Tight Kernel Query Complexity of Kernel Ridge Regression and Kernel k-means Clustering

Manuel Fernández V, David P. Woodruff, Taisuke Yasuda

Carnegie Mellon University

Kernel Method

- Many machine learning tasks can be expressed as a function of the inner product matrix $\mathbf{G}$ of the data points (rather than the design matrix)
- Easily adapt to an algorithm for the data under a feature map through the use of a *kernel*

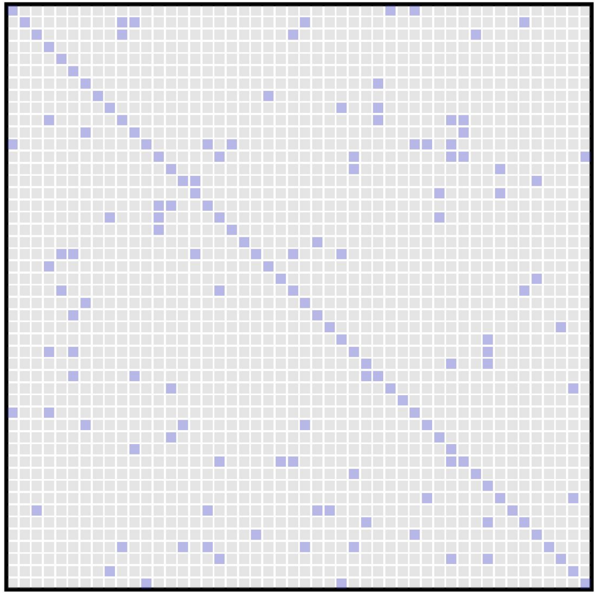
$$\mathbf{G}_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

$$\Downarrow$$

$$\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$$

Kernel Query Complexity

- In this work, we study *kernel query complexity*: the number of entries of the kernel matrix $\mathbf{K}$ read by an algorithm



Kernel Ridge Regression (KRR)

- Kernel method applied to ridge regression

$$\begin{aligned}\boldsymbol{\alpha}_{\text{opt}} &= \underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{K}\boldsymbol{\alpha} - \mathbf{z}\|_2^2 + \lambda\boldsymbol{\alpha}^\top\mathbf{K}\boldsymbol{\alpha} \\ &= (\mathbf{K} + \lambda\mathbf{I}_n)^{-1}\mathbf{z}\end{aligned}$$

- For large data sets, computing the above is prohibitively expensive
- Approximation guarantee

$$\|\hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha}_{\text{opt}}\|_2 \leq \varepsilon \|\boldsymbol{\alpha}_{\text{opt}}\|_2$$

Query-Efficient Algorithms

- State of the art approximation algorithms have *sublinear* and *data-dependent* runtime and query complexity (Musco and Musco NeurIPS 2017, El Alaoui and Mahoney NeurIPS 2015)
- Key quantity: effective statistical dimension

$$d_{\text{eff}}^{\lambda}(\mathbf{K}) := \operatorname{tr}\Big(\mathbf{K}(\mathbf{K}+\lambda \mathbf{I}_n)^{-1}\Big) = \sum_{i=1}^{r} \frac{\sigma_i^2}{\sigma_i^2+\lambda}$$

