Adapting Kernel Representations Online Using Submodular Maximization

Matthew Schlegel [1] **Yangchen Pan** [1] **Jiecao Chen** [1] **Martha White** [1]

Abstract

Kernel representations provide a nonlinear representation, through similarities to prototypes, but require only simple linear learning algorithms given those prototypes. In a continual learning setting, with a constant stream of observations, it is critical to have an efficient mechanism for sub-selecting prototypes amongst observations. In this work, we develop an approximately submodular criterion for this setting, and an efficient online greedy submodular maximization algorithm for optimizing the criterion. We extend streaming submodular maximization algorithms to continual learning, by removing the need for multiple passes—which is infeasible—and instead introducing the idea of coverage time. We propose a general block-diagonal approximation for the greedy update with our criterion, that enables updates linear in the number of prototypes. We empirically demonstrate the effectiveness of this approximation, in terms of approximation quality, significant runtime improvements, and effective prediction performance.

1. Introduction

Kernel representations provide an attractive approach to representation learning, by facilitating simple linear prediction algorithms and providing an interpretable representation. A kernel representation consists of mapping an input observation into similarity features, with similarities to a set of prototypes. Consequently, for an input observation, a prediction can be attributed to those prototypes that are most similar to the observation. Further, the transformation to similarity features is non-linear, enabling non-linear function approximation while using linear learning algorithms that simply optimize for weights on these transformed features. Kernel representations are universal function approximators[1] and the flexibility in choosing the kernel (similarity function) has enabled impressive prediction performance for a range of settings, including speech (Huang et al., 2014), computer vision (Mairal et al., 2014), and object recognition (Lu et al., 2014).

In a continual learning setting, such as in online learning or reinforcement learning, there is a constant, effectively unending stream of data, necessitating some care when using kernel representations. The issue arises from the choice of prototypes. Before the advent of huge increases in dataset sizes, a common choice was to use all of the training data as prototypes. This choice comes from the representer theorem, which states that for a broad class of functions, the empirical risk minimizer is a linear weighting of similarity features, to a set of prototypes that consists of the training data. For continual learning, however, the update should be independent of the the total number of samples— which is not clearly defined for continual learning. Conversely, we want to permit selection of a sufficiently large number of prototypes, to maintain sufficient modeling power. For efficient, continual updating, therefore, we require per-step prototype selection strategies that are approximately linear in the number of prototypes.

Currently, most algorithms do not satisfy the criteria for a continual learning setting. Incremental selection of prototypes has been tackled in a wide range of areas, due to the fundamental nature of this problem. Within the streaming community, approaches typically assume that the batch of data, though large, is accessible and fixed. The most related of these areas[2] include active set selection for Gaussian process regression (Seeger et al., 2003), with streaming submodular maximization approaches (Krause et al., 2008b;a; Badanidiyuru et al., 2014); incremental Nystrom methods within kernel recursive least-squares (KRLS)

[1]Department of Computer Science, Indiana University, Bloomington. Correspondence to: Martha White <martha@indiana.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

[1]Radial basis function networks are an example of a kernel representation, that have been shown to be universal function approximators (Park & Sandberg, 1991). Further, the representer theorem further characterizes the approximation capabilities under empirical risk minimization for a broad class of functions.

[2]Facility location, k-medians and k-centers are three problems that focus on selecting representative instances from a set (c.f. (Guha et al., 2003)). The criteria and algorithms are not designed with the intention to use the instances for prediction and so we do not consider them further here.

(Rudi et al., 2015)[3]; and functional gradients that sample random bases which avoid storing prototypes but require storing n scalars, for n training samples (Dai et al., 2014).

Kernel representation algorithms designed specifically for the online setting, on the other hand, are typically too computationally expensive in terms of the number of prototypes. Kernel least-mean squares (KLMS) algorithms use stochastic updates, maintaining the most recent prototypes and truncating coefficients on the oldest (Kivinen et al., 2010; Schraudolph et al., 2006; Cheng et al., 2007); though efficient given sufficient truncation, this truncation can introduce significant errors (Van Vaerenbergh & Santamaria, 2013). Random feature approximations (Rahimi & Recht, 2007) can be used online, but require a significant number of random features. Gaussian process regression approaches have online variants (Csató & Opper, 2006; Cheng & Boots, 2016), however, they inherently require at least quadratic computation to update the variance parameters. KRLS can be applied online, but has a threshold parameter that makes it difficult to control the number of prototypes and requires quadratic computation and space (Engel et al., 2004). More efficient coherence heuristic have been proposed (Richard et al., 2009; Van Vaerenbergh et al., 2010; Chen et al., 2013; Van Vaerenbergh & Santamaria, 2013), but provide no approximation quality guarantees.

In this work, we provide a simple and efficient greedy algorithm for selecting prototypes for continual learning, by extending recent work in prototype selection with submodular maximization. We introduce a generalized coherence criterion for selecting prototypes, which unifies two previously proposed criteria: the coherence criterion and the log determinant. Because this criterion is (approximately) submodular, we pursue a generalization to streaming submodular maximization algorithms. We avoid the need for multiple passes over the data—which is not possible in continual learning— by introducing the idea of coverage time, which reflects that areas of the observation space are repeatedly visited under sufficient mixing. We prove that our online submodular maximization achieves an approximation-ratio of 1/2, with a small additional approximation introduced due to coverage time and from using an estimate of the submodular function. We then provide a linear-time algorithm for approximating one instance of our submodular criterion, by exploiting the block-diagonal form of the kernel matrix. We empirically demonstrate that this approximation closely matches the true value, despite using significantly less computation, and show effective prediction performance using the corresponding kernel representation.

[3]There is a large literature on fast Nystrom methods using related approaches, such as determinant point processes for sampling landmark points (Li et al., 2016). The primary goal for these methods, however, is to approximate the full kernel matrix.

2. Using kernel representations

A kernel representation is a transformation of observations into similarity features, consisting of similarities to prototypes. A canonical example of such a representation is a radial basis function network, with radial basis kernels such as the Gaussian kernel; however, more generally any kernel similarity can be chosen. More formally, for observations $\mathbf{x} \in \mathcal{X}$, the kernel representation consists of similarities to a set of prototypes $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_b\} \subset \mathcal{X}$

$$\mathbf{x} \to [k(\mathbf{x}, \mathbf{z}_1), \ldots, k(\mathbf{x}, \mathbf{z}_b)] \in \mathbb{R}^b.$$

for kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The observations need not be numerical; as long as a similarity k can be defined between two observations, kernel representations can be used and conveniently provide a numeric feature vector in $\mathbb{R}^b$. We use the term prototype, instead of center, to emphasize that the chosen observations are representative instances, that are sub-selected from observed data.

A fundamental result for kernel representations is the representer theorem, with significant recent generalizations (Argyriou & Dinuzzo, 2014), which states that for a broad class of function spaces $\mathcal{H}$, the empirical risk minimizer $f \in \mathcal{H}$ on a training set $\{(\mathbf{x}_i, y_i\}_{i=1}^n$ has the simple form

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, \mathbf{x}_i).$$

This result makes use of a key property: the kernel function can be expressed as an inner product, $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ for some implicit expansion ϕ. The function f can be written $f = \sum_{i=1}^n \alpha_i \phi(\mathbf{x}_j)$, with

$$f(\mathbf{x}) = \langle \phi(\mathbf{x}), \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_j) \rangle = \sum_{i=1}^{n} \alpha_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_j) \rangle.$$

It is typically impractical to use all $\mathbf{x}_i$ as prototypes, and a subset needs to be chosen. Recently, there has been several papers (Krause et al., 2008b;a; Krause & Gomes, 2010; Badanidiyuru et al., 2014) showing that prototypes can be effectively selected in the streaming setting using greedy submodular maximization on the log-determinant of the kernel matrix, $\mathbf{K}_S \in \mathbb{R}^{b \times b}$ where $(\mathbf{K}_S)_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$. Given some ground set Ω and its powerset $\mathcal{P}(\Omega)$, submodular functions $g : \mathcal{P}(\Omega) \to \mathbb{R}$ are set functions, with a diminishing returns property: the addition of a point to a given set increases the value less or equal than adding a point to a subset of that set. For prototype selection, the ground set considered are sets of all observations $\mathcal{X}$, so $S \subset \Omega = \mathcal{X}$. The log-determinant of the resulting kernel matrix, $\log\det \mathbf{K}_S$, is a submodular function of S. Though maximizing submodular functions is NP-hard, greedy approximation algorithms have been shown to obtain reasonable approximation ratios, even for the streaming setting

(Krause & Gomes, 2010; Badanidiyuru et al., 2014), and the resulting algorithms are elegantly simple and theoretically sound.

In the following sections, we derive a novel criterion for prototype selection, that includes the log-determinant as a special case. Then, we provide an efficient prototype selection algorithm for the continual learning setting, using submodular maximization.

3. Selecting kernel prototypes

Many prototype selection strategies are derived based on diversity measures. The coherence criterion (Engel et al., 2004) approximates how effectively the set of prototypes spans the set of given observations. The log-determinant measures the spread of eigenvalues for the kernel matrix, and is related to information gain (Seeger, 2004). These selection criteria are designed for a finite set of observed points; here, we step back and reconsider a suitable objective for prototype selection for continual learning.

Our goal is to select prototypes that minimize distance to the optimal function. In this section, we begin from this objective and demonstrate that the coherence criterion and log-determinant are actually upper bounds on this objective, and special cases of a more general such upper bound. The analysis justifies that the log-determinant is a more suitable criteria for continual learning, which we then pursue in the remainder of this work.

3.1. Criteria to select prototypes from a finite set

Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ be a set of points, with corresponding labels $\mathbf{y}_1, \dots, \mathbf{y}_n$. We will not assume that this is a batch of data, but could rather consist of all possible observations for a finite observation space. Ideally, we would learn $S = \{\mathbf{z}_1, \dots, \mathbf{z}_b\} \subset \mathcal{X}$ and corresponding $f_{S,\beta}(\cdot) = \sum_{i=1}^b \beta_j k(\cdot, \mathbf{x}_i)$ according to the loss

$$\min_{S\subset\mathcal{X}} \min_{\boldsymbol{\beta}\in\mathbb{R}^b} \|f - f_{S,\beta}\|^2 \tag{1}$$

$$\text{where} \|f - f_{S,\beta}\| = \left\| \sum_{i=1}^n \alpha_i \phi(\mathbf{x}_i) - \sum_{j=1}^b \beta_j \phi(\mathbf{z}_j) \right\|$$

for the optimal f for the set of points $\mathcal{X}$. Because we do not have f, we derive an upper bound on this value. Introducing dummy variables $\beta_j^{(i)} \in \mathbb{R}$ such that $\beta_j = \sum_{i=1}^n \beta_j^{(i)}$,

$$(1) \leq \min_{S\subset\mathcal{X}, \boldsymbol{\beta}\in\mathbb{R}^b} \sum_{i=1}^n \left\| \alpha_i \phi(\mathbf{x}_i) - \sum_{j=1}^b \beta_j^{(i)} \phi(\mathbf{z}_j) \right\|^2$$

$$= \min_{S\subset\mathcal{X}} \sum_{i=1}^n \min_{\beta_1^{(i)},\dots,\beta_b^{(i)}} \left\| \alpha_i \phi(\mathbf{x}_i) - \sum_{j=1}^b \beta_j^{(i)} \phi(\mathbf{z}_j) \right\|^2 \tag{2}$$

