A Study of Clustering Techniques and Hierarchical Matrix Formats for Kernel Ridge Regression

Xiaoye Sherry Li, xsli@lbl.gov
Liza Rebrova, Gustavo Chavez, Yang Liu, Pieter Ghysels
Lawrence Berkeley National Laboratory

IPAM Workshop: Big Data Meets Large-Scale Computing

Plan

- Introduction of Kernel Ridge Regression
- How to improve efficieny of KRR?
  - compressed representation of kernel matrices
  - clustering points
  - hyperparameter optimization
  - sampling methods
  - ...

Kernel matrices

Intuitively, kernel matrices can be viewed as similarity matrices

$$K(i,j) = \text{ similarity score } x_i \leftrightarrow x_j,$$

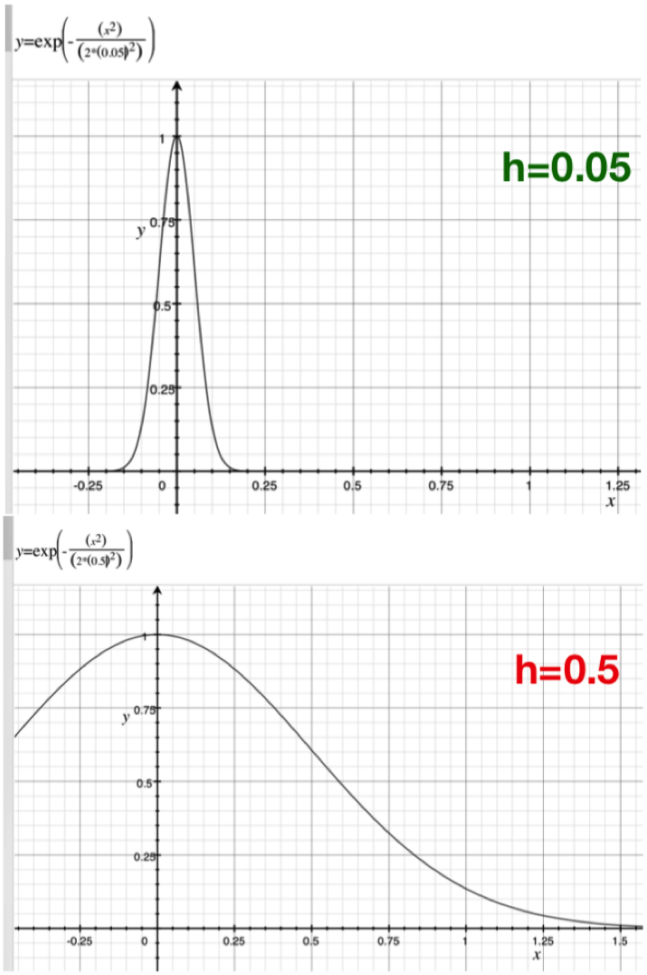
where K is $n \times n$ positive semi-definite matrix, defined by a set of objects $x_1, \ldots, x_n$.

Kernel example

Gaussian kernel

Similarity is closeness in Euclidean distance, h is reweighting.

$$K(i,j) = \exp\left(-\frac{\|x_i - x_j\|_2^2}{2h^2}\right)$$



More kernel examples

- Laplacian kernel

$$K(i,j) = \exp\left(-\frac{\|x_i - x_j\|_2}{h}\right)$$

- Linear kernel Similarity is the length of the projection of one vector on another.

$$K(i,j) = \langle x_i, x_j \rangle$$

- Degree d polynomial kernel
- Sigmoid neural network kernel
- ...

Kernel Ridge Regression
= Ridge Regression + Kernel Trick

1. Ridge regression

Want to minimize the cost function:

$$C(\mathbf{w}) = \sum_i (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|^2 \to \min$$

- x_i's are data points (rows of the data matrix $X^{n \times d}$)
- y_i's are their labels
- w is the normal vector to the target hyperplane

Differentiating

$$C'(\mathbf{w})_j = 2 \sum_i (y_i - \mathbf{w}^T \mathbf{x}_i)\mathbf{x}_j + 2\lambda \mathbf{w}_j = 0$$

In matrix form, the argmin is

$$\mathbf{w} = X^T (\lambda I + X X^T)^{-1} \mathbf{y},$$

where X - train matrix, y - a vector of train labels.

So, we do regression with the optimal weights

$$\mathbf{w} = X^T(\lambda I + XX^T)^{-1}\mathbf{y}$$

Prediction step: given x' - a vector from the test set,

$$\begin{aligned} y' := \mathbf{w}^T\mathbf{x'} &= [(\lambda I + XX^T)^{-1}\mathbf{y}]^T X\mathbf{x'} \\ &= [(\lambda I + \mathcal{K}(X,X))^{-1}\mathbf{y}]^T \cdot \mathcal{K}(X,\mathbf{x'}) \leftarrow \text{kernel trick} \end{aligned}$$

Regularization term λ helps to stabilize the numerical inverse by bounding the smallest eigenvalues away from zero.

Kernel Ridge Regression

2. Kernel trick

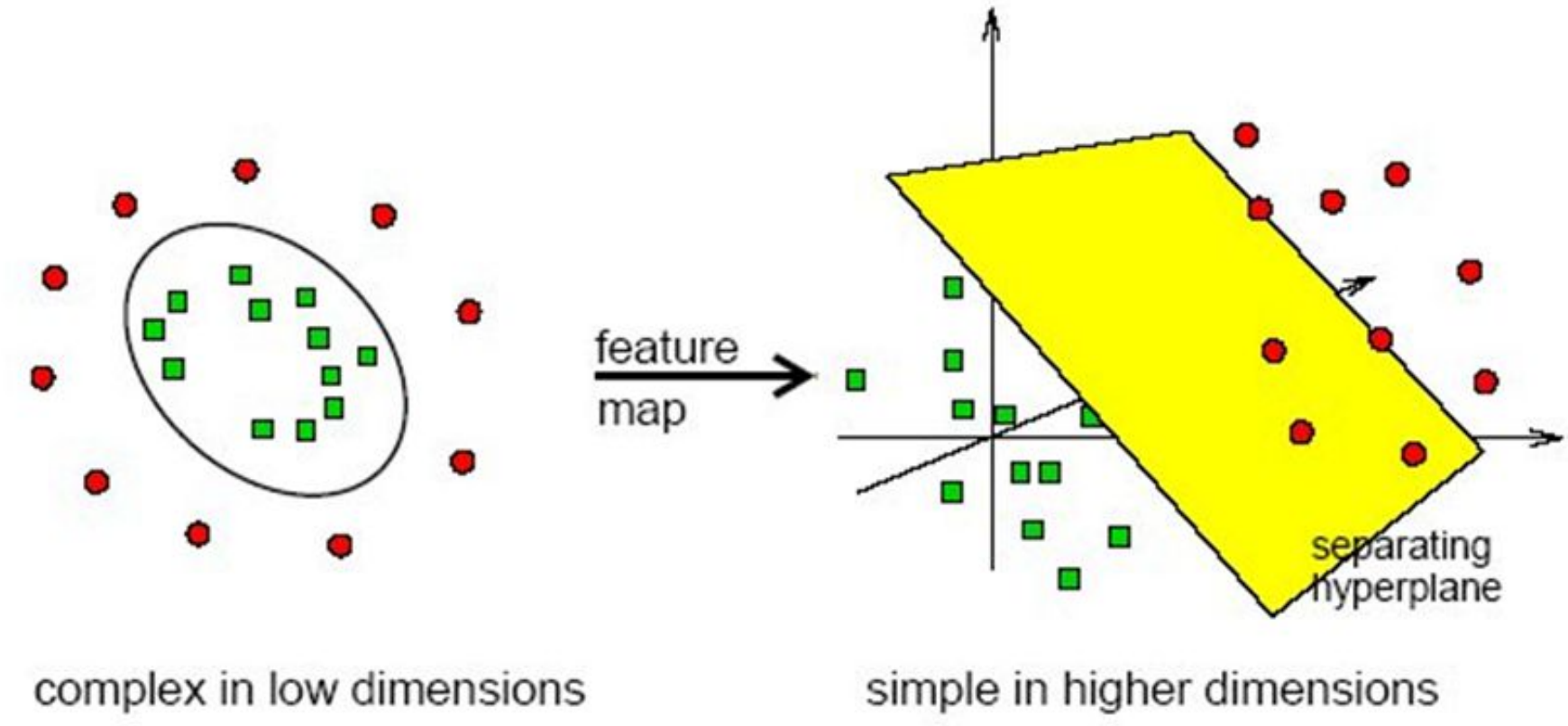
- map points to higher-dimensional feature space using function $\varphi(x)$ (e.g. $x \mapsto \varphi(x)]$).
- replace scalar product in the new space by kernel function K

$$K(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle.$$

Properties of K:

