Accurate, Fast and Scalable Kernel Ridge Regression on Parallel and Distributed Systems

speaker: **Yang You**

PhD student at UC Berkeley, advised by James Demmel

with James Demmel\(^1\), Cho-Jui Hsieh\(^2\), and Richard Vuduc\(^3\)

\(^1\)Professor at UC Berkeley
\(^2\)Assistant Professor at UCLA
\(^3\)Associate Professor at Georgia Tech
Outline

- Introduction
- Existing Approaches
- Our Approach
- Analysis and Results
Kernel Ridge Regression (KRR)

- Given $n$ samples $(x_1, y_1), \ldots, (x_n, y_n)$, find the empirical minimizer

$$\hat{\alpha} = \text{argmin} \frac{1}{n} \sum_{i=1}^{n} \| f_i - y_i \|_2^2 + \lambda \| f \|_H^2$$

$$f_i = \sum_{j=1}^{n} \alpha_j \Phi(x_j, x_i) = \sum_{j=1}^{n} \alpha_j \exp(-\| x_i - x_j \|^2 / (2\sigma^2))$$

- This problem has a closed-form solution:

$$(K + \lambda nI) \alpha = y$$

$$f \in \mathbb{R}^n, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}, \alpha \in \mathbb{R}^n, \lambda \in \mathbb{R}, \Phi : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$$

---

$^4$ $H$ is a Reproducing Kernel Hilbert Space

$^5$ $K$ is a $n$-by-$n$ matrix where $K[i][j] = \Phi(x_j, x_i)$, $I$ is an identity matrix
Algorithm 1: Kernel Ridge Regression (KRR)

Input:
- $n$ labeled data points $(x_i, y_i)$ for training;
- another $k$ labeled data points $(\hat{x}_j, \hat{y}_j)$ for testing;
- both $x_i$ and $\hat{x}_j$ are $d$-dimensional vectors;
- $i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, k\}$;
- tuned parameters $\lambda$ and $\sigma$

Output:
Mean Squared Error ($MSE$) of prediction

1. Create a $n$-by-$n$ kernel matrix $K$
2. for $i \in 1 : n$ do
   3. for $j \in 1 : n$ do
      4. $K[i][j] \leftarrow \exp(-||x_i - x_j||^2/(2\sigma^2))$
   5. Solve linear equation $(K + \lambda n I)\alpha = y$ for $\alpha$
3. for $j \in 1 : k$ do
   4. $\hat{y}_j \leftarrow \sum_{i=1}^{n} \alpha_i K(x_i, \hat{x}_j)$
   5. $MSE \leftarrow \frac{1}{k} \sum_{j=1}^{k} (\hat{y}_j - \hat{y}_j)^2$

- $MSE$: correctness metric, lower is better
  - difference between the predicted label and true label
Bottleneck: solve a large linear equation \((K + \lambda nl)\alpha = y\)

- \(n\)-by-\(n\) dense kernel matrix \(K\)
  - machine learning input dataset: a \(n\)-by-\(d\) matrix
  - \(n\): num of samples (e.g. num of users on Facebook: \(\sim 2.2\) billion)
  - \(d\): num of features (e.g. num of movies a user rated: \(\sim 1000\))
  - \(n \gg d\), a small input dataset can generate a huge kernel matrix
    - 357 MB dataset (520,000 \(\times\) 90 matrix) = 2 TB kernel matrix
- \(\Theta(n^3)\) to solve the linear equation directly
  - very expensive in practice

\[K + \lambda nl\]
Weak Scaling Issue

- primary interest for machine learning at scale
  - keep each machine fully loaded (more users, buy more servers)
  - keep $d$ and $n/p$ fixed as $p$ grows ($p$ is # nodes)
- KRR: memory grows as $\Theta(p)$ and the flops as $\Theta(p^2)$ per node
  - perfect scaling: memory and flops are constant per node
Outline

- Introduction
- Existing Approaches
- Our Approach
- Analysis and Results
Bottleneck: solve a large linear equation \( (K + \lambda nl)\alpha = y \)

- Low-rank matrix approximation
  - Kernel PCA (Scholkopf et al., 1998)
  - Incomplete Cholesky Decomposition (Fine and Scheinberg, 2002)
  - Nystrom Sampling (Williams and Seeger, 2001)

- Iterative optimization algorithm
  - Gradient Descent (Raskutti et al., 2011)
  - Conjugate Gradient Methods (Blanchard and Kramer, 2010)

- None of these methods can achieve the same level of accuracy as the direct method does\(^6\)
  - We reserve them for future study

---

\(^6\) Y. Zhang, J. Duchi, M. Wainwright, Divide and Conquer Kernel Ridge Regression, COLT’13
- $K + \lambda nI$ is symmetric positive definite (Cholesky decomposition).
- weak scaling efficiency drops to 0.32% when we increase to 64 nodes
Divide-and-Conquer KRR (DC-KRR)\(^7\)

Communication overhead is low, good scaling!