For $\mathbf{k}_i \doteq [k(\mathbf{x}_i, \mathbf{z}_1), \dots, k(\mathbf{x}_i, \mathbf{z}_b)]$, the interior minimization can be re-written as

$$\min_{\boldsymbol{\beta}^{(i)}} \left\| \alpha_i \phi(\mathbf{x}_i) - \sum_{j=1}^b \beta_j^{(i)} \phi(\mathbf{z}_j) \right\|^2$$
$$= \min_{\boldsymbol{\beta}} {\boldsymbol{\beta}^{(i)}}^\top \mathbf{K}_S \boldsymbol{\beta} - 2\alpha_i {\boldsymbol{\beta}^{(i)}}^\top \mathbf{k}_i + \alpha_i^2 k(\mathbf{x}_i, \mathbf{x}_i)$$

To provide more stable approximations, we regularize

$$(2) \leq \min_{S\subset\mathcal{X}} \sum_{i=1}^n \min_{\boldsymbol{\beta}^{(i)}} \left\| \alpha_i \phi(\mathbf{x}_i) - \sum_{j=1}^b \beta_j^{(i)} \phi(\mathbf{z}_j) \right\|^2 + \lambda \|\boldsymbol{\beta}^{(i)}\|_2^2$$
$$= \min_{S\subset\mathcal{X}} \sum_{i=1}^n \min_{\boldsymbol{\beta}^{(i)}} {\boldsymbol{\beta}^{(i)}}^\top (\mathbf{K}_S + \lambda \mathbf{I}) \boldsymbol{\beta} - 2\alpha_i {\boldsymbol{\beta}^{(i)}}^\top \mathbf{k}_i + \alpha_i^2 k(\mathbf{x}_i, \mathbf{x}_i).$$

For $\lambda = 0$, the inequality is equality. Otherwise adding regularization theoretically increases the upper bound, though in practice will be key for stability.

Solving gives $\boldsymbol{\beta}^{(i)} = \alpha_i (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i$, and so

$$\begin{aligned} &{\boldsymbol{\beta}^{(i)}}^\top (\mathbf{K}_S + \lambda \mathbf{I}) \boldsymbol{\beta} - 2\alpha_i {\boldsymbol{\beta}^{(i)}}^\top \mathbf{k}_i + \alpha_i^2 k(\mathbf{x}_i, \mathbf{x}_i) \\ &= \alpha_i^2 \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i - 2\alpha_i^2 \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i \\ &\qquad + \alpha_i^2 k(\mathbf{x}_i, \mathbf{x}_i) \\ &= \alpha_i^2 k(\mathbf{x}_i, \mathbf{x}_i) - \alpha_i^2 \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i \end{aligned}$$

We can now simplify the above upper bound

$$(2) \leq \min_{S\subset\mathcal{X}} \sum_{i=1}^n \left(\alpha_i^2 k(\mathbf{x}_i, \mathbf{x}_i) - \alpha_i^2 \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i \right)$$

and obtain equivalent optimization

$$\operatorname*{argmax}_{S\subset\mathcal{X}} \sum_{i=1}^n \alpha_i^2 \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i.$$

This criteria closely resembles the coherence criterion (Engel et al., 2004). The key idea for the coherence criterion is to add a prototype $\mathbf{x}_i$, to kernel matrix $\mathbf{K}_S$ if $1 - \mathbf{k}_i \mathbf{K}_S^{-1} \mathbf{k}_i \leq \nu$ for some threshold parameter ν. The coherence criterion,

$$\operatorname*{argmax}_{S\subset\mathcal{X}} \sum_{i=1}^n \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i$$

therefore, can be seen as an upper bound on the distance the optimal function, with further relaxation because $\sum_{i=1}^n \alpha_i^2 \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i \leq \max\{\alpha_1^2, \dots, \alpha_n^2\} \sum_{i=1}^n \mathbf{k}_i^\top (\mathbf{K}_S + \lambda \mathbf{I})^{-1} \mathbf{k}_i$.

The relationship to another popular criterion—the log determinant— arises when we consider the extension to an infinite state space.

3.2. Criteria to select prototypes from an infinite set

The criterion above can be extended to an uncountably infinite observation space $\mathcal{X}$. For this setting, the optimal $f = \int_{\mathcal{X}} \omega(\mathbf{x})\phi(\mathbf{x})d\mathbf{x}$, for a function $\omega : \mathbb{R}^d \to \mathbb{R}$. Let $\mathbf{k}(\mathbf{x}, S) = [k(\mathbf{x}, \mathbf{z}_1), \ldots, k(\mathbf{x}, \mathbf{z}_b)]$ Then, using a similar analysis to above,

$$\min_{S\subset\mathcal{X}} \min_{\boldsymbol{\beta}\in\mathbb{R}^b} \|f - f_{S,\beta}\|^2 \leq \min_{S\subset\mathcal{X}} \int_{\mathcal{X}} \omega(\mathbf{x})^2 k(\mathbf{x},\mathbf{x})d\mathbf{x} - \int_{\mathcal{X}} \omega(\mathbf{x})^2 \mathbf{k}(\mathbf{x},S)^\top (\mathbf{K}_S + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x},S)d\mathbf{x}.$$

and so the resulting goal is to optimize

$$\operatorname*{argmax}_{S\subset\mathcal{X}} \int_{\mathcal{X}} \omega(\mathbf{x})^2 \mathbf{k}(\mathbf{x},S)^\top (\mathbf{K}_S + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x},S)d\mathbf{x}. \quad (3)$$

This provides a nice relation to the log determinant criterion, with normalized kernels[4]: $k(\mathbf{z},\mathbf{z}) = 1$. If $\mathbf{k}(\mathbf{x}, S)$ maps to a unique kernel vector $\mathbf{k} \in [0,1]^b$, and the function $\mathbf{k}(\cdot, S)$ also maps onto $[0,1]^b$, then for $\omega(\mathbf{x}) = 1$,

$$\begin{aligned}
&\int_{\mathcal{X}} \omega(\mathbf{x})^2 \mathbf{k}(\mathbf{x},S)^\top (\mathbf{K}_S + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x},S)d\mathbf{x} \\
&= \int \mathbf{k}^\top (\mathbf{K}_S + \lambda\mathbf{I})^{-1}\mathbf{k}\, d\mathbf{k} \\
&= \det(\mathbf{K}_S + \lambda\mathbf{I}).
\end{aligned}$$

In general, it is unlikely to have a bijection $\mathbf{k}(\cdot, S)$. More generally, we can obtain the above criterion by setting the coefficient function ω so that each possible kernel vector $\mathbf{k} \in \mathbb{R}^b$ has uniform weighting, or implicitly so the integration is uniformly over $\mathbf{k} \in [0,1]^b$. Because $\log$ is monotonically increasing, maximizing $\det(\mathbf{K}_S + \lambda\mathbf{I})$ with a fixed b is equivalent to maximizing $\log\det(\mathbf{K}_S + \lambda\mathbf{I})$.

This derivation of an upper bound on the distance to the optimal function provides new insights into the properties of the log-determinant, clarifies the connection between the coherence criterion and the log-determinant, and suggesting potential routes for providing criteria based on the prediction utility of a prototype. The choice of weighting ω to obtain the log-determinant removes all information about the utility of a prototype and essentially assumes a uniform distribution over the kernel vectors $\mathbf{k}$. For more general coefficient functions ω, let $\boldsymbol{\mu}(S) = \mathbb{E}[\mathbf{k}(\boldsymbol{X}, S)]$ and $\boldsymbol{\Sigma}(S) = \mathrm{Cov}(\mathbf{k}(\boldsymbol{X}, S))$, where the expectations are according to density ω^2/c for normalizer $c = \int_{\mathcal{X}} \omega(\mathbf{x})d\mathbf{x}$. By the quadratic expectations properties (Brookes, 2004)

$$(3) = \operatorname*{argmax}_{S\subset\mathcal{X}} \mathrm{tr}((\mathbf{K}_S + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}(S)) + \boldsymbol{\mu}(S)^\top (\mathbf{K}_S + \lambda\mathbf{I})^{-1}\boldsymbol{\mu}(S). \quad (4)$$

[4] A kernel can be normalized by $k(\mathbf{x},\mathbf{z})/\sqrt{k(\mathbf{z},\mathbf{z})k(\mathbf{x},\mathbf{x})}$.

This more general form in (4) enables prototypes to be more highly weighted based on the magnitude of values in ω. We focus in this work first on online prototype selection for the popular log-determinant, and leave further investigation into this more general criteria to future work. We nonetheless introduce the form here to better motivate the log-determinant, as well as demonstrate that the above analysis is amenable to a host of potential directions for more directed prototype selection.

4. Online submodular maximization

In this section, we introduce an OnlineGreedy algorithm for submodular maximization, to enable optimization of the prototype selection objective from an online stream of data. Current submodular maximization algorithms are designed for the streaming setting, which deals with incrementally processing large but fixed datasets. Consequently, the objectives are specified for a finite batch of observations and the algorithms can do multiple passes over the dataset. For the online setting, both of these conditions are restrictive. We show that, with a minor modification to StreamGreedy (Krause & Gomes, 2010), we can obtain a comparable approximation guarantee that applies to the online setting.

We would like to note that there is one streaming algorithm, called Sieve Streaming, designed to only do one pass of the data and avoid too many calls to the submodular function (Badanidiyuru et al., 2014); however, it requires keeping parallel solutions, which introduces significant complexity and which we found prohibitively expensive. In our experiments, we show it is significantly slower than our approach and found it typically maintained at least 500 parallel solutions. For this reason, we opt to extend the simpler StreamGreedy algorithm, and focus on efficient estimates of the submodular function, since we will require more calls to this function than Sieve Streaming.

Our goal is to solve the submodular maximization problem

$$\max_{S\subset\mathcal{X}:|S|\leq b} \; g(S) \quad (5)$$

where $\mathcal{X}$ is a general space of observations and g is a submodular function. The key modification is to enable $\mathcal{X}$ to be a large, infinite or even uncountable space. For such $\mathcal{X}$, we will be unable to see all observations, let alone make multiple passes. Instead, we will use a related notion to mixing time, where we see a cover of the space.

The greedy algorithm consists of greedily adding in a new prototype if it is an improvement on a previous prototype. The resulting greedy algorithm—given in Algorithm 1— is similar to StreamGreedy, and so we term it OnlineGreedy. The algorithm queries the submodular function on each set, with a previous prototype removed and the new observation added. To make this efficient, we will rely on us-

Algorithm 1 OnlineGreedy

Input: threshold parameter ϵ_t, where a prototype is only added if there is sufficient improvement
$S_0 \leftarrow \emptyset$
for $t = 1 : b$ **do** $S_t \leftarrow S_{t-1} \cup \{\mathbf{x}_t\}$
while interacting, $t = b+1, \dots$ **do**
  $\mathbf{z}' = \underset{\mathbf{z} \in S_{t-1}}{\operatorname{argmax}} \hat{g}(S_{t-1} \backslash \{\mathbf{z}\} \cup \{\mathbf{x}_t\})$
  $S_t \leftarrow S_{t-1} \backslash \{\mathbf{z}'\} \cup \{\mathbf{x}_t\}$
  if $\hat{g}(S_t) - \hat{g}(S_{t-1}) < \epsilon_t$ **then**
    $S_t \leftarrow S_{t-1}$

ing only an approximation to the submodular function g. We will provide a linear-time algorithm—in the number of prototypes— for querying replacement to all prototypes, as opposed to a naive solution which would be cubic in the number of prototypes. This will enable us to use this simple greedy approach, rather than more complex streaming submodular maximization approaches that attempt to reduce the number of calls to the submodular function.