- positive semi-definite
- often much faster to compute than $\varphi(x)$ themselves
- can be applied if method depends on scalar products

Separation can be easier in higher dimensions



(picture is found in Radha Chitta presentation on Kernel K-Means)

Kernel matrix can be easy to compute

Suppose higher-dimensional features are second order terms of $x_i's$:

$$\varphi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_2x_1 \\ .... \\ x_nx_n \end{bmatrix}$$

Then the scalar product

$$K(x,z) := \varphi(x)^T\varphi(z) = \sum_{i,j=1}^{n}(x_ix_j)(z_iz_j) =$$

$$= (\sum_{i=1}^{n} x_iz_i)(\sum_{j=1}^{n} x_jz_j) = (x^Tz)^2.$$

Note: calculating the high-dimensional $\varphi(x)$ requires $O(n^2)$ time, finding $K(x,z)$ takes only $O(n)$ time - linear in dimension of input attributes.

Algorithm for binary classification with the Gaussian kernel

1. Compute kernel matrix K:

$$K(i,j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2h^2}\right)$$

   for all $\mathbf{x}_i, \mathbf{x}_j$ from the train set

2. Compute the weights by solving linear system ($\mathbf{y}$ - train labels)

$$\mathbf{v}: \quad \mathbf{y} = (\lambda I + K)\mathbf{v}$$

3. Compute kernel vector K' for the test vector $\mathbf{x}$':

$$K'(i) = \exp(-\|\mathbf{x}_i - \mathbf{x}\text{'}\|_2^2/2h^2)$$

4. Predict the sign

$$y' := \operatorname{sign}\langle \mathbf{v}, K' \rangle$$

Ways to improve KRR efficiency

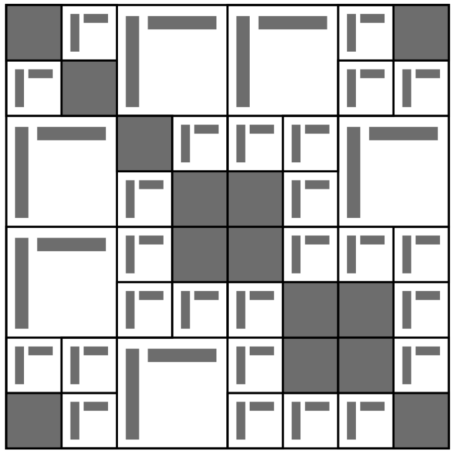
- Fast equation solve: compressed representation of kernel matrices
- Clustering points
- Hyperparameter optimization
- Sampling methods
- ...

Fast equation solve

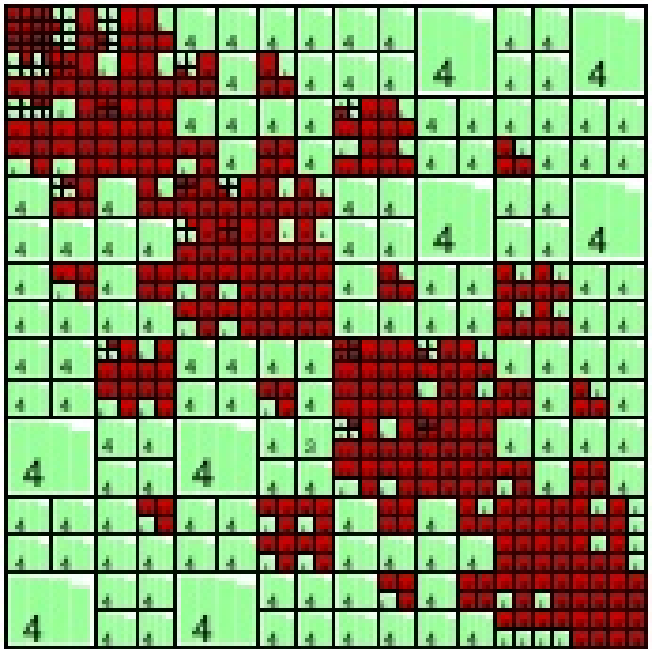
Kernel matrices are good candidates for compression:

- contain many similar elements, amenable to low-rank compression
- but usually full rank - ones on diagonal, off-diagonal blocks are low-rank

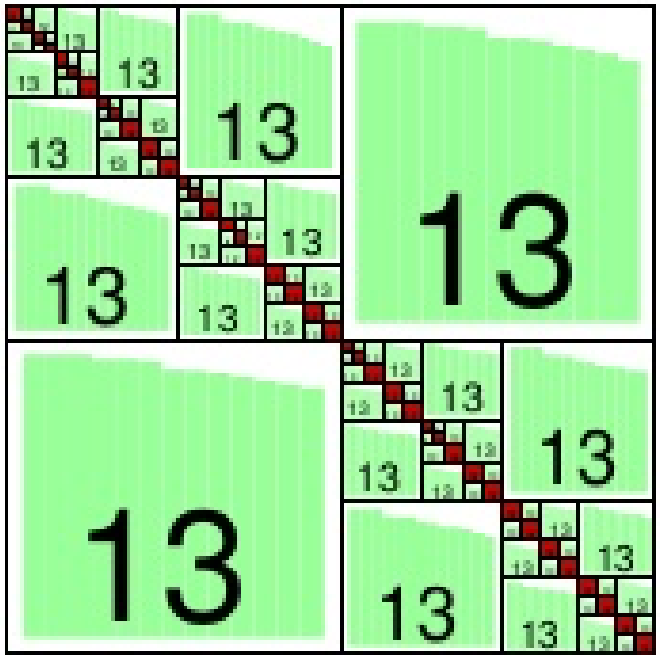
Hierarchical matrices are compressed representation of dense matrices.



