execution time for synthetic data under different feature dimensions d, with sample size $N = 3000$ and K set to $K = 50$ in subfigure (a) and $K = 200$ in subfigure (b).

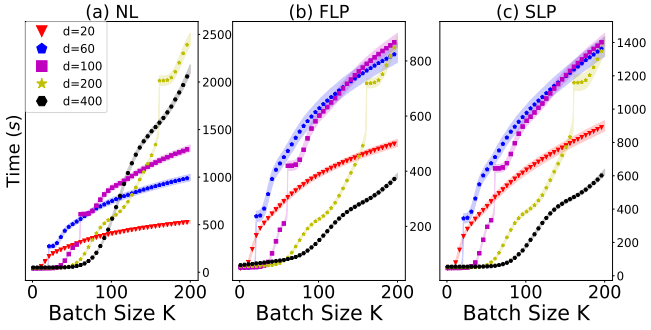


Figure 4: Average execution time as a function of K on synthetic data with $N = 5000$ for three lazy greedy algorithms.

as a function of the data size N for synthetic datasets. The quadratic–$O(N^2)$–scaling of all algorithms is clearly

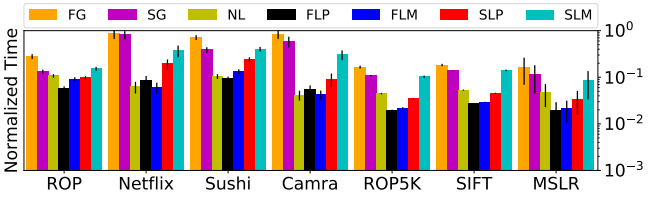


Figure 5: Execution time for different datasets and different algorithms, normalized by NG execution time (see col. $T_n(s)$ in Fig. 1(a)).

evident, although the actual execution time varies drastically between different algorithms. Both FG and SG improve upon NG by almost two orders of magnitude. Lazy algorithms improve over NG by as much as 3 orders of magnitude when $K = 50$. However, scalar lazy greedy with memoization (SLM) performs similarly to FG and SG when $K = 200$, even worse than NL.

Almost universally, pre-processing versions (FLP and SLP) outperform the corresponding memoized versions of the lazy algorithms (FLM and SLM). This is because pre-computation involves a matrix-vector multiplication: in python's NumPy library this is performed in C language, and is more efficient than the python for-loop inherent in memoization. This negates any benefit of computing only the values needed via memoization. Finally, SLP is the best performer when $K = 50$, while FLP outperforms it for large N when $K = 200$.

Dependence on d. Fig. 3 shows performance over synthetic datasets as a function of dimension d. The advantage of FG and SG over the naïve algorithm (NG) is clearly evident: the latter grows super-linearly in d. In contrast, the effect of d on FG is linear, while on SG it is almost imperceptible. A striking difference in behavior is observed in the lazy greedy algorithms, that are very sensitive to d. Indeed, these algorithms perform *poorly in lower dimensions*, with the gap between performance for low to high dimensions being sometimes close to two orders of magnitude. This is because, for high d, there is are many new dimensions to discover; as a result, almost maximal elements in the heap remain almost maximal in subsequent iterations, leading to early loop terminations. In contrast, in low d, maximality changes drastically between iterations,

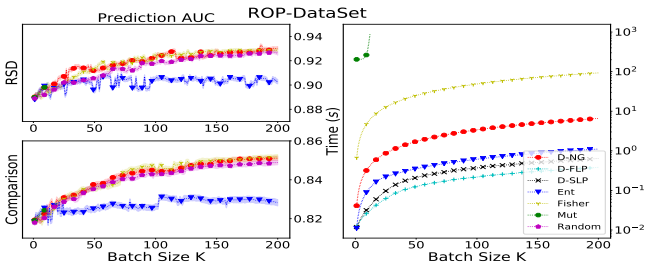


Figure 6: Test set AUC and execution time for ROP dataset, when comparisons samples are selected via D-optimal, Fisher, Entropy and Random and Mutual Information. The classifier is trained via MAP (3.5) on the training set. The left top figure is the test AUC for RSD label, the left bottom figure is the test AUC for comparison labels. The right figure is the execution time for different algorithms. For the D-optimal objective, we record the execution time for Naive Greedy, Factorization Lazy and Scalar Lazy Greedy Algorithm. We only execute Mutual Information for $K \leq 12$.

leading to full loop executions.

Dependence on K. We further explore this phenomenon in Fig. 4, that shows the dependence of lazy algorithm on K. We observe a 'jump' in execution time, indicating an expensive loop execution that contributes highly to the execution cost. The smaller d is, the earlier this jump is observed.