We bound approximation error, relative to the optimal solution. We extend an algorithm that uses multiple passes; our approach suggests more generally how algorithms from the streaming setting can be extended to an online setting. To focus the on this extension, we only consider submodular functions here; in Appendix B, we generalize the result to approximately submodular functions. Many set functions are approximately submodular, rather than submodular, but still enjoy similar approximation properties. The log-determinant is submodular, however, it is more likely that, for the variety of choices for ω, the generalized coherence criterion is only approximately submodular. For this reason, we provide this generalization to approximate submodularity, as it further justifies the design of (approximately) submodular criteria for prototype selection.

We compare our solution to the optimal solution

$$S^* = \underset{S \subset \mathcal{X} : |S| \leq b}{\operatorname{argmax}} \; g(S) = \{\mathbf{z}_1^*, \dots, \mathbf{z}_b^*\}.$$

Assumption 1 (Submodularity)**.** *g is monotone increasing and submodular.*

Assumption 2 (Approximation error)**.** *We have access to a set function $\hat{g}$ that approximates g: for some $\epsilon_f \geq 0$ for all $S \subset \mathcal{X}$, with $|S| \leq b$,*

$$|\hat{g}(S) - g(S)| \leq \epsilon_f$$

Assumption 3 (Submodular coverage time)**.** *For a fixed $\epsilon_r > 0$ and $\delta > 0$ there exists a $\rho \in \mathbb{N}$ such that for all $S \subset \mathcal{X}$ where $|S| \leq b$, with probability $1 - \delta$, for any $\mathbf{z}^* \in S^*$ an observation $\mathbf{x}$ is observed within ρ steps (starting from any point in $\mathcal{X}$) that is similar to $\mathbf{z}^*$ in that*

$$|g(S \cup \{\mathbf{x}\}) - g(S \cup \{\mathbf{z}^*\})| \leq \epsilon_r.$$

This final assumption characterizes that the environment is sufficiently mixing, to see a cover of the space. We introduce the term coverage, instead of cover time for finite-state, to indicate a relaxed notion of observing a covering of the space rather than observing all states.

For simplicity of the proof, we characterize the coverage time in terms of the submodular function. We show that the submodular function we consider—the log-determinant—satisfies this assumption, given a more intuitive assumption that instead requires that observations be similar according to the kernel. The statement and proof are in Appendix A.

Now we prove our main result.

Theorem 1. *Assume Assumptions 1-3 and that $g(S^*)$ is finite and $g(\emptyset) \geq 0$. Then, for $t > \rho g(S^*)/\epsilon_t$, all sets S_t chosen by OnlineGreedy using $\hat{g}$ satisfy, with probability $1 - \delta$,*

$$g(S_t) \geq \frac{1}{2} g(S^*) - \frac{b}{2}(\epsilon_r + 2\epsilon_f + \epsilon_t)$$

Proof: The proof follows closely to the proof of Krause & Gomes (2010, Theorem 4). The key difference is that we cannot do multiple passes through a fixed dataset, and instead use submodular coverage time.

Case 1: There have been $t \geq \rho g(S^*)/\epsilon_t$ iterations, and S_t has always changed within ρ iterations (i.e., there has never been ρ consecutive iterations where S_t remained the same). This mean that for each ρ iterations, $\hat{g}(S_t)$ must have been improved by at least ϵ_t, which is the minimum threshold for improvement. This means that over the t iterations, $\hat{g}(S_0)$ has improved by at least ϵ_t each ρ,

$$\hat{g}(S_0) + \epsilon_t t/\rho \geq \epsilon_t t/\rho = \hat{g}(S^*) \geq g(S^*) - \epsilon_f$$

The solution is within ϵ_f of $g(S^*)$, and we are done.

Case 2: At some time t, S_t was not changed for ρ iterations, i.e., $S_{t-\rho} = S_{t-\rho-1} = \dots S_t$. Order the prototypes in the set as $\mathbf{z}_i = \operatorname{argmax}_{\mathbf{z} \in S_t} g(\{\mathbf{z}_1, \dots, \mathbf{z}_{i-1}\} \cup \{\mathbf{z}\})$, with

$$\delta_i = g(\{\mathbf{z}_1, \dots, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \dots, \mathbf{z}_{i-1}\}).$$

By Lemma 3, $\delta_{i-1} \geq \delta_i$.

Because the point that was observed $\mathbf{r}_i$ that was closest to $\mathbf{z}_i^*$ was not added to S, we have the following inequalities

$$\begin{aligned} |\hat{g}(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{r}_i\})| &\leq \epsilon_f \\ \hat{g}(S \cup \{\mathbf{r}_i\}) - \hat{g}(S \cup \{\mathbf{z}_b\}) &\leq \epsilon_t \\ |g(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{z}_i^*\})| &\leq \epsilon_r \end{aligned}$$

where the last inequality is true for all $\mathbf{z}_i^*$ with probability $1 - \delta$. Using these inequalities, as shown more explicitly in the proof in the appendix, we get

$$g(S \cup \{\mathbf{z}_i^*\}) - g(S) \leq \delta_b + \epsilon_r + 2\epsilon_f + \epsilon_t$$

Algorithm 2 BlockGreedy: OnlineGreedy for Prototype Selection using a Block-Diagonal Approximation

r = block-size, with set of blocks $\mathcal{B}$, $S_0 \leftarrow \emptyset$
c, l book-keeping maps, with $(c(B), l(B)) = (\mathbf{z}, l)$ for $\mathbf{z}$ leading to smallest utility loss l if removed from block B.
$g_e \leftarrow 0$ is the incremental estimate of log-determinant
for $t = 1 : b$ **do**, $S_t \leftarrow S_{t-1} \cup \{\mathbf{x}_t\}$
while interacting, $t = b+1, \ldots$ **do**
  if added b new prototypes since last clustering **then**
    cluster S_t into $\lfloor b/r \rfloor$ blocks with k-means, initialize with previous clustering; update c, l, g_e
  BlockGreedy-Swap($\mathbf{x}_t$)

Algorithm 3 BlockGreedy-Swap($\mathbf{x}$)

$B_1 \leftarrow$ get-block($\mathbf{x}$) ▷ returns the nearest block to $\mathbf{x}$
$(\mathbf{z}_1, g_1) \leftarrow \underset{\mathbf{z} \in B_1}{\operatorname{argmax}}\, g(B_1 \backslash \{\mathbf{z}\} \cup \{\mathbf{x}\}) - g(B_1)$
if $g_e \neq 0$ and $\frac{g_1 - g_e}{g_e} < \epsilon_t$ **then**
  return with no update if low percentage improvement
$(B_2, g_2) \leftarrow \underset{B \in \mathcal{B} \backslash B_1}{\operatorname{argmax}}\, g(B_1 \cup \{\mathbf{x}\}) - l(B)$
if $g_1 < g_2$ **then** ▷ remove point from same block
  $B_1 \leftarrow B_1 \backslash \{\mathbf{z}_1\} \cup \{\mathbf{x}\}$
  update $c(B_1)$ ▷ using Appendix E.3
  $g_e \leftarrow g_e + g_1$
else ▷ remove point from a different block
  $B_1 \leftarrow B_1 \cup \{\mathbf{x}\}$
  $B_2 \leftarrow B_2 \setminus \{c(B_2)\}$
  update $c(B_1)$, $c(B_2)$ ▷ using Appendix E.3
  $g_e \leftarrow g_e + g_2$

By the definition of submodularity, $g(S \cup S^*) - g(S) \leq \sum_{i=1}^{b} g(S \cup \{\mathbf{z}_i^*\}) - g(S))$.

Putting this all together, with probability $1 - \delta$,

$$
\begin{aligned}
g(S^*) &\leq g(S \cup S^*) \\
&\leq g(S) + \sum_{i=1}^{b} g(S \cup \{\mathbf{z}_i^*\}) - g(S)) \\
&\leq g(S) + \sum_{i=1}^{b} (\delta_b + \epsilon_r + 2\epsilon_f + \epsilon_t) \\
&\leq g(S) + \left(\sum_{i=1}^{b} \delta_i \right) + b(\epsilon_r + 2\epsilon_f + \epsilon_t) \\
&= g(S) + \sum_{i=1}^{b} g(\{s_1, \ldots, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}\}) \\
&\qquad + b(\epsilon_r + 2\epsilon_f + \epsilon_t) \\
&\leq g(S) + g(\{\mathbf{z}_1, \ldots, \mathbf{z}_b\}) + b(\epsilon_r + 2\epsilon_f + \epsilon_t) \\
&\leq 2g(S_t) + b(\epsilon_r + 2\epsilon_f + \epsilon_t)
\end{aligned}
$$

where the last inequality uses $g(S) \leq g(S_t)$ which follows from monotonicity. ■

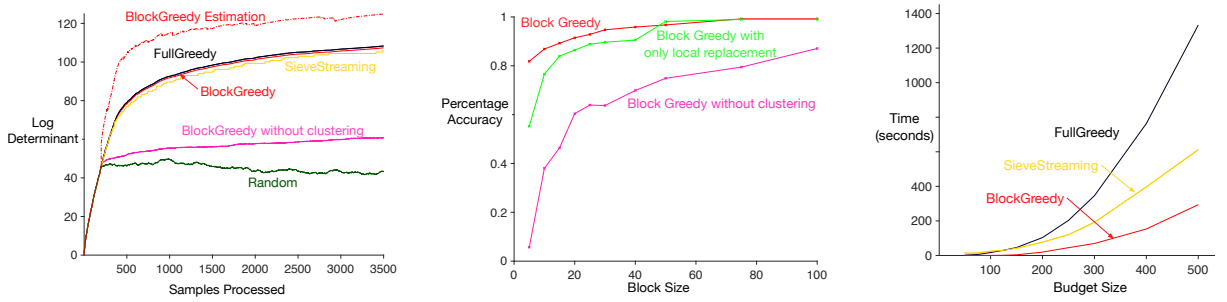
5. Block-diagonal approximation for efficient computation of the submodular function

The computation of the submodular function g is the critical bottleneck in OnlineGreedy and other incremental submodular maximization techniques. In this section, we propose a time and memory efficient greedy approach to computing a submodular function on $\mathbf{K}_S$, enabling each step of OnlineGreedy to cost $O(db)$, where d is the feature dimension and b is the budget size. The key insight is to take advantage of the block-diagonal structure of the kernel matrix, particularly due to the fact that the greedy algorithm intentionally selects diverse prototypes. Consequently, we can approximately cluster prototypes into small groups of size r, and perform updates on only these blocks.

Approximations to the kernel matrix have been extensively explored, but towards the aim of highly accurate approximations for use within prediction. These methods include low-rank approximations (Bach & Jordan, 2005), Nystrom methods (Drineas & Mahoney, 2005; Gittens & Mahoney, 2013) and a block-diagonal method for dense kernel matrices, focused on storage efficiency (Si et al., 2014). Because these approximations are used for prediction and because they are designed for a fixed batch of data and so do not take advantage of incrementally updating values, they are not sufficiently efficient for use on each step, and require at least $O(b^2)$ computation. For OnlineGreedy, however, we only need a more coarse approximation to $\mathbf{K}_S$ to enable effective prototype selection. By taking advantage of this fact, saving computation with incremental updating and using the fact that our kernel matrix is not dense—making it likely that many off-diagonal elements are near zero— we can reduce storage and computation to linear in b.