Hierarchical matrix formats

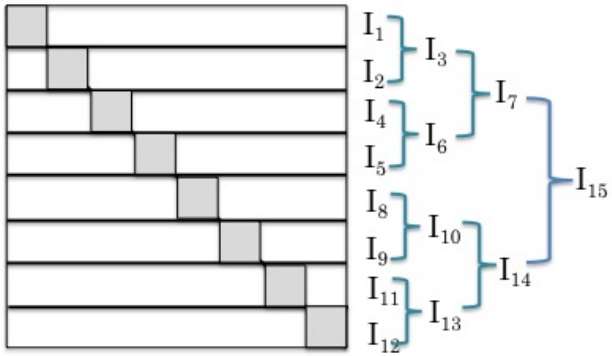


H-matrix (W. Hackbusch et al.)
O(r N log N)



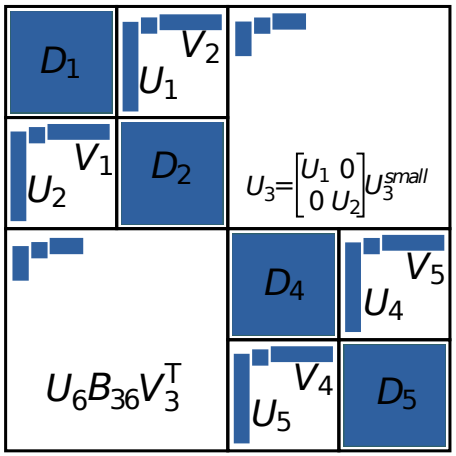
HSS matrix (J Xia et al.)
O(r N)

Hierarchically Semi-Separable matrices (HSS)



- Diagonal blocks are full rank: $D_\tau = A(I_\tau, I_\tau)$
- Off-diagonal blocks as low-rank:

$$A_{\nu_1,\nu_2} = A(I_{\nu_1}, I_{\nu_2}) = U_{\nu_1} B_{\nu_1,\nu_2} V^*_{\nu_2}$$



- **Column bases U and row bases V^* are nested:**

$$U_\tau = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} U^{\text{small}}_\tau, V_\tau = \begin{bmatrix} V_{\nu_1} & 0 \\ 0 & V_{\nu_2} \end{bmatrix} V^{\text{small}}_\tau$$

HSS matrix – ULV factorization

ULV-like factored form (U and V^* unitary, L triangular)

$$\Gamma_{1;b\leftrightarrow 2;t}\begin{bmatrix} I & & \\ & \Omega_1 & \\ & & I \\ & & & \Omega_2 \end{bmatrix}\begin{bmatrix} \Gamma_{3;b\leftrightarrow 4;t} & \\ & \Gamma_{5;b\leftrightarrow 6;t} \end{bmatrix}\begin{bmatrix} \Omega_3 & & \\ & \Omega_4 & \\ & & \Omega_5 \\ & & & \Omega_6 \end{bmatrix} A \begin{bmatrix} Q_3^* & & \\ & Q_4^* & \\ & & Q_5^* \\ & & & Q_6^* \end{bmatrix}\begin{bmatrix} \Gamma_{3;b\leftrightarrow 4;t}^T & \\ & \Gamma_{5;b\leftrightarrow 6;t}^T \end{bmatrix}\begin{bmatrix} I & & \\ & Q_1^* & \\ & & I \\ & & & Q_2^* \end{bmatrix}\Gamma_{1;b\leftrightarrow 2;t}^T$$

$$= \left[\begin{array}{cccccc} L_3 & & & & & \\ 0 & L_4 & & & & \\ (\Omega_1 L_{4,3})_t & (\Omega_1 L_{3,4})_t & L_1 & & & \\ & & & L_5 & & \\ & 0 & & 0 & L_6 & \\ & & & (\Omega_2 L_{6,5})_t & (\Omega_2 L_{5,6})_t & L_2 \\ (\Omega_1 L_{4,3})_b & (\Omega_1 L_{3,4})_b & W_{1;b}Q_{1;t}^* & B_{1,2}V_2^*\begin{bmatrix} V_5^*Q_{5;t}^* & V_5^*Q_{5;b}^* \\ & V_6^*Q_{6;t}^* \end{bmatrix}\begin{bmatrix} I & \\ & Q_2^* \end{bmatrix} & & D_0 \\ B_{2,1}V_1^*\begin{bmatrix} V_3^*Q_{3;t}^* & V_3^*Q_{3;b}^* \\ & V_4^*Q_{4;t}^* \end{bmatrix}\begin{bmatrix} I & \\ & Q_1^* \end{bmatrix} & & & (\Omega_2 L_{6,5})_b & (\Omega_2 L_{5,6})_b & W_{2;b}Q_{2;t}^* \end{array}\right]$$

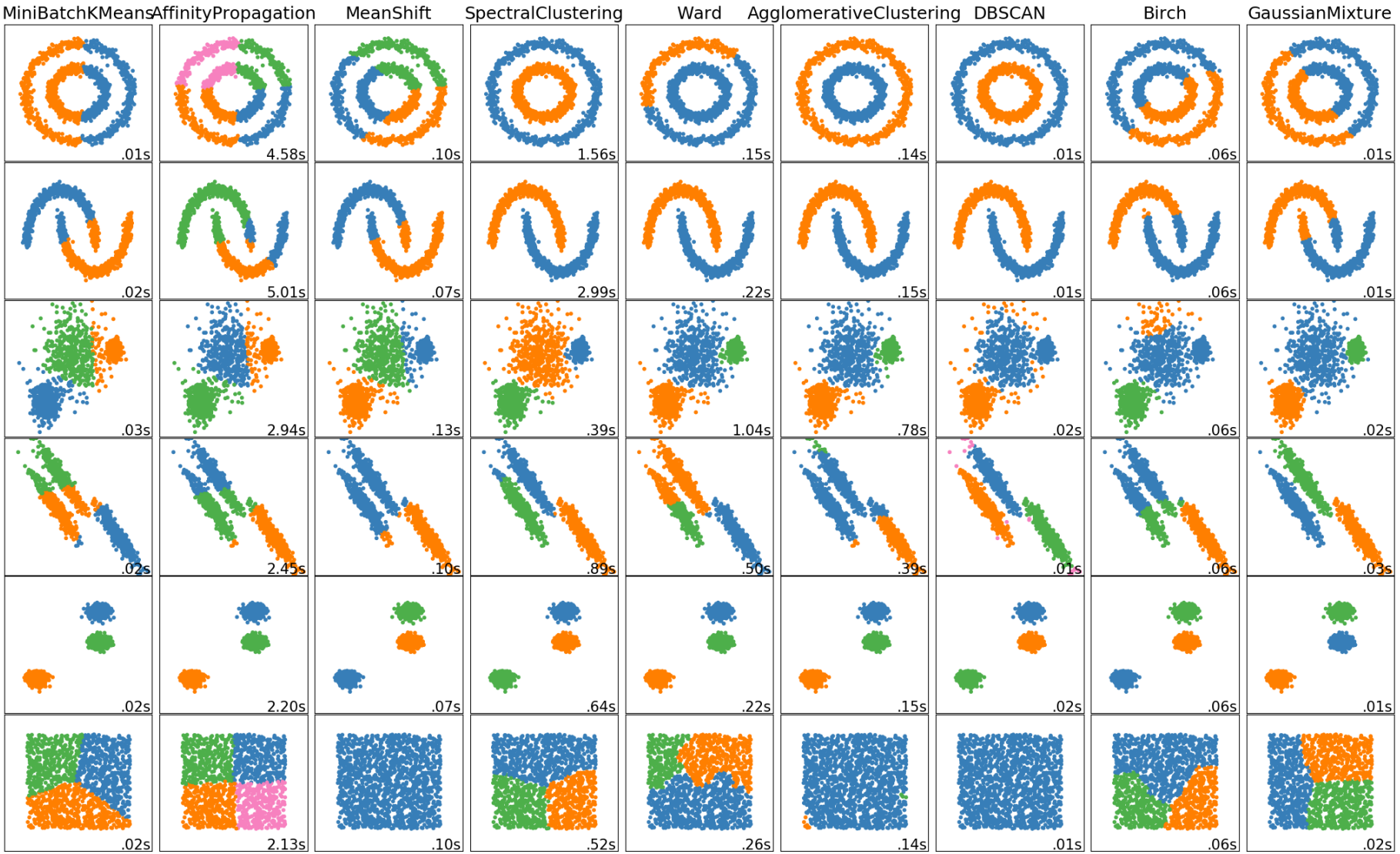
Clustering

Goal: in the context of kernel matrix, find good ordering to improve low rankness.