Scalability. All in all, we observe that our accelerations, on both standard and lazy greedy algorithms, can significantly reduce the execution time of experimental design. In Fig. 1(b), we illustrate this by running the accelerated lazy algorithms for a large synthetic dataset with $N = 15000$ and $d = 400$, containing more than 10^8 comparison pairs. The running time of Naïve Greedy (NG) on this dataset exceeds 10 days. As seen in Fig. 1, the running time can be shortened to less than 1 hour under the Factorization Lazy (FLP) algorithm. We also observe that SLM performs worse than NL, while SLP outperforms NL, again due to the advantage of matrix-vector multiplications over python for-loops.

Time Performance Evaluation on Real Datasets. Experiments on the seven real datasets further corroborate observations made over synthetic data. Fig. 5 shows the execution time normalized by the execution time of NG for each dataset (see column $T_n(s)$ in Fig. 1(a)). All algorithms yield an improvement. Lazy greedy algorithms perform well overall, but for the Sushi, CAMRa, and Netflix datasets, this improvement is diminished due to their low dimension d. The highest improvement in all algorithms compared to NG is observed in the largest of our datasets, ROP5K, where FG and SG yield an improvement of 1 order of magnitude, while SLP performs exceedingly well, leading to an improvement of 2 orders of magnitude over NG. Overall, FLP consistently improves performance over NL.

6.3 Prediction Performance

All 8 algorithms using D-Optimality as an objective produce the same selected set $\mathcal{S}$ (namely, the one determined by the greedy algorithm). We give some intuition of the quality of the model learned via MAP estimation (3.5) in comparison to competitors. This selection has been known to outperform competitors such as Fisher Information and Entropy objectives [5]. For the sake of completeness, we show in Fig. 6 the prediction quality of the resulting trained model w.r.t AUC of both absolute and comparison labels over the test set, on the ROP dataset. Remaining datasets for which we have comparison labels (Sushi, Netflix, and Camra) are shown in [39]. In all cases, estimators learned over labels collected by the greedy algorithm significantly outperform random selection. Fisher Information and Mutual Information are sometimes better, but are also exceedingly time consuming, between $10^2 - 10^3$ times slower than D-optimal NG. Finally, Entropy is fast, but the prediction performance is not as good as under NG.

We observe similar performance in the remaining datasets, shown in [39]. For the Netflix and Camra datasets, we can only scale Mutual Information and Fisher Information to $K \leq 10$, as their complexity is $O(N^2 2^K)$ and $O(N^4 K)$, respectively. In the Sushi and Netflix dataset, the best AUC comes from the D-optimal design algorithm. For the ROP dataset, the D-optimal and Fisher information outperform random for all batch sizes. For the Camra dataset, as $d = 10$, the D-optimal design algorithm outperforms Random only when the batch size is less than 100. Finally, Entropy is time efficient but has worse accuracy than all other methods. Our D-optimal Naïve Greedy and its variants have good accuracy and are time-efficient; accelarated methods FLP and SLP are even faster than Entropy.

7 Conclusion

We have shown that experimental design for pairwise comparisons under the D-optimality criterion can be significantly accelerated by exploiting the underlying geometry of pairwise comparisons. Given the prevalence of submodularity in batch active learning objectives, it would be interesting to identify methods through which these results could extend to other objectives of interest. These include objectives that are structurally similar (such as A-optimality or E-optimality [11]), as well as objectives like mutual information, for which even the function value oracle is not tractable.

8 Acknowledgement

Our work is supported by NIH (R01EY019474, P30EY10572), NSF (SCH-1622542 at MGH; SCH-1622536 and CCF-1750539 at Northeastern; SCH-1622679 at OHSU), and by unrestricted departmental funding from Research to Prevent Blindness (OHSU).

References

[1] J. K-Cramer, J. P. Campbell, D. Erdogmus, et al. *Plus disease in retinopathy of prematurity: improving diagnosis by ranking disease severity and using quantitative image analysis.* Ophthalmology, 2016.

[2] D. Sculley. *Combined regression and ranking.* In KDD, 2010.

[3] M. S. Desarkar, S. Sarkar, and P. Mitra. *Aggregating preference graphs for collaborative rating prediction.* In Recsys, 2010.

[4] M. S. Desarkar, R. Saxena, and S. Sarkar. *Preference relation based matrix factorization for recommender systems.* In UMAP, 2012.

[5] Y. Guo, P. Tian, J. Kalpathy-Cramer, S. Ostmo, J. P. Campbell, M. F. Chiang, D. Erdogmus, J. Dy, and S. Ioannidis. *Experimental Design Under the Bradley-Terry Model.* In IJCAI, 2018.

[6] N. Stewart, G. DA Brown, and N. Chater. *Absolute identification by relative judgment.* Psychological Review, 2005.