The key steps in the algorithm are to maintain a clustering of prototypes, compute all pairwise swaps between prototypes within a block—which is much more efficient than pairwise swaps between all prototypes— and finally perform a single swap between two blocks. The computational complexity of Algorithm 2 on each step is $O(bd + r^3)$ for block size r (see Appendix F for an in-depth explanation). We assume that, with a block-diagonal $\mathbf{K}_S$ with blocks $\mathcal{B}$, the submodular function separates into $g(S) = \sum_{B \in \mathcal{B}} g(B)$. For both the log-determinant and the trace of the inverse of $\mathbf{K}_S$, this is the case because the inverse of a block-diagonal matrix corresponds to a block-diagonal matrix of the inverses of these blocks. Therefore, $\log\det(\mathbf{K}_S) = \sum_{B \in \mathcal{B}} \log\det(\mathbf{K}_B)$.

We use this property to avoid all pairwise comparisons. If $\mathbf{x}$ is added to S, it gets clustered into its nearest block, based



(a) $\log\det$ of $\mathbf{K}$ (b) Estimate Accuracy, with $b = 200$ (c) Runtime with increasing b

Figure 1. Performance of BlockGreedy in Telemonitoring. Figure **(a)** shows the true log determinant of $\mathbf{K}$ for the prototypes selected by each algorithm. Our algorithm, BlockGreedy, achieves almost the same performance as FullGreedy, which uses no approximation to $\mathbf{K}$ to compute the log-determinant. Figure **(b)** shows the accuracy of the log determinant estimate as the block size increases. We can see that clustering is key, and that for smaller block sizes, comparing between blocks is key. Figure **(c)** shows the runtime of the main prototype selection competitors, FullGreedy and SieveStreaming, versus BlockGreedy with block size $r = 10$.

on distance to the mean of that cluster. To compute the log-determinant for the new S, we simply need to recompute the log-determinant for the modified block, as the log-determinant for the remaining blocks is unchanged. Therefore, if $\mathbf{K}_S$ really is block-diagonal, computing all pairwise swaps with $\mathbf{x}$ is equivalent to first computing the least useful point $\mathbf{z}$ in the closest cluster to $\mathbf{x}$, and then determining if $g(S \cup \{\mathbf{x}\})$ would be least reduced by removing $\mathbf{z}$ or removing the least useful prototype from another cluster. With some book-keeping, we maintain the least-useful prototype for each cluster, to avoid recomputing it each step.

6. Experiments

We empirically illustrate the accuracy and efficiency of our proposed method as compared to OnlineGreedy with no approximation to the submodular function (which we call Full Greedy), Sieve Streaming, and various naive versions of our algorithm. We also show this method can achieve reasonable regression accuracy as compared with KRLS (Engel et al., 2004). For these experiments we use four well known datasets: Boston Housing (Lichman, 2015), Parkinson's Telemonitoring (Tsanas et al., 2010), Sante Fe A (Weigend, 1994) and Census 1990 (Lichman, 2015). Further details about each dataset are in Appendix C. We use a Gaussian kernel for the first three datasets, and a Hamming distance kernel for Census, which has categorical features. To investigate the effect of the block-diagonal approximation, we select the log-determinant as the criterion, which is an instance of our criterion, and set $\lambda = 1$.

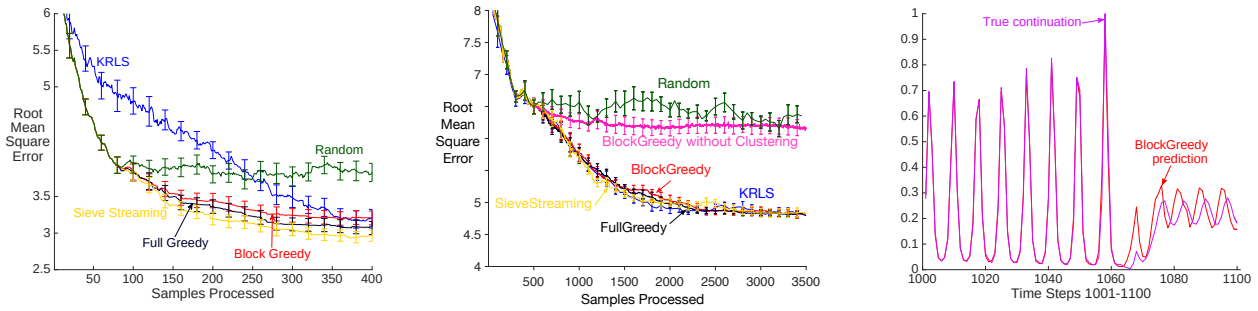
Quality of the log-determinant approximation.

We first investigate the quality of prototypes selection and their runtimes, depicted in Figure 1. We compare our algorithm with the FullGreedy, SieveStreaming and a random prototype selection baseline. We also use variants of our algorithm including without clustering—naively dividing prototypes into equal-sized blocks—and one where we only consider replacement in the closest block. We include these variants to indicate the importance of clustering and of searching between blocks as well within blocks, in BlockGreedy. For all experiments on maximization quality, we use percentage gain with a threshold of $\epsilon_t = 0.001$.

We plot the log determinant with increasing samples, in Figure 1(a). Experiments on the other datasets are included in Appendix C. BlockGreedy maximizes the submodular function within 1% of the FullGreedy method. Though BlockGreedy achieves nearly as high a log determinant value, we can see that its approximation of the log determinant is an overestimate for this small block size, $r = 5$.

Our next experiment, therefore, focuses on the estimate accuracy of BlockGreedy with increasing block size, in Figure 1(b). The accuracy is computed by $1 - |\frac{g_{actual} - g_{estimate}}{g_{actual}}|$. We can see our algorithm, BlockGreedy performs much better as compared to the other variants, ranging in accuracy from 0.82 to 0.99. This suggests that one can choose reasonably small block sizes, without incurring a significant penalty in maximizing the log determinant. In Figure 1(a), the estimate is inaccurate by about 20%, but follows the same trend of the full log determinant and picks similar prototypes to those chosen by FullGreedy.

The runtime of our algorithm should be much less than that of FullGreedy, and memory overhead much less than SieveStreaming. In Figure 1(c), we can see our method scales much better than FullGreedy and even has gains in speed over SieveStreaming. Though not shown, the number of sieves generated by SieveStreaming is large, in many instances well over 600, introducing a significant amount of



(a) Boston housing (b) Telemonitoring (c) Santa Fe Data Set A

Figure 2. Figure **(a)** is the learning curve on Bostondata, averaged over 50 runs, with $\eta = 0.01$. On average, KRLS uses 116.46 prototypes. Figure **(b)** is the learning curve on the Telemonitoring data set, over 50 runs, with $b = 500$ and $\eta = 0.001$. Figure **(c)** plots the predicted values of our algorithm and true values. The regularizer $\eta = 0.001$, and the utility threshold is $\epsilon_t = 0.0001$.

overhead. Overall, by taking advantage of the block structure of the kernel matrix, our algorithm obtains significant runtime and memory improvements, while also producing a highly accurate estimate of the log determinant.

Learning performance for regression problems.

While the maximization of the submodular function is useful in creating a diverse collection of prototypes, ultimately we would like to use these representations for prediction. In Figure 2, we show the effectiveness of solving $(\mathbf{K}_S + \eta\mathbf{I})\mathbf{w} = \mathbf{y}$ for the three regression datasets, by using our algorithm to select prototypes for $\mathbf{K}_S$. For all regression experiments, we use a threshold of $\epsilon_t = 0.01$ unless otherwise specified.

For Boston housing data, in figure 2(a), we see that BlockGreedy can perform almost as well as FullGreedy and SieveStreaming, and outperforms KRLS at early learning and finally converges to almost same performance. We set the parameters for KRLS using the same parameter settings for this dataset as in their paper (Engel et al., 2004). For our algorithms we set the budget size to $b = 80$, which is smaller than what KRLS used, and chose a block size of 4. We also have lower learning variance than KRLS, likely because we use explicit regularization, whereas KRLS uses its prototype selection mechanism for regularization.

On the Telemonitoring dataset, the competitive algorithms all perform equally well, reaching a RMSE of approximately 4.797. BlockGreedy, however, uses significantly less computation for selecting prototypes. We used a budget of $b = 500$, and a block size of $r = 25$; a block size of $r = 5$ for this many prototypes impacted the log determinant estimation enough that it was only able to reach a RMSE of about 5.2. With the larger block size, BlockGreedy obtained a log determinant value within 0.5% of FullGreedy.

On the benchmark time series data set *Santa Fe Data Set A*, we train on the first 1000 time steps in the series and predict the next 100 steps, calculating the normalized MSE (NMSE), as stated in the original competition. We set the width parameter and budget size to that used with KRLS after one iteration on the training set. The NMSE of our algorithm and KRLS were 0.0434 and 0.026 respectively. While our method performs worse, note that KRLS actually runs on 6×1000 samples according to its description (Engel et al., 2004), but with 1000 samples it performs worse with a NMSE of 0.0661. We demonstrate the 100-step forecast with BlockGreedy, in Figure 2(c); we include forecast plots for the other algorithms in Figure 4, Appendix C.4.

7. Conclusion

We developed a memory and computation efficient incremental algorithm, called BlockGreedy, to select centers for kernel representations in a continual learning setting. We derived a criterion for prototype selection, and showed that the log-determinant is an instance of this criterion. We extended results from streaming submodular maximization, to obtain an approximation ratio for OnlineGreedy. We then derived the efficient variant, BlockGreedy, to take advantage of the block-diagonal structure of the kernel matrix, which enables separability of the criteria and faster local computations. We demonstrated that, by taking advantage of this structure, BlockGreedy can significantly reduce computation without incurring much penalty in maximizing the log-determinant and maintaining competitive prediction performance. Our goal within continual learning was to provide a principled, near-linear time algorithm for prototype selection, in terms of the number of prototypes. We believe that BlockGreedy provides one of the first such algorithms, and is an important step towards effective kernel representations for continual learning settings, like online learning and reinforcement learning.

Acknowledgements

This research was supported in part by NSF CCF-1525024, IIS-1633215 and the Precision Health Initiative at Indiana University. We would also like to thank Inhak Hwang for helpful discussions.

References

Argyriou, A. and Dinuzzo, F. A Unifying View of Representer Theorems. In *International Conference on Machine Learning*, 2014.

Bach, F. R. and Jordan, M. I. Predictive low-rank decomposition for kernel methods. In *International Conference on Machine Learning*, 2005.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: massive data summarization on the fly. *Conference on Knowledge Discovery and Data Mining*, 2014.

Brookes, M. *Matrix reference manual*. Imperial College London, 2004.

Chen, B., Zhao, S., Zhu, P., and Principe, J. C. Quantized Kernel Recursive Least Squares Algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

Cheng, C.-A. and Boots, B. Incremental Variational Sparse Gaussian Process Regression. *Advances in Neural Information Processing Systems*, 2016.

Cheng, L., Vishwanathan, S. V. N., Schuurmans, D., Wang, S., and Caelli, S. W. Implicit Online Learning with Kernels. In *Advances in Neural Information Processing Systems*, 2007.

Csató, L. and Opper, M. Sparse On-Line Gaussian Processes. *dx.doi.org*, 2006.