1. Find groups of points with large between-group distances and small within-group distances
2. Permute matrix K so that the points of each group occupy consecutive indices, so they will form dense diagonal blocks

Clustering methods

Machine Learning in Python, http://scikit-learn.org/stable/

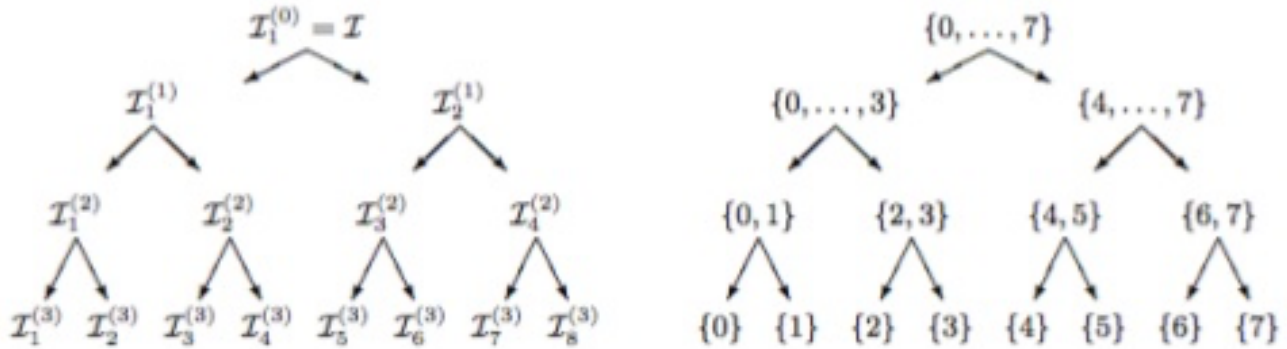


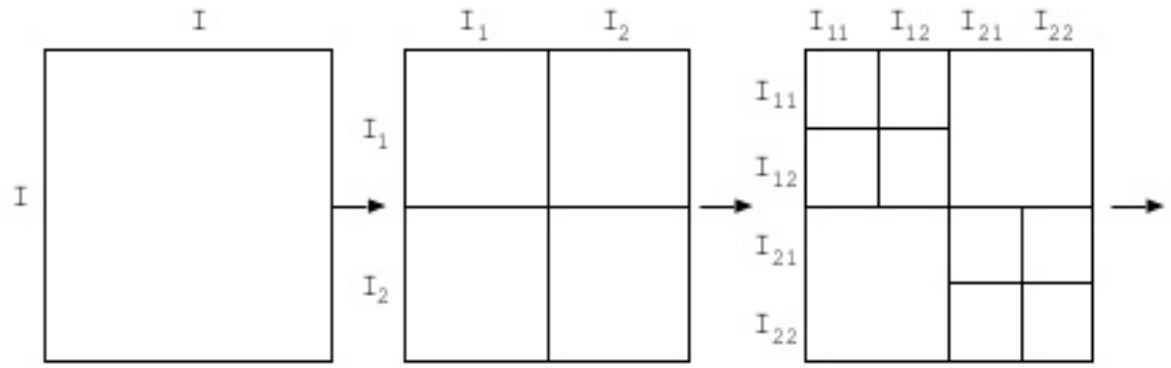
Specific requirements on clustering

- Clusters should be small enough (otherwise dense HSS-blocks use too much memory)
- Clusters should have similar size
- Need to construct the whole hierarchical tree of embedded clusters

HSS tree reminder

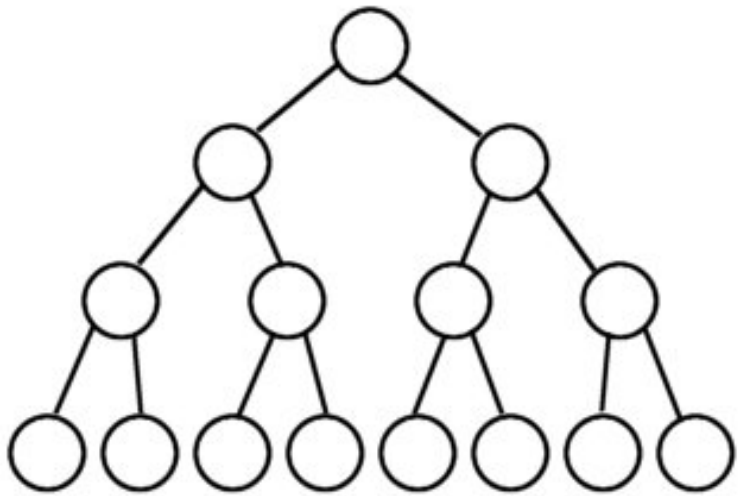
Every vertex of a tree corresponds to a consecutive range of indices, representing sub-block. In every iteration this range is splitted into two children sub-ranges.





Trees

1. Natural tree

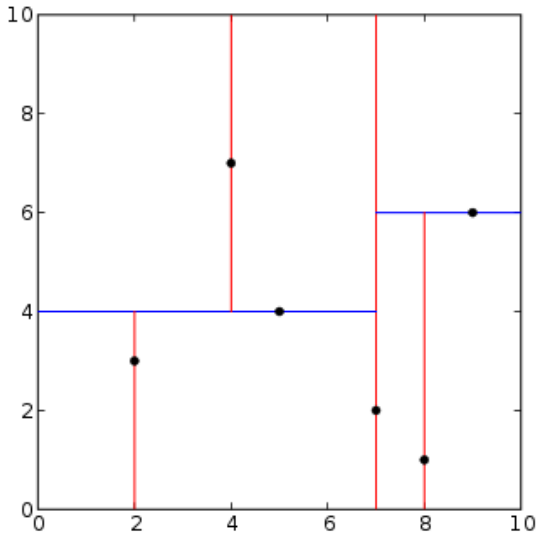


- number of levels is chosen to get good leaf size
- does not use any information about mutual distances
- least efficient method

Divisive trees

2. KD tree

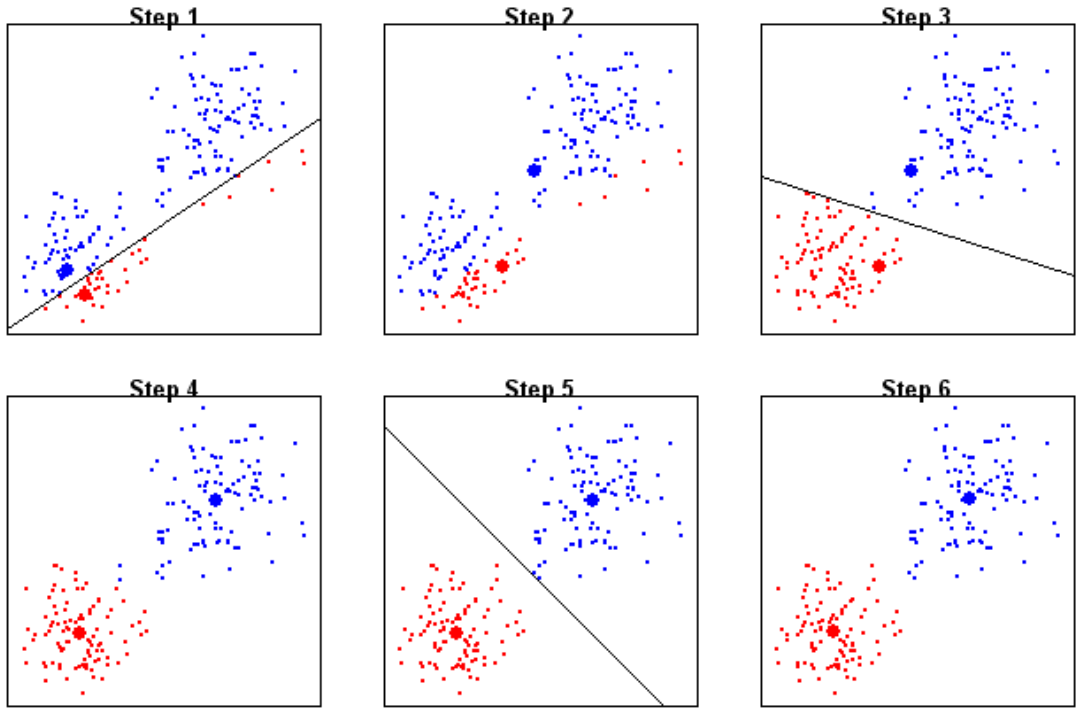
- At every step, choose some dimension (feature) $i \in \{1, \ldots, d\}$, as the splitting pivot
- Split all the points in two classes with respect to a mean/median of the column i
- Better to choose pivot direction as the direction of maximum spread
- Easy to implement
  widely used in similar tasks
- Requires small number of features
  (tall-skinny data matrix)



