Computationally Efficient Nyström Approximation using Fast Transforms

Si Si
Department of Computer Science
University of Texas at Austin

ICML 2016
Joint work with Cho-Jui Hsieh and Inderjit S. Dhillon
Outline

- Background
- Motivation
- Fast Transforms for Nyström Approximation
  - Construct Structured Landmark Points
  - Using Fast Transforms to Speed up Kernel Value Evaluation
- Experimental Results on Fast Prediction
Popular kernel machines: training set \( \{x_i, y_i\}_{i=1}^{n}, x_i \in \mathbb{R}^d \),

- **Kernel SVM**
  \[
  \alpha^* \leftarrow \arg \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \quad \text{s.t.} \quad 0 \leq \alpha \leq C,
  \]

- **Kernel Ridge Regression**
  \[
  \alpha^* \leftarrow \arg \min_{\alpha} \alpha^T G \alpha + \lambda \alpha^T \alpha - 2 \alpha^T y,
  \]

  \( G \in \mathbb{R}^{n \times n} \) is the kernel matrix; \( G_{ij} = K(x_i, x_j) \); \( Q_{ij} = y_i y_j G_{ij} \).

**Slow Training:**
- \( O(n^2 d) \) time to form the kernel matrix.
- \( O(n^2) \) space to store the kernel matrix.

**Slow Prediction:**
- Kernel SVM: \( y = \text{sign}(\sum_{i=1}^{n} \alpha_i K(x, x_i)) \); \( O(d \# SV) \).
- Kernel Ridge Regression: \( y = \sum_{i=1}^{n} \alpha_i K(x, x_i) \); \( O(nd) \).
Nyström approximation (Williams and Seeger, 2001): based on $m$ landmark points $u_1, \ldots, u_m$:

$$G \approx \tilde{G} = CWC^T.$$ 

Replace $G$ with $\tilde{G}$ in the objective and train the model:

- Speed up training and prediction of kernel machines.
Background: Prediction Using Nyström Approximation

- Perform prediction on a new point $\mathbf{x}$ given the model $\alpha$:

$$\sum_{i=1}^{n} \alpha_i \tilde{K}(\mathbf{x}, \mathbf{x}_i) = \tilde{\mathbf{x}}^T \mathbf{W} \mathbf{C}^T \alpha = \tilde{\mathbf{x}}^T \beta$$

where $\tilde{\mathbf{x}} = [K(\mathbf{x}, \mathbf{u}_1), \ldots, K(\mathbf{x}, \mathbf{u}_m)]^T$.

- Compute $\tilde{\mathbf{x}} = [k(\mathbf{x}, \mathbf{u}_1), \ldots, k(\mathbf{x}, \mathbf{u}_m)]^T$: $O(md)$.
- Compute $\tilde{\mathbf{x}}^T \beta$: $O(d)$.
- Time complexity for prediction: $O(nd) \rightarrow O(md)(n \gg m)$. 

![Diagram showing the relationship between training and prediction stages using Nyström approximation.](image)
Consider the kernel form: $K(x_i, x_j) = f(x_i)f(x_j)g(x_i^T x_j)$.

Gaussian kernel: $f(x) = e^{-\|x\|^2}, g(z) = e^{2z}$.

On mnist dataset, with $n = 60K, m = 160$

total time is 11.15 seconds and $TUx$ takes 10.81 seconds.

The dominant term is $TUx$—usually takes $O(md)$. 
Construct Landmark Points

- How to speed up $Ux$?
  
  **Our solution:** construct structured landmark points based on fast transforms.

- The form of landmark points $U$:

  $U = [H_d V_1, H_d V_2, \ldots, H_d V_m]$. 

- Given initial landmark point $v_i$, $V_i$ is a diagonal matrix:

  $$V_i = \text{diag}(v_{i1}, \cdots, v_{id}).$$

- $H_d$ contains the sign pattern of the Haar or Hadamard matrix:

  $$H_{\text{haar}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 \end{pmatrix}, \quad H_{\text{hadamard}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 0 \\ 1 & 1 & -1 & -1 & 0 \\ 1 & -1 & -1 & 1 & 0 \\ 1 & -1 & -1 & 1 & 0 \end{pmatrix}.$$
Fast Transforms based Landmark Points

One landmark point $v_i$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
</table>

Diagonalize $V_i$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
</table>

$H_d V_i$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

$d$ new landmark points

Haar Matrix $H_d$
Initially $m$ landmark points $v_1, \cdots, v_m$.