Dai, B., Xie, B., He, N., Liang, Y., Raj, A., Balcan, M.-F. F., and Song, L. Scalable Kernel Methods via Doubly Stochastic Gradients. *Advances in Neural Information Processing Systems*, 2014.

Das, A. and Kempe, D. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection. *arXiv.org*, 2011.

Drineas, P. and Mahoney, M. W. On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *Journal of Machine Learning Research*, 2005.

Engel, Y., Mannor, S., and Meir, R. The kernel recursive least-squares algorithm. *IEEE Transactions on Signal Processing*, 2004.

Gittens, A. and Mahoney, M. W. Revisiting the Nystrom method for improved large-scale machine learning. In *International Conference on Machine Learning*, 2013.

Guha, S., Meyerson, A., Mishra, N., Motwani, R., and O'Callaghan, L. Clustering Data Streams: Theory and Practice. *IEEE Transaction on Knowledge and Data Engineering*, 2003.

Huang, P. S., Avron, H., and Sainath, T. N. Kernel methods match deep neural networks on timit. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

Kivinen, J., Smola, A., and Williamson, R. C. Online learning with kernels. *IEEE Transactions on Signal Processing*, 2010.

Krause, A. and Gomes, R. G. Budgeted nonparametric learning from data streams. In *International Conference on Machine Learning*, 2010.

Krause, A., McMahon, H. B., Guestrin, C., and Gupta, A. Robust Submodular Observation Selection. *Journal of Machine Learning Research*, 2008a.

Krause, A., Singh, A. P., and Guestrin, C. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research*, 2008b.

Li, C., Jegelka, S., and Sra, S. Fast DPP Sampling for Nystrom with Application to Kernel Methods. In *International Conference on Machine Learning*, 2016.

Lichman, M. *UCI machine learning repository*. URL http://archive. ics. uci. edu/ml, 2015.

Lu, Z., May, A., Liu, K., Garakani, A. B., Guo, D., Bellet, A., Fan, L., Collins, M., Kingsbury, B., Picheny, M., and Sha, F. How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets. *CoRR abs/1202.6504*, 2014.

Mairal, J., Koniusz, P., Harchaoui, Z., and Schmid, C. Convolutional Kernel Networks. *NIPS*, 2014.

Matic, I. Inequalities with determinants of perturbed positive matrices. *Linear Algebra and its Applications*, 2014.

Park, J. and Sandberg, I. Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation*, 1991.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

Richard, C., Bermudez, J. C. M., and Honeine, P. Online Prediction of Time Series Data With Kernels. *IEEE Transactions on Signal Processing*, 2009.

Rudi, A., Camoriano, R., and Rosasco, L. Less is More: Nyström Computational Regularization. *Advances in Neural Information Processing Systems*, 2015.

Schraudolph, N. N., Smola, A. J., and Joachims, T. Step size adaptation in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, 2006.

Seeger, M. Greedy Forward Selection in the Informative Vector Machine. 2004.

Seeger, M., Williams, C., and Lawrence, N. Fast Forward Selection to Speed Up Sparse Gaussian Process Regression. *Artificial Intelligence and Statistics*, 2003.

Si, S., Hsieh, C.-J., and Dhillon, I. Memory efficient kernel approximation. In *International Conference on Machine Learning*, 2014.

Tsanas, A., Little, M. A., and McSharry, P. E. Accurate Telemonitoring of Parkinson's Disease Progression by Noninvasive Speech Tests. *IEEE transactions on Biomedical Engineering*, 2010.

Van Vaerenbergh, S. and Santamaria, I. A comparative study of kernel adaptive filtering algorithms. In *Digital Signal Processing and Signal Processing Education Meeting*, 2013.

Van Vaerenbergh, S., Santamaría, I., Liu, W., and Principe, J. C. Fixed-budget kernel recursive least-squares. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010.

Weigend, A. S. Time Series Prediction: Forecasting the Future and Understanding the Past. *Santa Fe Institute Studies in the Sciences of Complexity*, 1994.

A. Coverage time for the log-determinant

The more general coverage time condition assumes similarity between points, based on the submodular function itself. However, more intuitively, we would like the condition to be related to the similarity measure. In this section, we show how we can obtain Assumption 3 for the log-determinant, but using instead coverage in terms of the kernel rather than the submodular function itself. In the below, we show that, for an ϵ related to ϵ_r, when $||\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)|| \leq \sqrt{2\epsilon}, \epsilon \geq 0$ for two points $\mathbf{x}_1, \mathbf{x}_2$— which for a normalized kernel means $k(\mathbf{x}_1, \mathbf{x}_2) \geq 1 - \epsilon$— then we get that $|g(S \cup \{\mathbf{x}_1\}) - g(S \cup \{\mathbf{x}_2\})| \leq \epsilon_r$.

Lemma 2. *Assume k is a continuous Mercer kernel. For $g(S) = \log\det(\lambda\mathbf{I} + \mathbf{K}_S)$ and $\epsilon_r = \log\frac{\lambda+\sqrt{8\epsilon\lambda}+2\epsilon}{\lambda}$, we have that if $||\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)|| \leq \sqrt{2\epsilon}$ then*

$$|g(S \cup \{\mathbf{x}_1\}) - g(S \cup \{\mathbf{x}_2\})| \leq \epsilon_r.$$

Proof: Let $\mathbf{K}$ be the kernel matrix without $\mathbf{x}_1$ and without $\mathbf{x}_2$, let k_{ii} denote $k(\mathbf{x}_i, \mathbf{x}_i)$ for short. Then $\mathbf{K}_j = [\mathbf{K}\ \mathbf{k}_j^\top; \mathbf{k}_j\ k_{jj}]$ for $\mathbf{k}_j = [k(\mathbf{z}_1, \mathbf{x}_j), \ldots, k(\mathbf{z}_{b-1}, \mathbf{x}_j)]$. The determinant of $\lambda\mathbf{I} + \mathbf{K}_j$, by (Matic, 2014, Theorem 2.4), is

$$\begin{aligned}
&\det(\lambda\mathbf{I} + \mathbf{K}_j) \\
&= \det(\lambda\mathbf{I} + \mathbf{K})\det(k_{jj} + \lambda - \mathbf{k}_j^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_j) \\
&= \det(\lambda\mathbf{I} + \mathbf{K})(k_{jj} + \lambda - \mathbf{k}_j^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_j)
\end{aligned}$$

The ratio between the two determinants then reduces to

$$\frac{\det(\lambda\mathbf{I} + \mathbf{K}_1)}{\det(\lambda\mathbf{I} + \mathbf{K}_2)} = \frac{\lambda + k_{11} - \mathbf{k}_1^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_1}{\lambda + k_{22} - \mathbf{k}_2^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_2} \leq \frac{\lambda + k_{11}}{\lambda}$$

with the lower bound on the denominator resulting from $k_{22} - \mathbf{k}_2^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_2 \geq 0$ because:

$$\begin{aligned}
&k_{jj} - \mathbf{k}_j^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_j \\
&\quad = \min_{\mathbf{a}} \left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_j) \right\|^2 + \lambda\|\mathbf{a}\|_2^2
\end{aligned}$$

One should note that this is a loose upper bound without restriction on ϵ. To get a tighter bound, we start from considering distance bound $\|\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)\| \leq \sqrt{2\epsilon}$. We will see that they have a similar error in being spanned by the kernels in $\mathbf{K}$. Let $\mathbf{a}$ be the optimal solution for approximating $\phi(\mathbf{x}_2)$ in above minimization problem, then:

$$\begin{aligned}
&\left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_1) \right\| \\
&= \left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_1) + \phi(\mathbf{x}_2) - \phi(\mathbf{x}_2) \right\| \\
&\leq \left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_2) \right\| + \|\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)\| \\
&\leq \left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_2) \right\| + \sqrt{2\epsilon}.
\end{aligned}$$

Let $b \doteq \lambda + k_{22} - \mathbf{k}_2^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_2$, which means $b = \left\|\sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_2)\right\|^2 + \lambda\|\mathbf{a}\|_2^2 + \lambda$. Then

$$\begin{aligned}
&\left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_1) \right\|^2 + \lambda\|\mathbf{a}\|_2^2 + \lambda \qquad (6) \\
&\leq \left(\left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_2) \right\| + \sqrt{2\epsilon} \right)^2 + \lambda\|\mathbf{a}\|_2^2 + \lambda \\
&\leq b + 2\left\| \sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_2) \right\| \sqrt{2\epsilon} + 2\epsilon \\
&\leq b + \sqrt{8b\epsilon} + 2\epsilon
\end{aligned}$$

where the last step follows because $\|\sum_{\mathbf{z}_i \in S} a_i\phi(\mathbf{z}_i) - \phi(\mathbf{x}_2)\|^2 \leq b$. Because $\mathbf{a}$ is not optimal for $\mathbf{x}_1$,

$$\begin{aligned}
\lambda + k_{11} - \mathbf{k}_1^\top(\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{k}_1 &\leq (6) \\
&\leq b + \sqrt{8\epsilon b} + 2\epsilon
\end{aligned}$$

Because $b \geq \lambda$

$$\begin{aligned}
&\log\det(\lambda\mathbf{I} + \mathbf{K}_1) - \log\det(\lambda\mathbf{I} + \mathbf{K}_2) \\
&\qquad = \log\frac{\det(\lambda\mathbf{I} + \mathbf{K}_1)}{\det(\lambda\mathbf{I} + \mathbf{K}_2)} \\
&\qquad \leq \log\frac{b + \sqrt{8\epsilon b} + 2\epsilon}{b} \\
&\qquad \leq \log\frac{\lambda + \sqrt{8\epsilon\lambda} + 2\epsilon}{\lambda}
\end{aligned}$$

Note that this is a tighter lower bound than $\log\frac{\lambda+k_{11}}{\lambda}$ when $\epsilon \leq \sqrt{\lambda + k_{11}} - \sqrt{\lambda}$. We can swap the order of $\mathbf{x}_1$ and $\mathbf{x}_2$, to get a bound on the absolute difference, completing the proof. ■

B. Generalization to approximate submodularity

In this section, we generalize our approximate ratio results for OnlineGreedy to functions that are approximately submodularity.

Assumption 4 (Approximate submodularity)**.** *g is monotone increasing and approximately submodular with submodularity ratio $\gamma \geq 0$.*

An approximately submodular function (Das & Kempe, 2011) satisfies the looser requirement that

$$\gamma(g(S \cup A) - g(S)) \leq \sum_{a \in A} g(S \cup \{a\}) - g(S)).$$

For $\gamma \geq 1$, the function is submodular. An example of an approximately submodular function is feature selection using correlation between features and residuals (Das & Kempe, 2011). In some cases, adding features does not satisfy a diminishing returns property. This is due to interactions between pairs of features, where using both features can significantly improve prediction performance even though individually they are insufficient. For many features and sets, the correlation function is submodular; however, in the worst case, we can only guarantee approximate submodularity. Our generalized coherence criterion is guaranteed to be approximately submodular, for a sufficiently small γ. However, we anticipate a γ near one, since for feature selection, Das & Kempe (2011) demonstrated empirically that γ was typically above 0.8. The generalized coherence criterion is similar to feature selection, in that it selects prototypes to better span $\phi(\mathbf{x})$ for all $\mathbf{x}$, according to a weighted criterion.