Divisive trees

3. Recursive two means

a) Start with dividing all the points into two clusters



b) Divide each cluster into two using the same method

c) Continue splitting until *min cluster size* achieved

Divisive trees

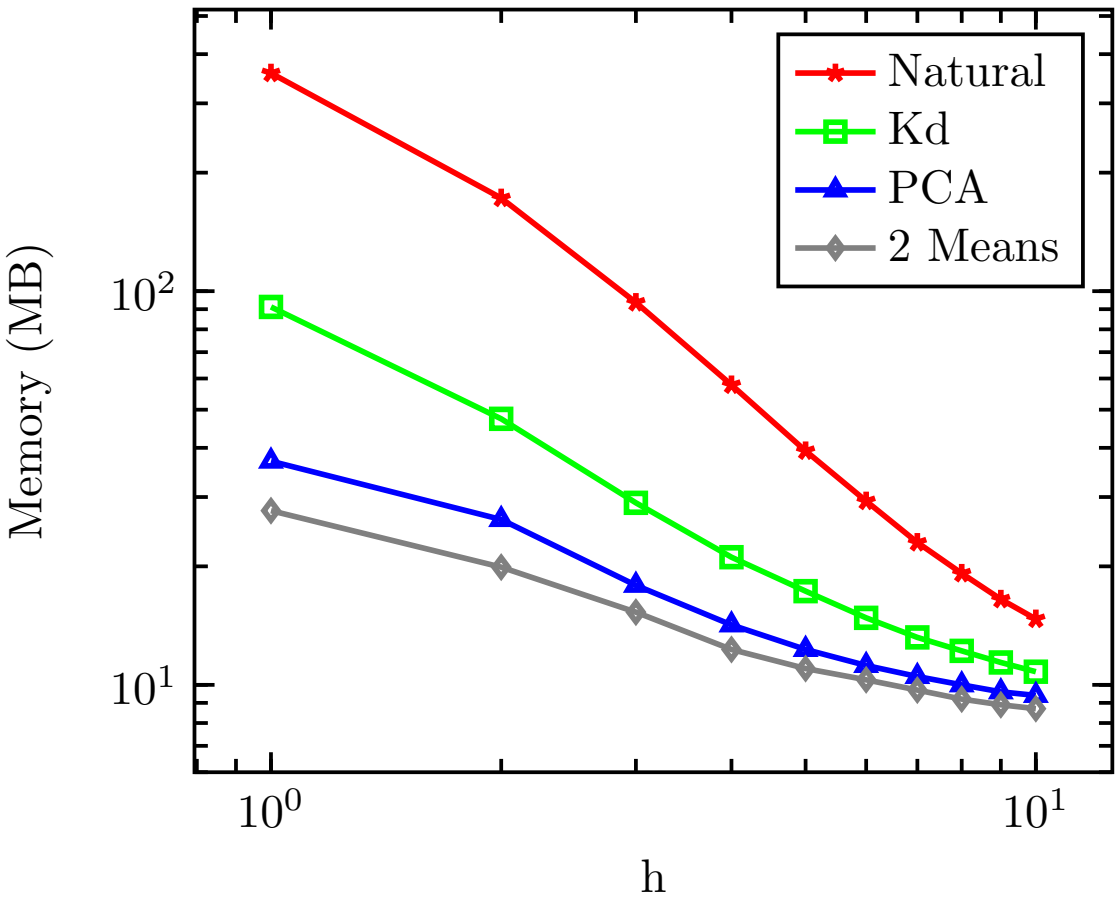
3. Recursive two means - continued
   - Best in terms of memory achieved
   - Quite optimal in other measures (rank, compression quality)
   - Variation in rank
   - Optimal *min cluster size* is ~ 100 heuristically, regardless of data

Datasets – UCI Machine Learning Repository

https://archive.ics.uci.edu/ml/index.php

- SUSY, HEPMASS: high-energy Physics
- COVTYPE: forest type, cartographic variables
- PEN, LETTER, MNIST: handwritten digits and letters
- GAS: concentration levels of gases

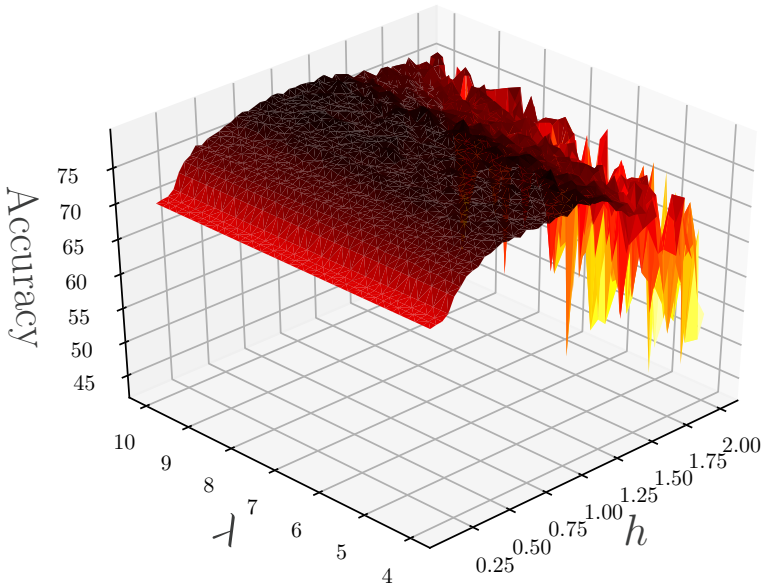
Clustering effect



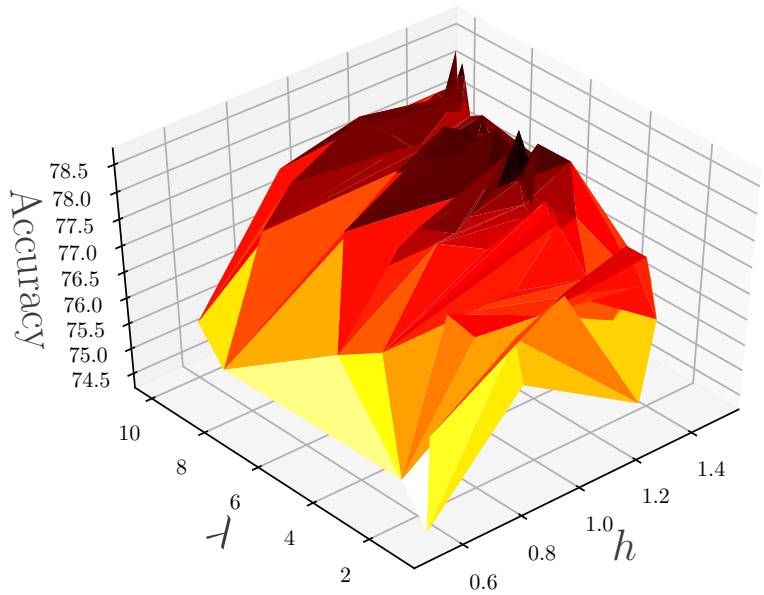
Hyperparameter tuning (h, λ)

