Hierarchical Decompositions of Kernel Matrices

Bill March
UT Austin
Dec. 12, 2015

Joint work with Bo Xiao, Chenhan Yu, Sameer Tharakan, and George Biros

On the job market!
Kernel Matrix Approximation

Inputs:
points \( x_i \in \mathbb{R}^d \) \( i = 1, \ldots, N \)
d \( d > 3 \)
kernel function \( \mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \)
weights \( w \in \mathbb{R}^N \)

Output:
\( u = K w \) \( \text{ where } K_{ij} = \mathcal{K}(x_i, x_j) \)

Exact Evaluation: \( O(N^2) \)
Fast Approximations: \( O(N \log N) \) or \( O(N) \)
Low Rank Approximation

\[ O(Nr^2) \] work
with sampling / Nystrom methods

\[
\begin{array}{cc}
\text{low rank} & \approx \text{full rank} \\
0.35 & 71.6 \\
0.22 & 74.0 \\
0.14 & 79.8 \\
0.02 & 95.4 \\
0.001 & 6.4 \\
\end{array}
\]

Bayes Classifier with Gaussian KDE
Hierarchical Approximations

— How do we know how to partition the matrix?
— How do we approximate the low-rank blocks?
Related Work

- Nystrom methods [Williams & Seeger, ’01; Drineas & Mahoney, ’05]: scalable, can be parallelized, require entire matrix to be low rank
- FMMs: [Greengard, ’85; Lashuk et al., ’12] — \( N > 10^{12} \), high accuracy, kernel specific, \( d = 3 \)
- FGTs:[Griebel et al., ’12]:  200K points, synthetic 20D, real 6D, low-order accuracy, sequential
- Other hierarchical kernel matrix factorizations & applications:
  - [Si et al., ’14] — block Nystrom factoring
  - [Zhong et al., ’12] — collaborative filtering
  - [Ballani & Kressner, ’14] — QUIC, sparse covariance inverses
  - [Borm & Garcke, ’07] — H matrices for kernels
  - [Wang et al, ’15] — block basis factorization
  - [Gray & Moore, ’00] — general kernel summation treecode
  - [Lee, et al., ’12] — kernel independent, parallel treecode, works in modestly high dimensions
ASKIT — Approximate Skeletonization Kernel-Independent Treecode

- ASKIT is a *kernel-independent* algorithm that scales with $N$ and $d$

- Uses *nearest neighbor information* to capture local structure

- Randomized linear algebra to compute approximations

- *Scalable, parallel* implementation and open-source library LIBASKIT
Hierarchical Approximations

— How do we know how to partition the matrix?
— How do we approximate the low-rank blocks?
Keys to ASKIT:
Skeletonization

- Approximate the interaction of a node with all other points
- Use a basis of columns

But requires $O(N m^2)$ work!
Keys to ASKIT: Randomized Factorization

- Subsample $\ell$ rows and factor
- Construct a sampling distribution using nearest neighbors to capture important rows
Keys to ASKIT: Combinatorial Pruning Rule

• When can we safely use the approximation?

• Use nearest neighbor information — any node containing a nearest neighbor must be evaluated exactly
Keys to ASKIT: Combinatorial Pruning Rule

- When can we safely use the approximation?
- Use nearest neighbor information — any node containing a nearest neighbor must be evaluated exactly
Keys to ASKIT: Combinatorial Pruning Rule

• When can we safely use the approximation?

• Use nearest neighbor information — any node containing a nearest neighbor must be evaluated exactly
Overall ASKIT Algorithm

- Inputs: coordinates, NN info
- Construct space partitioning tree
- Compute approximate factorizations using randomized linear algebra (Upward pass)
- Construct interaction lists using neighbor information and merge lists for FMM node-to-node lists
- Evaluate approximate potentials (Downward pass)
Theoretical Bounds

- Error: \( C \log(N) \sigma_{s+1}(G) \)

- Complexity:
  - Storage: \( (\kappa + s^2) \frac{Nd}{p} \)
  - Factorization: \( s^2 \frac{N}{p} + s^3 \log p \)
  - Evaluation: \( \frac{N}{p} \left( \kappa s \log \left( \frac{N}{s} \right) \right) \)
Accuracy and Work

<table>
<thead>
<tr>
<th>Data</th>
<th>$N$</th>
<th>$d$</th>
<th>$\epsilon_2$</th>
<th>$%K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>1M</td>
<td>64</td>
<td>5E-3</td>
<td>1.6%</td>
</tr>
<tr>
<td>Covtype</td>
<td>500K</td>
<td>54</td>
<td>8E-2</td>
<td>2.7%</td>
</tr>
<tr>
<td>SUSY</td>
<td>4.5M</td>
<td>18</td>
<td>5E-3</td>
<td>0.4%</td>
</tr>
<tr>
<td>HIGGS</td>
<td>10.5M</td>
<td>28</td>
<td>1E-1</td>
<td>11%</td>
</tr>
<tr>
<td>BRAIN</td>
<td>10.5M</td>
<td>246</td>
<td>5E-3</td>
<td>0.9%</td>
</tr>
</tbody>
</table>

Relative errors and fraction of kernel evaluations
Strong Scaling

<table>
<thead>
<tr>
<th>#cores</th>
<th>512</th>
<th>2,048</th>
<th>4,096</th>
<th>8,192</th>
<th>16,384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>2,297</td>
<td>778</td>
<td>544</td>
<td>438</td>
<td>363</td>
</tr>
<tr>
<td>Eval.</td>
<td>157</td>
<td>67</td>
<td>42</td>
<td>28</td>
<td>23</td>
</tr>
<tr>
<td>Eff.</td>
<td>1.00</td>
<td>0.72</td>
<td>0.50</td>
<td>0.32</td>
<td>0.20</td>
</tr>
</tbody>
</table>

HIGGS data, 11M points, 28d

\[k = 1024, s = 2048\]

Estimated L2 error = 0.09
The BRAIN experiments use Table 7, which results in over 20% of the direct evaluation interactions potentials down the tree (Alg. 5). As the neighbor lists overlap less, even though the problem, the method will still be much slower than the FMM. Even so, the method will still be much slower than the FMM.

In practice, we need to adjust the bandwidth as we change the number of kernel evaluations per target. The SUSY experiments use the ASKIT-FMM algorithm without changing other algorithm parameters. We report timings in seconds for different kernel functions.

<table>
<thead>
<tr>
<th>#cores</th>
<th>Mat-vec</th>
<th>Fact.</th>
<th>Inv.</th>
<th>Total</th>
<th>Eff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>1.5</td>
<td>95.1</td>
<td>0.9</td>
<td>96</td>
<td>1.00</td>
</tr>
<tr>
<td>2,048</td>
<td>0.8</td>
<td>51.4</td>
<td>0.5</td>
<td>52</td>
<td>0.92</td>
</tr>
<tr>
<td>4,096</td>
<td>0.4</td>
<td>29.0</td>
<td>0.3</td>
<td>30</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Normal data, 16M points, 64d (6 intrinsic)

\[ k = 128, \quad s = 256 \]

Inverse error = 4E-6

INV-ASKIT

Approximates \((\lambda I + K)^{-1}\)
Summary

• ASKIT is a kernel independent FMM that scales with dimension

  • Efficient and scalable, but requires geometric information

• Inv-ASKIT — can efficiently compute approximate inverses, also useful as a preconditioner

• Open-source, parallel library available — LIBASKIT

For code and papers:
www.ices.utexas.edu/~march
padas.ices.utexas.edu/libaskit