Query-Efficient Algorithms

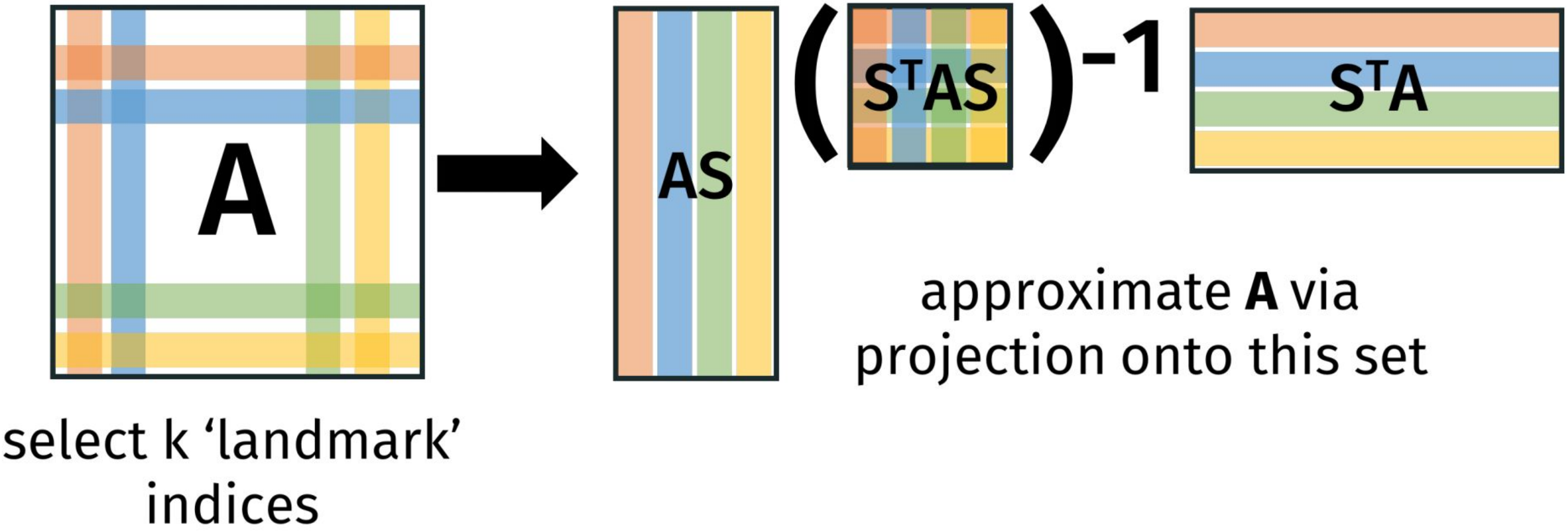


Figure from Cameron Musco's slides

Query-Efficient Algorithms

Theorem (informal)

There is a randomized algorithm computing a $(1+\varepsilon)$*-approximate KRR solution with probability at least 2/3 makes at most* $\tilde{O}(nd_{\mathrm{eff}}^{\lambda}/\varepsilon)$ *kernel queries.*

Is this tight?

Contribution 1: Tight Lower Bounds for KRR

Theorem (informal)

Any randomized algorithm computing a $(1+\varepsilon)$*-approximate KRR solution with probability at least 2/3 makes at least* $\Omega(nd_{\mathrm{eff}}^{\lambda}/\varepsilon)$ *kernel queries.*

- Effective against randomized and adaptive (data-dependent) algorithms
- Tight up to logarithmic factors
- Settles an open question (El Alaoui and Mahoney NeurIPS 2015)

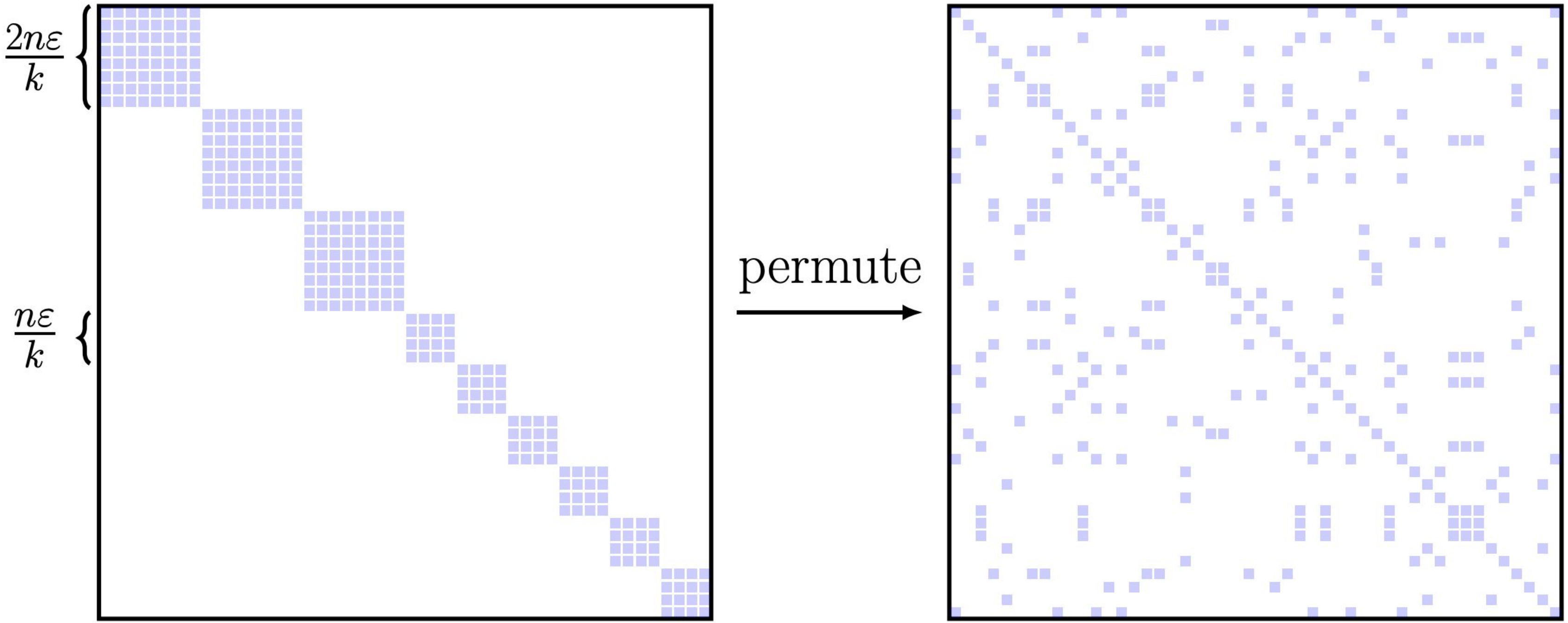
Contribution 1: Tight Lower Bounds for KRR