$\lambda I + K$, Gaussian kernel $K(i,j) = \exp\left(-\frac{\|x_i - x_j\|_2^2}{2h^2}\right)$

- When λ changes, only need to update diagonal of HSS
- When h changes, need to recompress HSS $\leftarrow$ expensive



Grid Search 128^2 runs



OpenTuner optimization 100 runs

opentuner.org

Low rank compression via randomized sampling (RS)

Approximate range of A:

1. Pick random matrix $\Omega_{n\times(k+p)}$, k target rank, p small, e.g. 10
2. Sample matrix $S = A\Omega$, with slight oversampling p
3. Compute $Q =$ ON-basis(S) via RRQR

Accuracy: [Halko, Martinsson, Tropp, '11]

- On average: $E(\|A - QQ^*A\|) = \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min\{m,n\}}\right)\sigma_{k+1}$
- Probabilistic bound: with *probability* $\geq 1 - 3\cdot 10^{-p}$,
  $\|A - QQ^*A\| \leq [1 + 9\sqrt{k+p}\sqrt{\min\{m,n\}}]\,\sigma_{k+1}$

(in 2-norm)

Low rank compression via randomized sampling (RS)

Approximate range of A:

1. Pick random matrix $\Omega_{n\times(k+p)}$, k target rank, p small, e.g. 10
2. Sample matrix $S = A\Omega$, with slight oversampling p
3. Compute $Q =$ ON-basis(S) via RRQR

Accuracy: [Halko, Martinsson, Tropp, '11]

- On average: $E(\|A - QQ^*A\|) = \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min\{m,n\}}\right)\sigma_{k+1}$
- Probabilistic bound: with *probability* $\geq 1 - 3\cdot 10^{-p}$,
  $\|A - QQ^*A\| \leq [1 + 9\sqrt{k+p}\sqrt{\min\{m,n\}}]\,\sigma_{k+1}$

(in 2-norm)

Benefits:

- Matrix-free, only need matvec
- When embedded in sparse frontal solver, simplifies "extend-add"

Time bottleneck is in sampling

SUSY: $n = 4.5M, d = 8, \lambda = 4, h = 1$
COVTYPE: $n = 0.5M, d = 54, \lambda = 1, h = 1$

	SUSY		COVTYPE	
Cores	32	512	32	512
$\mathcal{H}$ construction	173.7	18.3	36.5	32.2
HSS construction	3344.4	726.7	432.3	239.7
$\longrightarrow$ Sampling	2993.5	662.1	305.2	178.4
$\longrightarrow$ Other	350.9	64.6	127.1	61.3
ULV Factorization	14.2	3.3	26.5	4.6
Solve	0.5	0.3	0.5	0.4

What exactly takes so long?

Approximate bases of submatrices

As part of HSS construction we need to approximate bases of all rectangular (off-diagonal) parts, and later of their unions:

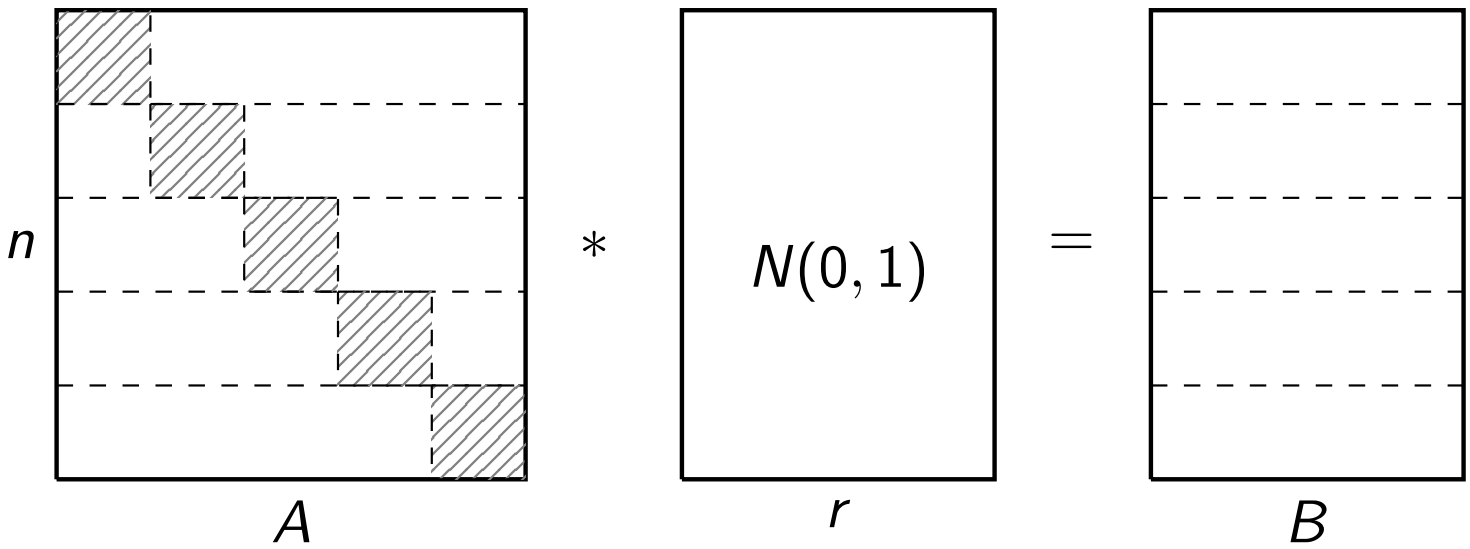


Figure: respective parts of B contain approx bases of parts of A

Slow way: via Gaussian projection (complexity $O(n^2 r)$ with $r =$ num rank; $\mathcal{H}$-matrix is constructed to speed up matrix-matrix multiplication)

Approximate bases of submatrices - 2

As part of HSS construction we need to approximate bases of all rectangular (off-diagonal) parts, and later of their unions:

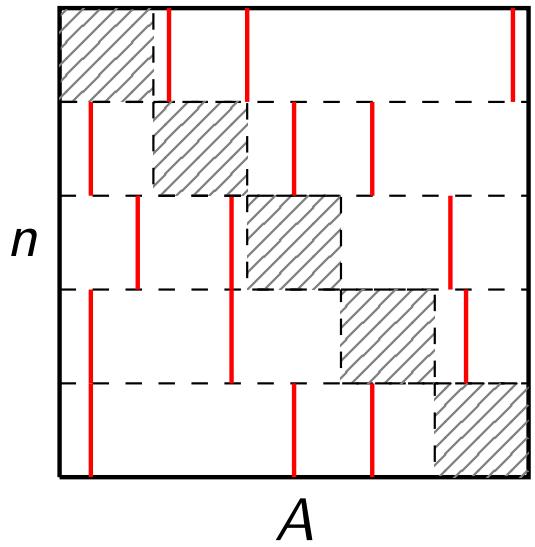


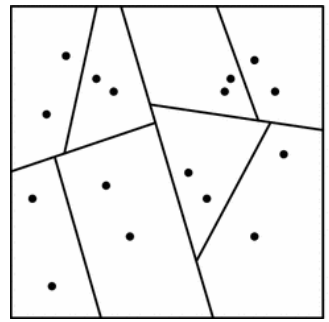
Figure: approx bases consist of red columns of A

New way: find column basis (r most important columns - how?)