Assumption 5 (Probabilistic approximate submodularity)**.** *The stream of data satisfies the probabilistic submodular property: with probability $1-\delta$, for the chosen set of prototypes $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_b\}$, with $\mathbf{z}_b = \operatorname{argmin}_{\mathbf{z} \in S} g(S) - g(S \backslash \{\mathbf{z}\})$ the single least important element, there exists an ordering $\{\mathbf{z}_1, \ldots, \mathbf{z}_{k-1}\}$ with $\delta_i = g(\{\mathbf{z}_1, \ldots, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}\})$ such that the average difference is approximately submodular: $\frac{1}{b}\sum_{i=1}^{b} \delta_i \geq \gamma \delta_b$.*

For a submodular function, this is automatically true, as we show in the next lemma

Lemma 3. *Given a set S and submodular g, let $\mathbf{z}_i = \operatorname{argmax}_{s \in S_t} g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}\} \cup \{s\})$ and $\delta_i = g(\{\mathbf{z}_1, \ldots, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}\})$. Then $\delta_{i-1} \geq \delta_i$.*

Proof:

$$g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}\}) + g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_i\}) \geq g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}, \mathbf{z}_i\}) + g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\})$$

we get that

$$\begin{aligned}\delta_{i-1} &= g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\}) \\ &\geq g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_i\}) \\ &\geq g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}\}) \\ &= \delta_i.\end{aligned}$$

The last inequality follows from the chosen ordering, giving $g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_i\}) \leq g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-2}\} \cup \{\mathbf{z}_{i-1}\})$. ■

Otherwise, for approximately submodular functions, in the worst case, there could be an adversarial ordering that results in $\delta_{i-1} = \gamma \delta_i$. Consequently $\delta_i = \gamma^{k-i}\delta_b$, which would lead to a poor approximation ratio for OnlineGreedy. This worst case behavior, however, is highly unlikely. For example, in feature selection, it is highly unlikely that all features interact to cause this worst case behavior. For our streaming setting, it is highly unlikely that we will see an adversarial sequence of observations that causes our criteria to have $\delta_i = \gamma^{k-i}\delta_b$.

Theorem 4. *Assume Assumptions 1-4 and that $g(S^*)$ is finite and $g(\emptyset) \geq 0$. Then, for $t > \rho g(S^*)/\epsilon_t$, all sets S_t chosen by OnlineGreedy using $\hat{g}$ satisfy with probability $1 - 2\delta$*

$$g(S_t) \geq \frac{1}{2} g(S^*) - \frac{b}{2\gamma}(\epsilon_r + 2\epsilon_f + \epsilon_t)$$

Proof: The proof follows closely to the proof of Krause & Gomes (2010, Theorem 4). The key difference is that we cannot do multiple passes through a fixed dataset, but instead simulate this by the fact that the sampling mixes sufficiently to see observations close to the set of optimal prototypes. Further, we relax the requirement of submodularity to approximate submodularity and do not require g to be bounded for all subsets, but rather only use the upper bound of the optimal solution.

Case 1: There have been $t \geq \rho g(S^*)/\epsilon_t$ iterations, and S_t has always changed within ρ iterations (i.e., there has never been ρ consecutive iterations where S_t remained the same). This mean that for each ρ iterations, $\hat{g}(S_t)$ must have been improved by at least ϵ_t, which is the minimum threshold for improvement. This means that over the t iterations, $\hat{g}(S_0)$ has improved by at least ϵ_t each ρ,

$$\begin{aligned}\hat{g}(S_0) + \epsilon_t t/\rho &\geq \epsilon_t t/\rho \\ &= \hat{g}(S^*) \\ &\geq g(S^*) - \epsilon_f\end{aligned}$$

In this case, the solution is within ϵ_f of $g(S^*)$, and we are done.

Case 2: At some time t, S_t was not changed for ρ iterations, i.e., $S_{t-\rho} = S_{t-\rho-1} = \ldots S_t$. Let $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_{b-1}\}$ and $\delta_i = g(\{\mathbf{z}_1, \ldots, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}\})$, with ordering given by Assumption 5. Because the point that was observed $\mathbf{r}_i$ that was closest to $\mathbf{z}_i^*$ was not added to S, we

have the following inequalities

$$\begin{aligned}|\hat{g}(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{r}_i\})| &\leq \epsilon_f \\ \hat{g}(S \cup \{\mathbf{r}_i\}) - \hat{g}(S \cup \{\mathbf{z}_b\}) &\leq \epsilon_t \\ |g(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{z}_i^*\})| &\leq \epsilon_r \\ g(S \cup \{\mathbf{z}_b\}) - g(S) &= \delta_b.\end{aligned}$$

where the second last inequality is true for all $\mathbf{z}_i^*$ with probability $1-\delta$. Therefore,

$$\begin{aligned}&g(S \cup \{\mathbf{z}_i^*\}) - g(S) \\ &= g(S \cup \{\mathbf{z}_i^*\}) - g(S \cup \{\mathbf{z}_b\}) + g(S \cup \{\mathbf{z}_b\}) - g(S) \\ &= g(S \cup \{\mathbf{z}_i^*\}) - g(S \cup \{\mathbf{r}_i\}) \\ &\qquad + g(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{z}_b\}) + \delta_b \\ &\leq \epsilon_r + \delta_b + g(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{z}_b\}) \\ &= \epsilon_r + \delta_b + g(S \cup \{\mathbf{r}_i\}) - \hat{g}(S \cup \{\mathbf{r}_i\}) \\ &\qquad + \hat{g}(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{z}_b\}) \\ &\leq \epsilon_r + \delta_b + \epsilon_f + \hat{g}(S \cup \{\mathbf{r}_i\}) - g(S \cup \{\mathbf{z}_b\}) \\ &= \epsilon_r + \delta_b + \epsilon_f + \hat{g}(S \cup \{\mathbf{r}_i\}) - \hat{g}(S \cup \{\mathbf{z}_b\}) \\ &\qquad + \hat{g}(S \cup \{\mathbf{z}_b\}) - g(S \cup \{\mathbf{z}_b\}) \\ &\leq \epsilon_r + \delta_b + \epsilon_f + \epsilon_t + \epsilon_f \\ &= \delta_b + \epsilon_r + 2\epsilon_f + \epsilon_t\end{aligned}$$

By definition of approximate submodularity,

$$\gamma(g(S \cup S^*) - g(S)) \leq \sum_{i=1}^{b} g(S \cup \{s_i^*\}) - g(S)).$$

Putting this all together, with probability $1-\delta$ additionally from Assumption 5, we get with probability $1-2\delta$

$$\begin{aligned}g(S^*) &\leq g(S \cup S^*) \\ &\leq g(S) + \frac{1}{\gamma}\sum_{i=1}^{b} g(S \cup \{\mathbf{z}_i^*\}) - g(S)) \\ &\leq g(S) + \frac{1}{\gamma}\sum_{i=1}^{b}(\delta_b + \epsilon_r + 2\epsilon_f + \epsilon_t) \\ &\leq g(S) + \frac{1}{\gamma^2}\left(\sum_{i=1}^{b}\delta_i\right) + \frac{b}{\gamma}(\epsilon_r + 2\epsilon_f + \epsilon_t) \\ &= g(S) + \frac{1}{\gamma^2}\sum_{i=1}^{b} g(\{\mathbf{z}_1, \ldots, \mathbf{z}_i\}) - g(\{\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}\}) \\ &\qquad + \frac{b}{\gamma}(\epsilon_r + 2\epsilon_f + \epsilon_t) \\ &\leq g(S) + g(\{\mathbf{z}_1, \ldots, \mathbf{z}_b\}) + \frac{b}{\gamma}(\epsilon_r + 2\epsilon_f + \epsilon_t) \\ &\leq 2g(S_t) + \frac{b}{\gamma}(\epsilon_r + 2\epsilon_f + \epsilon_t)\end{aligned}$$

where the last inequality follows from monotonicity, giving $g(S) \leq g(S_t)$. ■

C. Experimental Details

C.1. Details on Boston Housing Data Set

We conduct experiments on a frequently cited and easily understandable real world data set — *Boston housing* (Lichman, 2015). The data set includes 506 samples, each of which has 13 features. Our purpose is to predict the median value of owner-occupied homes in $\$1000$'s for various Boston neighborhoods. We normalize all the attributes to a range of $[0, 1]$. Our experiments follow the parameter settings specified in the KRLS paper (Engel et al., 2004): the RBF width as $\sigma = 0.295$, and ALD condition threshold $\nu = 0.001$. We also choose reasonably good parameters for our algorithms instead of fully optimize them. The budget is $b = 80$, the number of blocks $c = 20$, the regularization parameter for computing log determinant is $\lambda = 1.0$, and the regularization parameter for solving linear system is $\eta = 0.01$. We randomly shuffle the data set with 400 training samples and 106 testing samples to get learning curve averaged over 50 runs. The prediction error is calculated with the root mean squared error and can be found in figure 2(a). The utility change as number of samples processed is showed in figure 3(a) and figure 3(b).

C.2. Details on Parkinson's Telemonitoring Dataset

The Parkinsons Telemonitoring dataset (Tsanas et al., 2010) is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease. The task is to predict the clinician's Parkinson's disease symptom score on the UPDRS scale. Two out of the 26 attributes are targets: motor and total UPDRS scores. We choose motor UPDRS as our prediction target. After normalizing the data to range $[0, 1]$, we randomly shuffle the set and use the first 3500 sample as the training set and the remaining 2375 as the test set. We incrementally process the training samples and record RMSE of each algorithm at each step to obtain the learning curve, the true log determinant is also calculated and stored with the estimated value used by our algorithms. All experiments using this dataset are averaged over 10 runs with randomly shuffled training and testing sets. The parameter settings are as follows: the budget is set to a reasonably large value, 200, for the log determinant measurements and we sweep over the threshold parameter to find a $\epsilon_{opt} = 0.001$, we used a block size of 5 and 25 for measuring the log determinant and doing regression respectively. We set the regularizer parameters for calculating the log determinant and for solving the linear system as $\lambda = 1.0$ and $\eta = 0.001$ respectively. For Sieve-Streaming the parameter used for determining the coarseness of the generated sieve's Optimal Estimate is set to 0.01, which gave a good approximation for the maximization of the submodular function.

KRLS used around 900 prototypes when not bounded, but

this proved to high for use with the full greedy algorithm so we decided to restrict the budget to 500 prototypes for all algorithms. We can see that KRLS, Online Greedy, Block Greedy, and sieve streaming all perform equally as well. While KRLS does perform slightly better when not restricted to a small number of prototypes, this behavior would be unrealistic in more complex state spaces. The block accuracy is calculated with

$$1 - |\frac{g_{actual} - g_{estimate}}{g_{actual}}|$$

with three variants of our algorithm over various block sizes $\in \{5, 10, 15, 20, 25, 30, 40, 50, 75, 100\}$, figure 1(b). Our algorithm performs very well not only in accuracy, but also in selecting similar prototypes as the Full Greedy algorithm. Finally in the time trials, figure 1(c), we remove all regression implementation and sweep over $b \in \{50, 60, 75, 100, 125, 150, 175, 200, 250, 300, 500\}$. While it is possible to generate less sieves with SieveStreaming through a higher ϵ value, this results in a less accurate approximation of the Online-Greedy prototype selection.