[7] A. Brun, A. Hamad, O. Buffet, and A. Boyer. *Towards preference relations in recommender systems.* In ECML/PKDD, 2010.

[8] Y. Zheng, L. Zhang, X. Xie, and Wei-Ying Ma. *Mining interesting locations and travel sequences from GPS trajectories.* In WWW. ACM, 2009.

[9] Y. Koren and J. Sill. *OrdRec: an ordinal model for predicting personalized item rating distributions.* In Recsys, 2011.

[10] M. Schultz and T. Joachims. *Learning a distance metric from relative comparisons.* In NIPS, 2004.

[11] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[12] F. Pukelsheim. *Optimal design of experiments.* SIAM, 1993.

[13] G. L. Nemhauser, L. A. Wolsey, and M. L Fisher. *An analysis of approximations for maximizing submodular set functions.* Mathematical Programming, 1978.

[14] G. H. Golub and C. F. Van Loan. *Matrix computations.* JHU Press, 2012.

[15] J. Sherman and W. J. Morrison. *Adjustment of an inverse matrix corresponding to a change in one element of a given matrix.* Ann. Math. Stat, 1950.

[16] M. Minoux. *Accelerated greedy algorithms for maximizing submodular set functions.* In Optimization techniques. 1978.

[17] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. *Lazier Than Lazy Greedy.* In AAAI, 2015.

[18] H. Lin and J. Bilmes. *A class of submodular functions for document summarization.* In HLT, 2011.

[19] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. *Distributed submodular maximization: Identifying representative elements in massive data.* In NIPS, 2013.

[20] L. Chen, P. Zhang, and B. Li. *Fusing pointwise and pairwise labels for supporting user-adaptive image retrieval.* In ICMR, pp 67–74, 2015.

[21] H. Takamura and J. Tsujii. *Estimating numerical attributes by bringing together fragmentary clues.* In HLT, 2015.

[22] Y. Wang, S. Wang, J. Tang, H. Liu, and B. Li. *PPP: Joint pointwise and pairwise image label prediction.* In CVPR, 2016.

[23] J. Liepe, S. Filippi, M. Komorowski, and M. P. Stumpf. *Maximizing the information content of experiments in systems biology.* PLOS Comput. Biol, 2013.

[24] D. R. Cavagnaro, J. I. Myung, M. A. Pitt, and J. V. Kujala. *Adaptive design optimization: A mutual information-based approach to model discrimination in cognitive science.* Neural computation, 2010.

[25] A. Krause and C. E. Guestrin. *Near-optimal nonmyopic value of information in graphical models.* arXiv preprint arXiv:1207.1394, 2012.

[26] A. G. Busetto, A. Hauser, G. Krummenacher, M. Sunnåker, S. Dimopoulos, C. S. Ong, Jö. Stelling, and J. M. Buhmann. *Near-optimal experimental design for model selection in systems biology.* Bioinformatics, pp 2625–2632, 2013.

[27] A. Krause and D. Golovin. *Submodular function maximization.*, 2014.

[28] D. Golovin and A. Krause. *Adaptive submodularity: Theory and applications in active learning and stochastic optimization.* JAIR, 2011.

[29] K. G. Jamieson and R. Nowak. *Active ranking using pairwise comparisons.* In NIPS, 2011.

[30] U. Graßhoff and R. Schwabe. *Optimal design for the Bradley–Terry paired comparison model.* Statistical Methods and Applications, 2008.

[31] M. E. Glickman and S. T. Jensen. *Adaptive paired comparison design.* Journal of statistical planning and inference, pp 279–293, 2005.

[32] R. A. Bradley and M. E. Terry. *Rank analysis of incomplete block designs: I. The method of paired comparisons.* Biometrika, 1952.

[33] X. He. *Laplacian regularized D-optimal design for active learning and its application to image retrieval.* IEEE Transactions on Image Processing, 2010.

[34] D. A. Harville. *Matrix algebra from a statistician's perspective.* Springer, 1997.

[35] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[36] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. *Cost-effective outbreak detection in networks.* In KDD, 2007.

[37] A. Krause and C. Guestrin. *Near-optimal observation selection using submodular functions.* In AAAI, 2007.

[38] Z. Changshui, H. Guangdong, and W. Jun. *A fast algorithm based on the submodular property for optimization of wind turbine positioning.* Renewable Energy, 2011.

[39] Y. Guo, J. Kalpathy-Cramer, S. Ostmo, J. P. Campbell, M. F. Chiang, D. Erdogmus, J. Dy, and S. Ioannidis. *Accelerated Experimental Design for Pairwise Comparison 2019.* `http://arxiv.org/abs/1901.06080`.