Neighbor-based importance column sampling

Idea: Important entries in each row (for kernel matrix) = neighbors of the respective data point

- Find r approximate neighbors of each data point in $O(dn \log^2 n)$ time (see Freund & Dasgupta about randomized projection trees)
- Use these column indices to define important columns in each rectangular subblock (see Biros et al about GOFMM algorithm)
- Drop some *less important* columns when passing to higher levels, this keeps complexity low



(projection tree picture is taken from Dasgupta & Sinha s paper)

(Preliminary) experimental results

	Gaussian sampling			Column based sampling			Accuracy
	solution quality	time (s)	target rank	solution quality	time (s)	target rank	
COVTYPE, 10K	0.0014	43	1300 -> 1364	0.0014	12	1300	97.10%
SUSY, 10K	0.017	107	2000 -> 2200	0.03	53	2000	80.20%

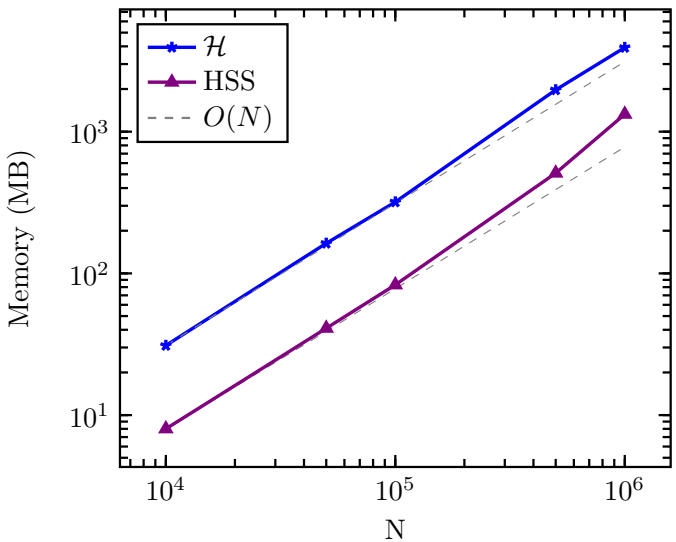
Figure: tolerance 0.01, h and λ are chosen by cross-validation

Next steps:

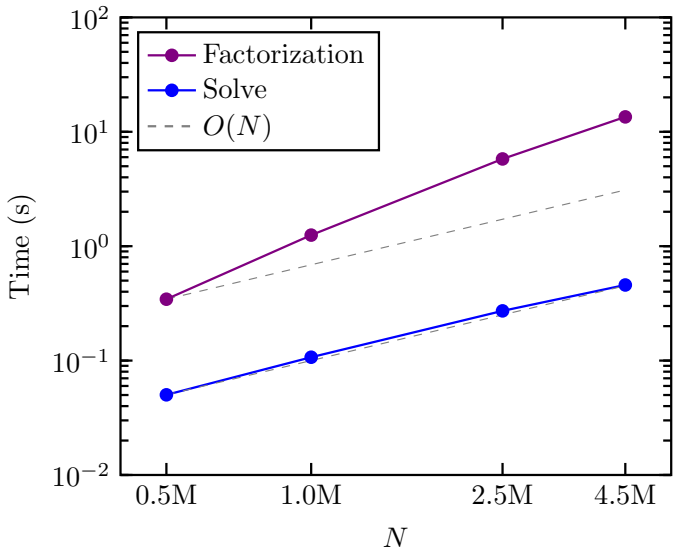
- Make target rank adaptive
- Test on large datasets e.g. $O(10^6)$
- Accommodate more general kernels (using neighbors in kernel-provided metric instead of Euclidean)

Algorithm scalability

SUSY dataset, n varies to 4.5M



Memory



Run time

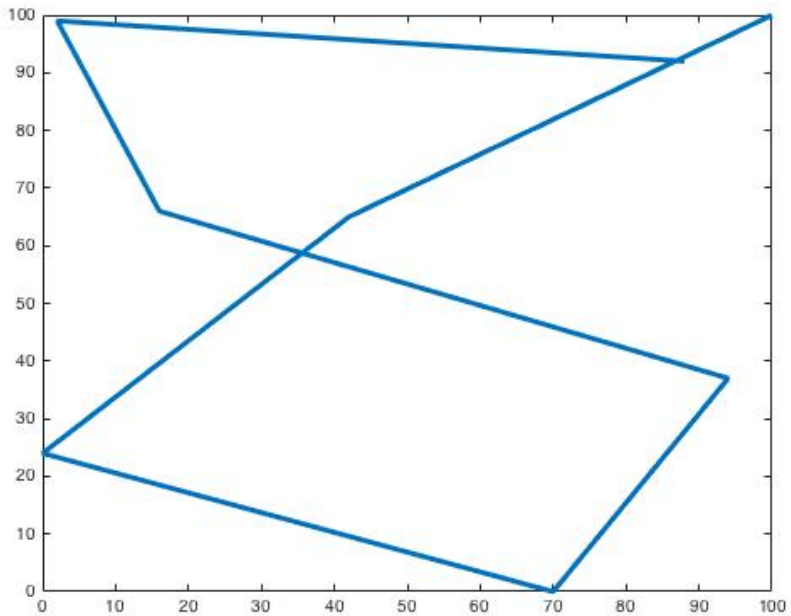
STRUMPACK – STRUctured Matrices PACKage

http://portal.nersc.gov/project/sparse/strumpack/

- Two components:
  - Dense – applicable to Toeplitz, Cauchy, BEM, integral equations, etc.
  - Sparse – aim at matrices discretized from PDEs.
- Open source on Github, BSD license.
- C++, hybrid MPI + OpenMP implementation
- Real & complex datatypes, single & double precision (via template), and 64-bit indexing.
- Input interfaces:
  - Dense matrix in standard format.
  - Matrix-free, with query function to return selected entries.
  - Sparse matrix in CSR format.
- Can take user input: cluster tree & block partitioning.
- Functions:
  - HSS construction, HSS-vector product, ULV factorization, Solution.
- Available from PETSc, MFEM.
- **Extensible to include other data-sparse formats.**

PENDIGITS example

- 7494 points in the training set
- 3498 points in the test set
- Every data point $\{x_1, y_1, x_2, y_2, \ldots, x_8, y_8\} \in \mathbb{R}^{16}$ contains 8 coordinates (x_i, y_i) regularly spaced in arc length along a written digit



PENDIGITS challenge: multi-class prediction

1. We had 10 classifiers of the type "two or not two".
   - Need to assign "most likely class". So, in step 4

$$\text{sign}\langle \mathbf{v}, K' \rangle \mapsto \text{argmax}_{t=1\ldots10} \langle \mathbf{v}_t, K' \rangle$$

   gives the most confidently predicted number.