The form of new landmark points $U$ (*md landmark points*):

$$U = [H_d v_1, H_d v_2, \ldots, H_d v_m]$$

**m initial landmark points** → $\{v_1, \cdots, v_m\}$ →

$$H_d \begin{bmatrix} v_{11} & 0 & \cdots & 0 \\ 0 & v_{12} & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & v_{1d} \end{bmatrix}$$

→

$$H_d \begin{bmatrix} v_{m1} & 0 & \cdots & 0 \\ 0 & v_{m2} & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & v_{md} \end{bmatrix}$$

**md new landmark points**
Benefit of U

- \( Ux \) will become:

\[
Ux = [(H_d V_1 x), (H_d V_2 x), \ldots, (H_d V_m x)].
\]

- Time for computing one block \( H_d V_i x \):
  - Haar landmark points: \( O(d) \).
  - Hadamard landmark points: \( O(d \log(d)) \).

---

<table>
<thead>
<tr>
<th>( H_d )</th>
<th>( V_i )</th>
<th>( x )</th>
<th>( H_d )</th>
<th>( V_i x )</th>
<th>( H_d V_i x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>a</td>
<td>e</td>
<td>1 1 1 1</td>
<td>ae</td>
<td>ae+bf+cg+dh</td>
</tr>
<tr>
<td>1 1 -1 -1</td>
<td>b</td>
<td>f</td>
<td>1 1 -1 -1</td>
<td>bf</td>
<td>ae+bf-cg-dh</td>
</tr>
<tr>
<td>1 -1 0 0</td>
<td>c</td>
<td>g</td>
<td>1 -1 0 0</td>
<td>cg</td>
<td>ae-bf</td>
</tr>
<tr>
<td>0 0 1 -1</td>
<td>d</td>
<td>h</td>
<td>0 0 1 -1</td>
<td>dh</td>
<td>cg-dh</td>
</tr>
</tbody>
</table>

New landmark points \( H_d V_i \)  
Test point  
Fast transform
Prediction Cost

\[ Ux = [(H_d V_1 x), (H_d V_2 x), \ldots, (H_d V_m x)] \]

- Time complexity analysis (\( U \) has \( md \) landmark points):
  \[ O(md) \]

Prediction time for \( md \) landmark points:
- Haar landmark points: \( O(md) \).
- Hadamard landmark points: \( O(md \log(d)) \).
## Time Complexity Analysis

Table: Time complexity analysis. \( d \) is the dimension of the instance.

<table>
<thead>
<tr>
<th></th>
<th># of initial landmark points</th>
<th># of new landmark points</th>
<th>Prediction time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Nyström</td>
<td>( m )</td>
<td>0</td>
<td>( O(md) )</td>
</tr>
<tr>
<td>Haar landmark points</td>
<td>( m )</td>
<td>( md )</td>
<td>( O(md) )</td>
</tr>
<tr>
<td>Hadamard landmark points</td>
<td>( m )</td>
<td>( md )</td>
<td>( O(md \log d) )</td>
</tr>
</tbody>
</table>

- \( d = 4; m = 4 \); initial landmark points are kmeans centroids

![Graph showing data points and landmark points](image)

**Setting:**
- 4 dimensional data points
- 4 initial landmark points
- 16 new landmark points

**Results:**
- Fast-Nys App. Error: 0.6033
- Kmeans-Nys App. Error: 0.7893
Learning the Structure

- How to learn $U$ automatically?
  - Satisfy the structural constraints $U = [H_d V_1, H_d V_2, \ldots, H_d V_m]$.
  - Minimize the upper bound of the kernel approximation error.

$$
\arg \min_{U \in S} \sum_{i=1}^{n} \left( \min_{t(i)} \| x_i - u_{t(i)} \|^2 \right)
$$

- How to do optimization? Kmeans-like way
  - Update indicator $t(i)$: the landmark point $u_{t(i)}$ that $x_i$ is closest to.
  - Update landmark points $U$: recompute the new centroids under structural constraints.