Proof (sketch)

- Our hard input distribution: all ones vector for the target vector $\mathbf{z}$, regularization $\lambda = n/k$, distribution over binary matrices with effective statistical dimension $d_{\mathrm{eff}}^{\lambda} = \Theta(k)$ and rank $\Theta(k/\varepsilon)$

Contribution 1: Tight Lower Bounds for KRR

- Data distribution μ_{KRR} for the kernel matrix:



Contribution 1: Tight Lower Bounds for KRR

Lemma

Any randomized algorithm for labeling the block size of a constant fraction of rows of a kernel matrix drawn from μ_{KRR} *must read* $\Omega(nk/\varepsilon)$ *kernel entries.*

- Proven using standard techniques

Contribution 1: Tight Lower Bounds for KRR

Reduction

Main Idea: one can just read off the labels of all the rows from the optimal KRR solution, and one can do this for a constant fraction of the rows from an approximate KRR solution.

Contribution 1: Tight Lower Bounds for KRR

Optimal KRR solution

$$\boldsymbol{\alpha}_{\text{opt}} = (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \mathbf{z}$$

Contribution 1: Tight Lower Bounds for KRR

Optimal KRR solution

$$\mathbf{e}_i^\top \boldsymbol{\alpha}_{\text{opt}} = \begin{cases} (2n\varepsilon/k + n/k)^{-1} = \frac{k/n}{1+2\varepsilon} & \text{if row } i \text{ has block size } 2n\varepsilon/k \\ (n\varepsilon/k + n/k)^{-1} = \frac{k/n}{1+\varepsilon} & \text{if row } i \text{ has block size } n\varepsilon/k \end{cases}$$

The entries are separated by a multiplicative $(1 \pm \Omega(\varepsilon))$ factor.

Contribution 1: Tight Lower Bounds for KRR

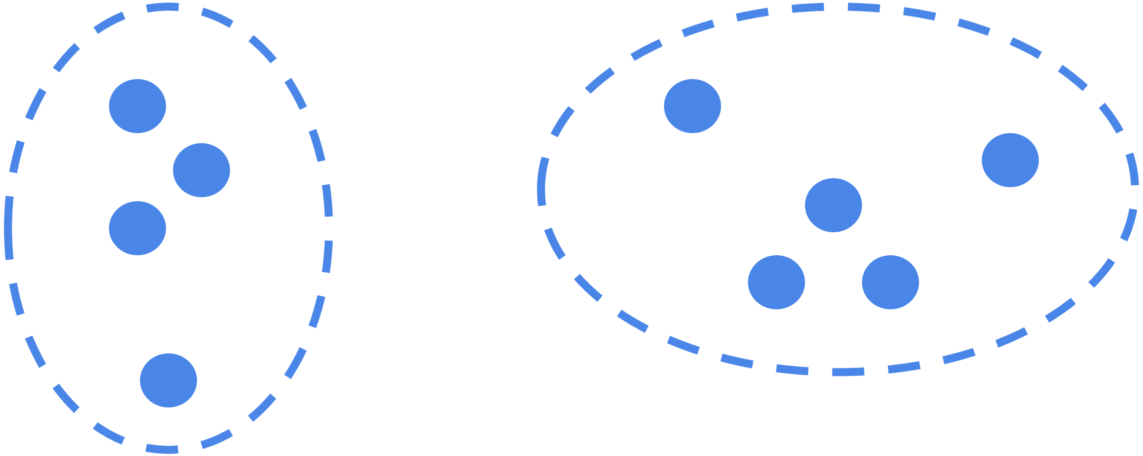
Approximate KRR solution

- By averaging the approximation guarantee over the coordinates, we can still distinguish the cluster sizes for a constant fraction of the coordinates

$$\|\hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha}_{\text{opt}}\|_2 \leq \varepsilon \|\boldsymbol{\alpha}_{\text{opt}}\|_2$$