1. randomly shuffle the \(n\)-by-\(d\) data matrix \(M\)

2. partition \(M\) into \(p\) parts and scatter them to \(p\) nodes

3. each node does local KRR training on \(n/p\) samples and gets its own model

4. average all the models and output a final model

---

\(^7\) Y. Zhang, J. Duchi, M. Wainwright, Divide and Conquer Kernel Ridge Regression, COLT’13
DC-KRR key idea: block-diagonal matrix approximate

Only keep the diagonal blocks, so that the matrix inversion is fast.

\[
K = \begin{pmatrix}
K_{11} & K_{12} & K_{13} & K_{14} & K_{15} & K_{16} \\
K_{21} & K_{22} & K_{23} & K_{24} & K_{25} & K_{26} \\
K_{31} & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\
K_{41} & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\
K_{51} & K_{52} & K_{53} & K_{54} & K_{55} & K_{56} \\
K_{61} & K_{62} & K_{63} & K_{64} & K_{65} & K_{66}
\end{pmatrix}
\Rightarrow

\begin{pmatrix}
K_{33} & K_{36} & 0 & 0 & 0 & 0 \\
K_{63} & K_{66} & 0 & 0 & 0 & 0 \\
0 & 0 & K_{44} & K_{42} & 0 & 0 \\
0 & 0 & K_{24} & K_{22} & 0 & 0 \\
0 & 0 & 0 & 0 & K_{11} & K_{15} \\
0 & 0 & 0 & 0 & K_{61} & K_{55}
\end{pmatrix}

Random Shuffle

Block Diagonalize

\[\text{figure from DC-KRR authors (Y. Zhang, J. Duchi, M. Wainwright)}\]
DC-KRR beats previous methods on tens of nodes

- Figure from DC-KRR authors (Y. Zhang, J. Duchi, M. Wainwright)
- Based on a dataset of music recommendation system
weak scaling in accuracy (i.e. $MSE$)

Table 1: $MSE$: lower is better. 2k samples per node

<table>
<thead>
<tr>
<th>Methods</th>
<th>8k samples</th>
<th>32k samples</th>
<th>128k samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DKRR (baseline)</td>
<td>90.9</td>
<td>85.0</td>
<td>0.002</td>
</tr>
<tr>
<td>DCKRR</td>
<td>88.9</td>
<td>85.5</td>
<td>81.0</td>
</tr>
</tbody>
</table>

- when we scale DC-KRR to many nodes, it is not correct
Outline

- Introduction
- Existing Approaches
- Our Approach
- Analysis and Results
Why DC-KRR does not work at scale?

- It is not safe to ignore the off-diagonal parts
  - there are many nonzero numbers in the off-diagonal parts
- A 5k-by-5k Gaussian Kernel matrix by UCI Covertype dataset, visualized by Matlab spy
How to diagonalize Kernel matrix?

- **k-means clustering algorithm**
  - cluster the samples based on Euclidean distance
  - $x_i$ and $x_j$ are in the same cluster: $||x_i - x_j||$ is small
  - $x_i$ and $x_j$ are in different clusters: $||x_i - x_j||$ is large
  - $||x_i - x_j|| \to \infty$ means $K[i][j] \to 0$
  - $K[i][j] = \Phi(x_j, x_i) = \exp(-||x_i - x_j||^2/(2\sigma^2))$

![Original Kernel vs. After K-means](image)

- nonzero threshold: larger than $10^{-6}$
K-means KRR (KKRR)

1. use k-means to cluster the $n$-by-$d$ data matrix $M$ into $p$ parts

2. scatter $p$ parts to $p$ nodes

3. each node does local KRR training on $n/p$ samples and gets its own model

4. average all the models and output a final model

- we expect KKRR achieves low MSE!
KKRR performs poorly

- our system tries different hyper parameters iteratively, until gets the lowest MSE
- dataset of music recommendation system, on 96 CPU processors
why KKRR performs poorly?