C.3. Details on the United States Census 1990 dataset

The USA Census 1990 dataset, found at (Lichman, 2015), is a large set containing over 2 million samples. Traditionally this set is used for clustering, but is a prime candidate to test efficient methods for submodular function maximization. Because this set is categorical in nature we decided to instead use a Hamming Distance type kernel, where

$$g = \exp(-\frac{\text{hamming}(\mathbf{x}, \mathbf{c})}{2\sigma^2})$$

and the hamming distance $\text{hamming}(\cdot, \cdot)$ is calculated as the number of mismatching attributes. We set the width parameter as $\sigma = 5$, set the budget as $b = 100$ and we used 20 blocks for our algorithm. You can see the behavior of the log determinant of the various methods in Figure 3(c). Our algorithm performs almost as well as the full online greedy algorithm and sieve streaming, but our method requires significantly less time than the full online greedy (about $\frac{3}{1000}$ of the time) and much less memory overhead than the sieve streaming method.

C.4. Details on Santa Fe Data Set A

The *Santa Fe Data Set A* (Weigend, 1994) is a difficult time series prediction data set. Training on the first 1000 time steps of the set, the task is to predict the values in the next 100 steps. We normalize the data to range $[0, 1]$, similar to what was done in (Engel et al., 2004). The KRLS paper also proposed a complicated framework using predictions as apart of the training set, we decided to approach this problem simply by directly training on the 1000 samples through a single iteration and predict using the kernel representation developed during training. Several other parameters are chosen based off settings from the KRLS paper, and our observations about the number of prototypes used by KRLS: RBF width $\sigma = 0.9$, model order $d = 40$, ALD condition $\nu = 0.01$, budget size $b = 310$, and the number of blocks 16. To choose the regularizer parameter and utility gain threshold ϵ, we use $1 - 900$ samples as training set and use $901 - 1000$ samples as a validation set. We set the final parameters as $\epsilon_{opt} = 0.0001$, and $\lambda_{opt} = 0.001$.

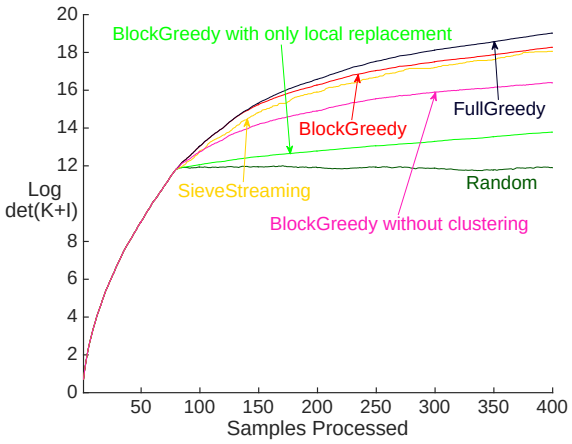
The predictions are in figure 4, and the log determinant as a function of processed samples can be found in figure 4(b). The NMSE (normalized mean squared error, i.e. mean square error divided by variance of the prediction series) our block algorithm achieved is 0.0434. Though slightly worse than the result from KRLS whose NMSE is 0.026 and the first entry whose NMSE is 0.028, our algorithm significantly outperforms the second entry 0.08 in that competition. While our algorithm is outperformed by KRLS, it is important to remember our algorithm has a limited number of weights used and was only trained through one iteration of the first 1000 time steps. KRLS also includes a growing weight vector and other relevant matrices such as $\mathbf{A}_t$. In the original KRLS paper they actually trained on 6×1000 time steps by iteratively incorporating the predictions, but when trained with a single iteration it performs significantly worse with a NMSE of 0.0661.

D. Efficient Modification of the Inverse of the Kernel Matrix

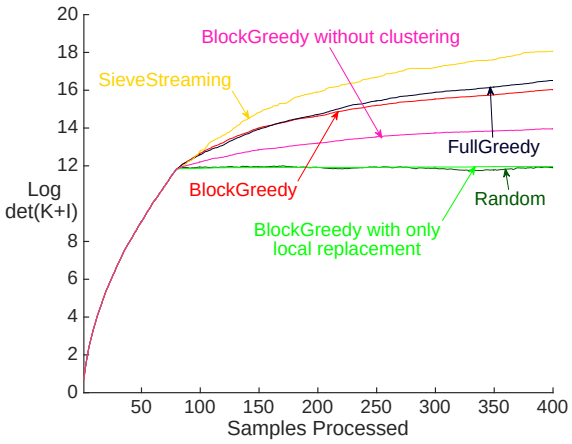
We derive updates when replacing, adding and deleting a prototype, to obtain the new inverse kernel matrix from the previous kernel matrix.

D.1. Replacement Update for the Inverse of the Kernel Matrix

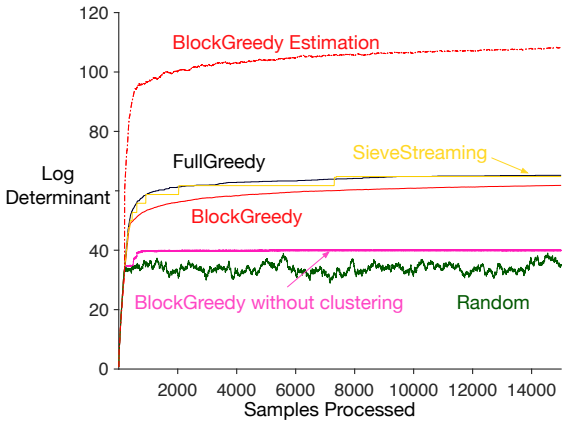
Updating the inverse of the kernel matrix can be regarded as four rank-one updates, which can be further reduced to two rank-one udpates via algebra computation. Let the original kernel matrix be $\mathbf{K}$, we want to replace the ith row and column, $\mathbf{k}^\top$ and $\mathbf{k}$ by new row and column $\tilde{\mathbf{k}}^\top$ and $\tilde{\mathbf{k}}$. Let $\mathbf{x} = \mathbf{0}, \mathbf{x}_i = 1$, $\mathbf{k}^0$ denote the vector by setting $\mathbf{k}_i = 0$, and so as $\tilde{\mathbf{k}}^0$. Then to get the inverse of the replaced kernel matrix, we want to compute $(\mathbf{K} - \mathbf{x}\mathbf{k}^\top - \mathbf{k}^0\mathbf{x}^\top + \mathbf{x}\tilde{\mathbf{k}}^\top + \tilde{\mathbf{k}}^0\mathbf{x}^\top)^{-1}$, which is equal to $(\mathbf{K} + \mathbf{x}(\tilde{\mathbf{k}} - \mathbf{k})^\top + (\tilde{\mathbf{k}}^0 - \mathbf{k}^0)\mathbf{x}^\top)^{-1}$. Note that $\tilde{\mathbf{k}}^0 - \mathbf{k}^0 = \tilde{\mathbf{k}} - \mathbf{k}$, so the final updating rule can be used as algorithm 6 described. One should note that since the regularization parameter λ will be canceled via the minus sign, so the updating rule for regularized kernel matrix will be exactly the same with the one without regularization.



(a) log det vs steps on boston data $\epsilon_t = 0.001$

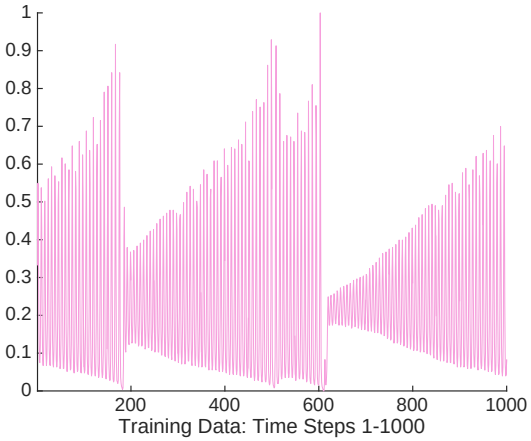


(b) log det vs steps on boston data $\epsilon_t = 0.01$

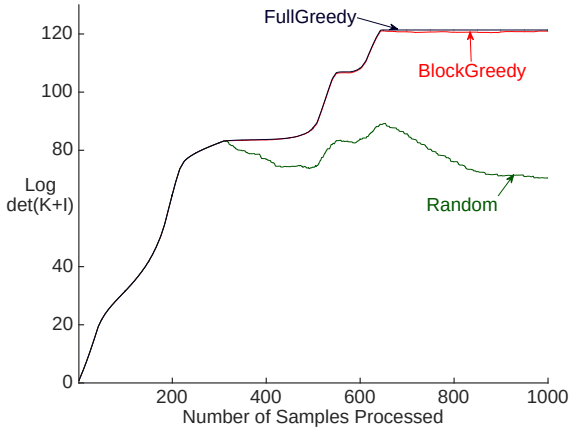


(c) log det Est vs steps on Census data

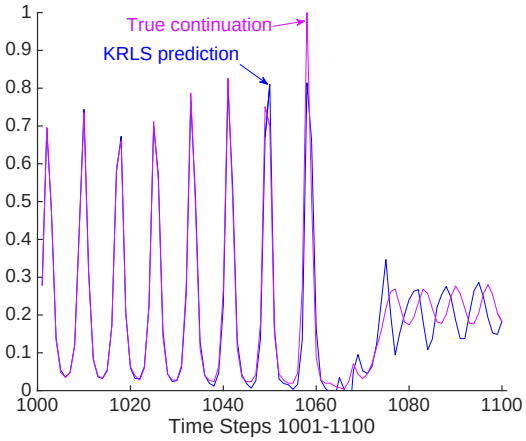
Figure 3. Figure **(a)** and **(b)** show log determinant maximization for each algorithm on boston data. The utility gain threshold is setted different on the two figures. Note that other parameters are the same: budget is 80 and block size is 4. **(c)** Census Data (100 prototypes) - This is the plot showing the true log determinant of each method's true K matrix as calculated with all the prototypes chosen by the algorithm. The color scheme is the same as in figure 1(a).



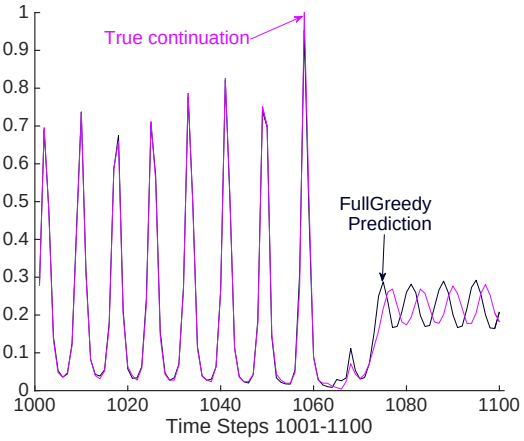
(a) Santa Fe Training Data



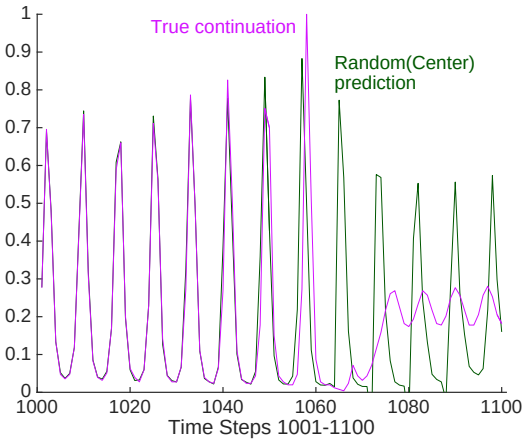
(b) Santa Fe Log Determinant Change Versus Steps



(c) KRLS (one pass) vs True



(d) FullGreedy vs True



(e) Random vs True

Figure 4. Figure **(a)** shows the difficulties on Santa Fe data, it has significant dynamics and noise. **(b)** is the log determinant utility change. One can see that it matches our intuition about from which part we should pick prototypes. For example, figure **b** shows our algorithm is picking prototypes during the steps $600 - 700$, which corresponds to the sharp change in the training set around those time steps. Figures **(c), (d), (e)** show the predictions from KRLS, and regression solutions computed by using prototypes picked randomly and picked by original online greedy algorithm. The greedy method with full kernel matrix can achieve almost the same NMSE with the first entry and original KRLS result (Engel et al., 2004).