Kernel k-means Clustering (KKMC)

- Kernel method applied to k-means clustering
- Objective: a partition of the data set into k clusters
- Minimize the cost: sum of squared distances to the nearest centroid



Contribution 2: Tight Lower Bounds for KKMC

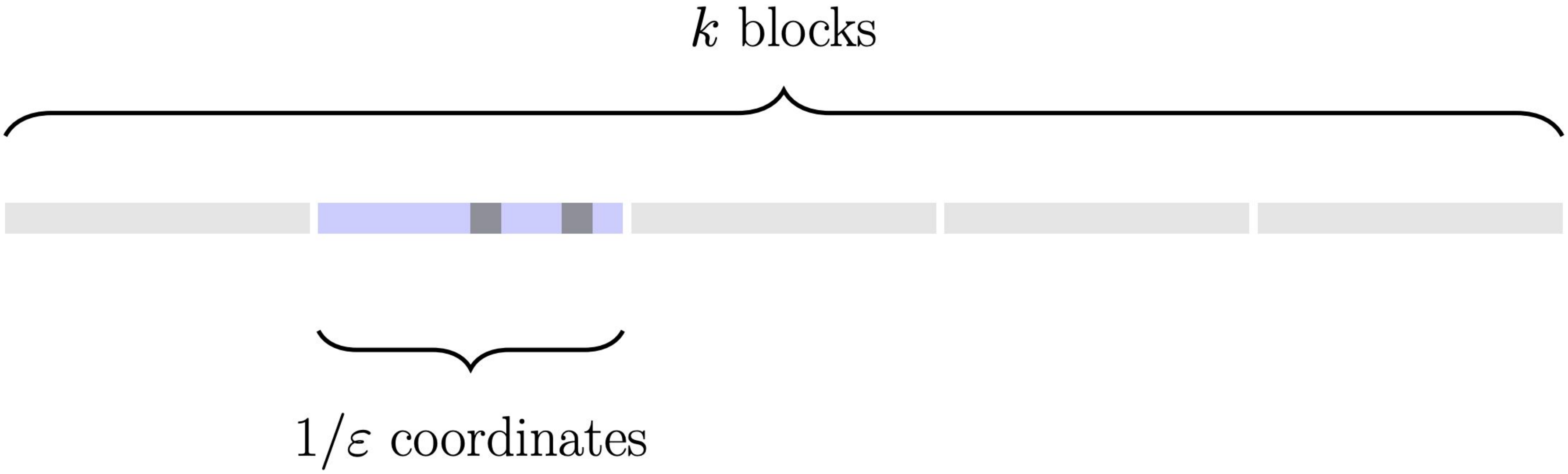
Theorem (informal)

Any randomized algorithm computing a $(1+\varepsilon)$*-approximate KKMC solution with probability at least 2/3 makes at least* $\boldsymbol{\Omega(nk/\varepsilon)}$ *kernel queries.*

- Effective against randomized and adaptive (data-dependent) algorithms
- Tight up to logarithmic factors

Contribution 2: Tight Lower Bounds for KKMC

- Similar techniques, show that a KKMC algorithm must find nonzero entries of a sparse kernel matrix
- Hard distribution is sums of standard basis vectors in $\mathbb{R}^{k/\varepsilon}$



Kernel k-means Clustering of Mixtures of Gaussians

- For input distributions encountered in practice, previous lower bound may be pessimistic
- We show that for a mixture of k isotropic Gaussians with the dot product kernel, we can solve KKMC in only $\tilde{O}(n/\varepsilon)$ kernel queries

Contribution 3: Query-Efficient Algorithm for Mixtures of Gaussians

Theorem (informal)

Given a mixture of k *Gaussians with mean separation* $\tilde{O}(\sigma)$, *there exists a randomized algorithm which returns a* $(1+\varepsilon)$*- approximate* k*-means clustering solution reading* $\tilde{O}(n/\varepsilon)$ *kernel queries with probability at least 2/3.*

Contribution 3: Query-Efficient Algorithm for Mixtures of Gaussians

Main Idea: Johnson-Lindenstrauss Lemma

- Dimension reduction by multiplying data set by a matrix of zero mean Gaussians
- Implemented with few kernel queries since inner products are precomputed