- Details are in the paper.
The Experimental Setting

- Methods compared in the experiments:
  1. The standard Nyström method (Nys) (Williams and Seeger, 2001);
  2. Kmeans Nyström (KNys) (Zhang et al. 2008);
  3. Random Kitchen Sinks (RKS) (Rahimi and Recht, 2007);
  4. Fastfood with “Hadamard features” (fastfood) (Le et al. 2013);
  5. Pseudo Landmark points (Pseudo) (Hsieh et al. 2014);
  6. The Local Deep Kernel Learning method (LDKL) (Jose et al. 2013);
  7. Divide-and-Conquer based fast Prediction (DC-Pred++) (Hsieh et al. 2014);

- Data set statistics ($n$: number of samples):

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$d$</th>
<th>Dataset</th>
<th>$n$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>usps</td>
<td>9298</td>
<td>256</td>
<td>webspam</td>
<td>350,000</td>
<td>254</td>
</tr>
<tr>
<td>a9a</td>
<td>48,842</td>
<td>123</td>
<td>mnist</td>
<td>60,000</td>
<td>784</td>
</tr>
<tr>
<td>letter</td>
<td>18,000</td>
<td>16</td>
<td>cifar</td>
<td>60,000</td>
<td>400</td>
</tr>
</tbody>
</table>
Results on Fast Prediction

- Prediction Time vs. Prediction Accuracy.
- The unit of prediction time is linear SVM’s prediction time.
- Both Haar and Hadamard landmark points yield similar results.
More Results on Fast Prediction

- Combine Fast-Nys with divide-and-conquer framework: **DC-Fast-Nys**.
- The unit in prediction time is linear SVM prediction time.
- DC-Pred++, and LDKL are two state-of-the-art fast prediction algorithms.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>DC-Fast-Nys</th>
<th>DC-Pred++</th>
<th>LDKL</th>
<th>KNys</th>
<th>RKS</th>
<th>Fastfood</th>
<th>Liblinear</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Prediction Time</td>
<td></td>
<td>Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>letter</td>
<td></td>
<td>7.6x</td>
<td>12.8x</td>
<td>29x</td>
<td>140x</td>
<td>61x</td>
<td>50x</td>
<td>1x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95.4%</td>
<td>95.90%</td>
<td>95.78%</td>
<td>87.58%</td>
<td>89.93%</td>
<td>89.9%</td>
<td>73.08%</td>
</tr>
<tr>
<td>usps</td>
<td>Prediction Time</td>
<td>7.31x</td>
<td>14.4x</td>
<td>12.01x</td>
<td>200x</td>
<td>72.5x</td>
<td>80x</td>
<td>1x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>94.9%</td>
<td>95.56%</td>
<td>95.96%</td>
<td>92.53%</td>
<td>91.33%</td>
<td>94.39%</td>
<td>83.65%</td>
</tr>
<tr>
<td>webspam</td>
<td>Prediction Time</td>
<td>11.21x</td>
<td>20.5x</td>
<td>23x</td>
<td>200x</td>
<td>34.5x</td>
<td>80x</td>
<td>1x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>98.0%</td>
<td>98.4%</td>
<td>95.15%</td>
<td>95.01%</td>
<td>96.4%</td>
<td>96.7%</td>
<td>93.10%</td>
</tr>
</tbody>
</table>
Conclusions

- Observation: Computing $Ux$ is the bottleneck.
  - Construct Structured Landmark Points.
  - Using Fast Transforms to Speed up Kernel Value Evaluation.
- Experimental Results on Fast Prediction.

Our poster # 66 — Tuesday 10:00am to 1:00pm
References

Algorithm

- Learn the initial $m$ landmark points.
- Construct $md$ new landmark points.
- Use $md$ new landmark points during training and prediction.

One landmark point $V_i$

$$\arg\min_{U \in S} \sum_{i=1}^{n} \left( \min_{t(i)} \|x_i - u_{t(i)}\|^2 \right).$$

Diagonalize $V_i$

Haar Matrix $H_d$
Learning Structure of Landmark Points

- Learning vs. not learning

![Graph showing relative approximation error over time for different methods]

- Hadamard
- Haar
- Haar with structure constraint
Relative kernel approximation error $\|G - \hat{G}\|_F / \|G\|_F$.

- Time vs. kernel approximation error:

(c) a9a, polynomial.

(d) mnist, Gaussian.