D.2. Addition Update for the Inverse of Kernel Matrix

Updating the inverse of kernel matrix when a new row and column attached can be done via block matrix inversion lemma. Let $\mathbf{K}_t$ be the updated matrix from $\mathbf{K}_{t-1}$. Then

$$\mathbf{K}_t^{-1} = \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k} \\ \mathbf{k}^\top & k(\mathbf{x}_t, \mathbf{x}_t) + \lambda \end{bmatrix}^{-1}.$$

Define $v \doteq (k_{xx} + \lambda - \mathbf{k}^\top \mathbf{K}_{t-1}^{-1}\mathbf{k})^{-1}$, then we have:

$$\mathbf{K}_t^{-1} = \begin{bmatrix} \mathbf{K}_{t-1}^{-1} + v\mathbf{K}_{t-1}^{-1}\mathbf{k}\mathbf{k}^\top\mathbf{K}_{t-1}^{-1} & -v\mathbf{K}_{t-1}^{-1}\mathbf{k} \\ -v\mathbf{k}^\top\mathbf{K}_{t-1}^{-1} & v \end{bmatrix}$$

D.3. Deletion Update for the Inverse of the Kernel Matrix

This needs to be used when we delete a prototype from a block. Assume one wants to get $\mathbf{K}_t$ by deleting the ith row and column in the kernel matrix $\mathbf{K}_{t-1}$. To better illustrate the derivation, in below reasoning we rearrange the ith row and column to the last. Define $\mathbf{x} = \mathbf{0}, \mathbf{x}_i = -1$ and $\mathbf{y}$ as the ith column in $\mathbf{K}_{t-1}$ but with $\mathbf{y}_i = 0$. Then one can easily derive the updating rule as following.

$$\begin{aligned} \mathbf{K}_{t-1} &= \begin{bmatrix} \mathbf{K}_t & \mathbf{k} \\ \mathbf{k}^\top & k(\mathbf{x},\mathbf{x}) + \lambda \end{bmatrix} \\ (\mathbf{K}_{t-1} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top)^{-1} &= \begin{bmatrix} \mathbf{K}_t & \mathbf{0} \\ \mathbf{0}^\top & k(\mathbf{x},\mathbf{x}) + \lambda \end{bmatrix}^{-1} \\ &= \begin{bmatrix} \mathbf{K}_t^{-1} & \mathbf{0} \\ \mathbf{0}^\top & (k(\mathbf{x},\mathbf{x}) + \lambda)^{-1} \end{bmatrix} \end{aligned}$$

Let $\tilde{\mathbf{K}_t}^{-1} = (\mathbf{K}^{t-1} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top)^{-1}$, then one can get $\mathbf{K}_t^{-1}$ by removing the last row and column in $\tilde{\mathbf{K}_t}^{-1}$.

E. Incremental Updating Rule for the Log Determinant

E.1. Updating Rule for $\mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top$ Perturbation

Both prototype replacement and deletion require us to efficiently compute $\log\det(\mathbf{K} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top)$. Note that $\det(\mathbf{I} + \mathbf{x}\mathbf{y}^\top) = 1 + \mathbf{x}^\top\mathbf{y}$. For vectors $\mathbf{x}, \mathbf{y}$, we have the following reasoning.

$$\begin{aligned} \det(\mathbf{K} + \mathbf{x}\mathbf{y}^\top) &= \det(\mathbf{K}(\mathbf{I} + \mathbf{K}^{-1}\mathbf{x}\mathbf{y}^\top)) \\ &= \det(\mathbf{K})\det(\mathbf{I} + \mathbf{K}^{-1}\mathbf{x}\mathbf{y}^\top) \\ &= (1 + \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{x})\det(\mathbf{K}) \end{aligned}$$

Taking logarithmic on both sides, one can get:

$$\begin{aligned} \log\det(\mathbf{K} + \mathbf{x}\mathbf{y}^\top) &= \log(1 + \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{x}) \\ &\quad + \log\det\mathbf{K} \\ \tilde{\mathbf{K}} &\doteq \mathbf{K} + \mathbf{x}\mathbf{y}^\top \\ \log\det(\tilde{\mathbf{K}} + \mathbf{y}\mathbf{x}^\top) &= \log(1 + \mathbf{x}^\top\tilde{\mathbf{K}}^{-1}\mathbf{y}) \\ &\quad + \log(1 + \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{x}) \\ &\quad + \log\det\mathbf{K} \end{aligned}$$

To get the term $\mathbf{x}^\top\tilde{\mathbf{K}}^{-1}\mathbf{y}$ in the above formulae, one can apply sherman morrison formulae again:

$$\begin{aligned} \tilde{\mathbf{K}}^{-1} &= \mathbf{K}^{-1} - \frac{\mathbf{K}^{-1}\mathbf{x}\mathbf{y}^\top\mathbf{K}^{-1}}{1 + \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{x}} \\ \mathbf{x}^\top\tilde{\mathbf{K}}^{-1}\mathbf{y} &= \mathbf{x}^\top\mathbf{K}^{-1}\mathbf{y} - \frac{\mathbf{x}^\top\mathbf{K}^{-1}\mathbf{x}\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}}{1 + \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{x}} \end{aligned}$$

Then if we define the following quantities:

$$\begin{aligned} a &\doteq \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{x} = \mathbf{x}^\top\mathbf{K}^{-1}\mathbf{y} \\ b &\doteq \mathbf{x}^\top\mathbf{K}^{-1}\mathbf{x} \\ c &\doteq \mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y} \\ u &\doteq \log\det\mathbf{K} \\ \log\det(\tilde{\mathbf{K}} + \mathbf{y}\mathbf{x}^\top) &= \log(1 + a - \frac{bc}{1+a}) \\ &\quad + \log(1 + a) + u \\ &= \log((1+a)^2 - bc) + u \end{aligned}$$

We will get the final updating rule:

$$\log\det(\mathbf{K} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top) = \log((1+a)^2 - bc) + u$$

One should note that when this formulae is very efficient when $\mathbf{x}$ is a sparse vector, which is the case when replacement or deletion operation are applied. In this case, $\mathbf{x}_i = \pm 1$, which make the computation of a linear time and $b = (\mathbf{K}^{-1})_{ii}$, the only quadratic operation is a matrix-vector product: $\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}$. The total time complexity of this update is two linear operation and one quadratic operation.

By keeping a copy of the inverse of the kernel matrix, we can apply this formula because each prototype replacement can be thought as two rank-one update as described in previous section. With regularization, one can simply replace $\mathbf{K}$ by $\mathbf{K} + \lambda\mathbf{I}$. Everything else is the same.

E.2. Update log det for Attaching

Let $\mathbf{K}_t$ be the matrix after attaching column and row to $\mathbf{K}_{t-1}$.

$$\begin{aligned}\det \mathbf{K}_t &= \det \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k} \\ \mathbf{k}^\top & k(\mathbf{x},\mathbf{x})+\lambda \end{bmatrix} \\ &= \det \mathbf{K}_{t-1} \det(\lambda + k(\mathbf{x},\mathbf{x}) - \mathbf{k}^\top \mathbf{K}_{t-1}^{-1}\mathbf{k}) \\ \log\det \mathbf{K}_t &= \log\det \mathbf{K}_{t-1} + \log\det(\lambda + k(\mathbf{x},\mathbf{x}) \\ &\quad - \mathbf{k}^\top \mathbf{K}_{t-1}^{-1}\mathbf{k})\end{aligned}$$

E.3. Update log det for Deletion

First, let $\mathbf{K}_t$ be the matrix after deleted ith row and column from $\mathbf{K}_{t-1}$. One can first arrange the ith row and column of the $\mathbf{K}_{t-1}$ to the last row and last column. One should note that:

$$\log\det \begin{bmatrix} \mathbf{K}_t & \mathbf{0} \\ \mathbf{0}^\top & k(\mathbf{x},\mathbf{x})+\lambda \end{bmatrix} = \log\det \mathbf{K}_t + \log(1+\lambda) \tag{7}$$

Let $\mathbf{x} = 0, \mathbf{x}_i = -1$ and $\mathbf{y}$ be the ith row (after rearrangement, it becomes the last row) of the $\mathbf{K}_{t-1}$ matrix but set $\mathbf{y}_i = 0$. Then one has:

$$\begin{bmatrix} \mathbf{K}_t & \mathbf{0} \\ \mathbf{0}^\top & k(\mathbf{x},\mathbf{x})+\lambda \end{bmatrix} = \mathbf{K}_{t-1} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top \tag{8}$$

Plug 8 into 7, one can get the final updating rule as:

$$\log\det \mathbf{K}_t = \log\det(\mathbf{K}_{t-1} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top) - \log(1+\lambda)$$

The part $\log\det(\mathbf{K}_{t-1} + \mathbf{x}\mathbf{y}^\top + \mathbf{y}\mathbf{x}^\top)$ can be updated by rank-one update for log det derived in previous section.

F. Time Complexity Analysis

We specify the notations as following. Let r is the block size, b is the budget size, $|\mathcal{B}|$ is the number of blocks, and d is the number of features of an observation. First, we need to be clear about three important mathematical operations: update log determinant when attaching, deleting and replacment. Each of this operation would only take $O(r^2)$, in fact only one quadratic operation is required for each update. Please refer to E.2 for attaching update, E.3 for deleting update, and E.1 for replacement update.

We divide the analysis into two stages. The first stage, $t < b$, we simply accept every prototype resulting in constant time. At the second stage, we need to do clustering first, which costs $O(b^2 d)$. Because we initialize the clustering with the previous clustering, which should be relatively accurate, the number of required steps is reduced. Therefore, we do not pursue a precise clustering, and stop the iteration process after at most $b/|\mathcal{B}|$ steps. Since each clustering step costs $|\mathcal{B}|bd$, and there are at most $b/|\mathcal{B}|$ iterations, the worst-case cost is $O(b^2 d)$. Then, the rest of dominating costs come from the following steps.

1. Finding the nearest cluster. This involves computing distances to all means, costing $O(|\mathcal{B}|d)$. Since $|\mathcal{B}| \leq b$, we get at worst $O(bd)$
2. Updating the book-keeping maps. When we cluster, we recompute book-keeping maps for every block, which costs $O(|\mathcal{B}|r^3)$. Any of the above three updates for computing the maps for each block cost $O(r^3)$. Amortized over b steps, this is $\frac{|\mathcal{B}|r^3}{b}$. Since $|\mathcal{B}| \leq b$, we get at worst $O(r^3)$.
3. Looking for the replacement within the closest block to the current sample. This would cost $O(r^3)$ since each replacement update for log determinant costs $O(r^2)$ and there is at most r replacements in a block.
4. Looking for replacement through other blocks. This would cost only $O(|\mathcal{B}|)$, because the length $|\mathcal{B}|$ booking map has to be scanned.

As a summary, our algorithm would incur time cost $O(bd + r^3)$ at each step. Because $r \ll b$, we get an amortized cost linear in b.