- different clusters are very different from each other
- they generate different models: averaging them is a bad idea
we expect KKRR2 achieves low MSE!
KKRR2 performs much better than KKRR

- our system tries different hyper parameters iteratively, until gets the lowest MSE
- dataset of music recommendation system, on 96 CPU processors
how good this can be in the best situation?

suppose we can select the **best** model (try each one-by-one)
KKRR3: error lower bound for block diagonal method

- we believe KKRR3 will achieve lowest MSE!

1. use k-means to cluster the $n$-by-$d$ data matrix $M$ into $p$ parts

2. scatter $p$ parts to $p$ nodes

3. each node does local KRR training on $n/p$ samples and gets its own model

4. select the model with lowest MSE (cheating, unrealistic method)
Block diagonal is great by an optimal selection algorithm

our system tries different hyper parameters iteratively, until gets the lowest MSE

dataset of music recommendation system, on 96 CPU processors
However, KKRR family is slow

- our system tries different hyper parameters iteratively, until gets the lowest MSE
- dataset of music recommendation system, on 96 CPU processors
K-means clustering: imbalance partitioning

- the sizes of different blocks are different
K-means clustering: imbalance partitioning

1.3 Load Balance for Data Size

- different nodes have different num of samples \((n)\)
- memory: \(\Theta(n^2)\), flops: \(\Theta(n^3)\)
Basic Idea of K-balance algorithm

- Run K-means to get all the cluster centers
- Find the closest center (CC) for a given sample
  - If CC is already balanced, go on
- When every center has $n/p$ samples, done
### K-balance

- **distance matrix**: 8 samples in 4 centers
- \( d[i][j] \) = the distance between i-th center and j-th sample
- balanced case: each center has 2 samples

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
The center for $S_0 \Rightarrow C_2$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C0, C1, C2, C3
- balanced: None
The center for $S_1 \Rightarrow C_3$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C0, C1, C2, C3
- balanced: None
The center for $S2 \Rightarrow C0$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C0, C1, C2, C3
- balanced: None
### The center for S3 ⇒ C3

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- **underload**: C0, C1, C2, C3
- **balanced**: None
The center for $S4 \Rightarrow C0$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C0, C1, C2
- balanced: C3
The center for $S5 \Rightarrow C0 \Rightarrow C3 \Rightarrow C1$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C1, C2
- balanced: C0, C3
The center for $S6 \Rightarrow C3 \Rightarrow C2$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C1, C2
- balanced: C0, C3
The center for $S7 \Rightarrow C0 \Rightarrow C2 \Rightarrow C1$

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: C1
- balanced: C0, C2, C3
Done

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- underload: None
- balanced: C0, C1, C2, C3
by changing k-means to k-balance, we get BKRR, BKRR2, BKRR3
Outline

- Introduction
- Existing Approaches
- Our Approach
- Analysis and Results
## The Datasets

<table>
<thead>
<tr>
<th>name</th>
<th>MSD</th>
<th>cadata</th>
<th>MG</th>
<th>space-ga</th>
</tr>
</thead>
<tbody>
<tr>
<td># Train Samples</td>
<td>463,715</td>
<td>18,432</td>
<td>1,024</td>
<td>2,560</td>
</tr>
<tr>
<td># Test Samples</td>
<td>51,630</td>
<td>2,208</td>
<td>361</td>
<td>547</td>
</tr>
<tr>
<td># Features</td>
<td>90</td>
<td>8</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Application</td>
<td>Music</td>
<td>Housing</td>
<td>Dynamics</td>
<td>Politics</td>
</tr>
</tbody>
</table>
BKRR vs DCKRR

Based on MSD dataset

MSE vs Time (s)
BKRR achieves a good speedup over KKRR

1.5 Load Balance for Data Size

- different nodes have different num of samples ($n$)
- memory: $\Theta(n^2)$, flops: $\Theta(n^3)$
- based on MSD dataset
BKRR2 performs better than DCKRR

1.7 1024 train, 361 test

1.8 2560 train, 547 test

1.9 18432 train, 2208 test
Weak Scaling on MSD dataset

1.10 Time

1.11 Efficiency

1.12 Accuracy
Key idea: difference between BKRR2 and DCKRR

- **BKRR2**: partition the dataset into \( p \) different parts, generate \( p \) different models, and select the best model.

- **DCKRR**: partition the dataset into \( p \) similar parts, generate \( p \) similar models, and use the average of all the models.
Tradeoff: between accuracy and scaling

- BKRR
- DC-KRR
- BKRR2
- BKRR3
- KKRR
- KKRR2/3
- DKRR

Better Scaling vs. Higher Accuracy
Thanks for your time!

- check our source code:
  - https://www.cs.berkeley.edu/~youyang/cakrr.zip