PENDIGITS - parameters and results

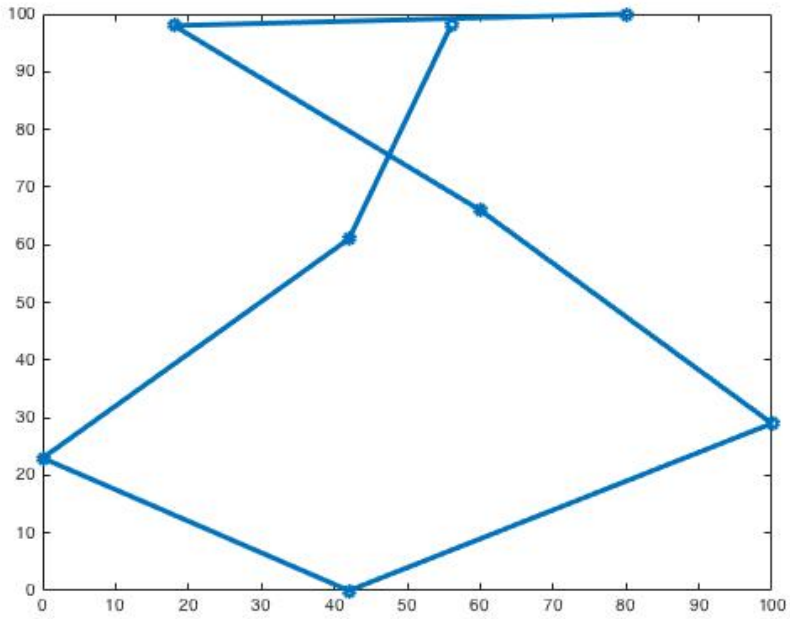
Parameters:

- tolerance = 1e-03
- regression parameters: $\lambda = 10, h = 5$
- number of Gaussian samples = 500
- tree type = top down recursive 2-means tree

Results:

- Memory used : 4.630 MB, 1.03% of dense
- Max rank in compression: 52
- Compression quality: 5e-06
- Accuracy = 0.9774 (79 points were misclassified out of 3498)

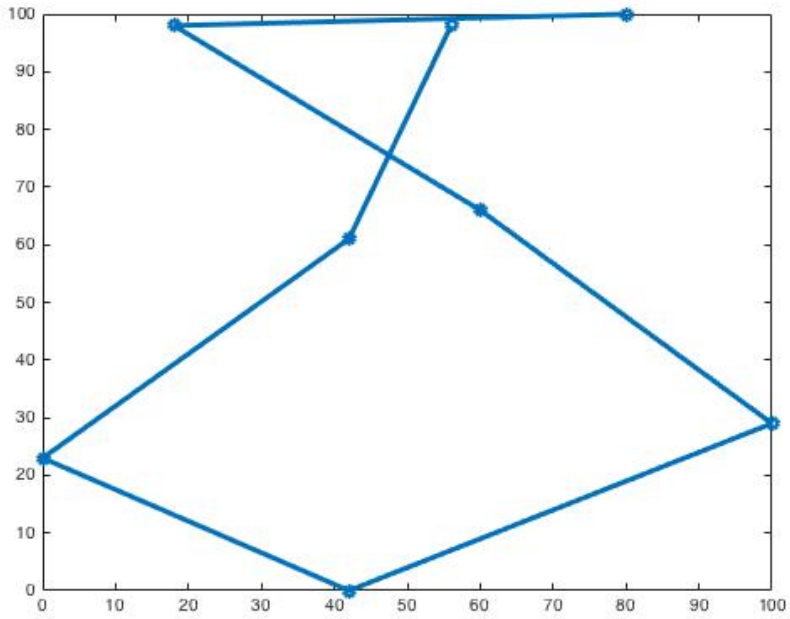
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

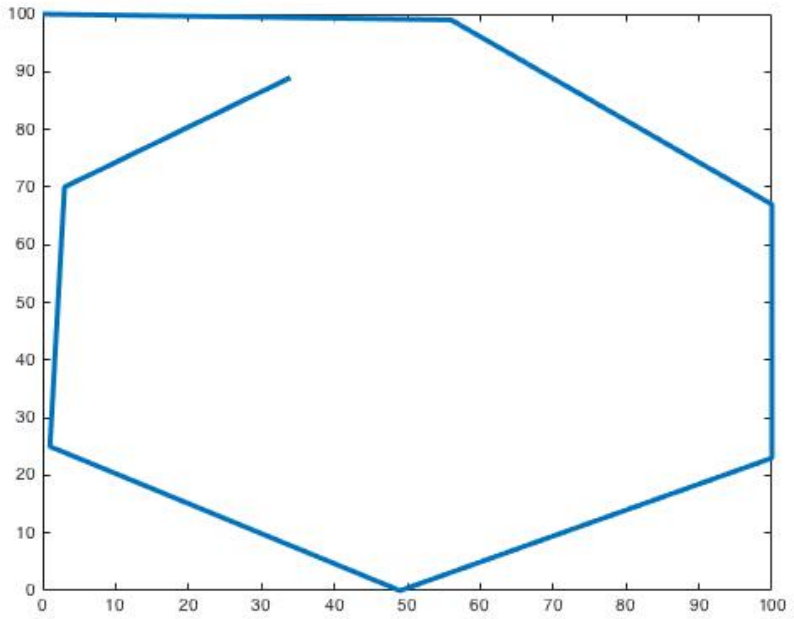
PENDIGITS - guess the number:)



8

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

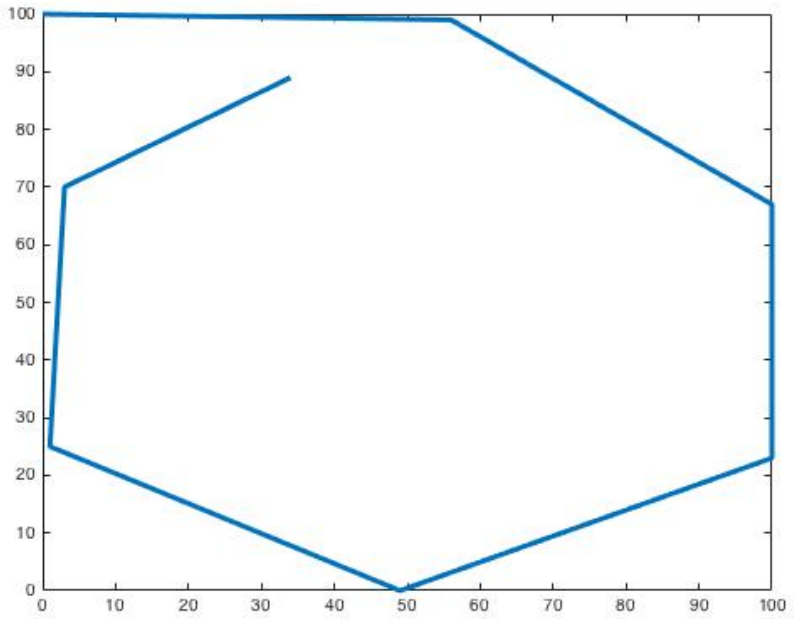
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

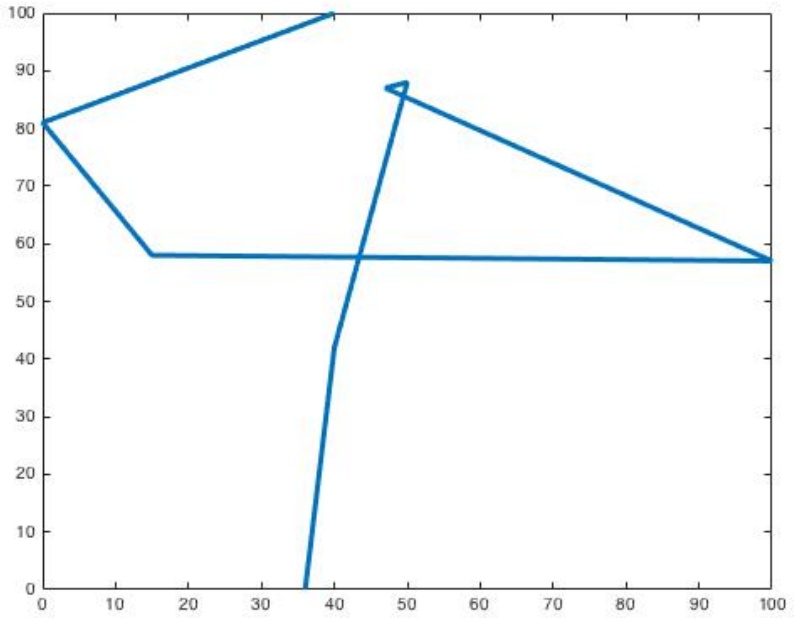
PENDIGITS - guess the number:)



0

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

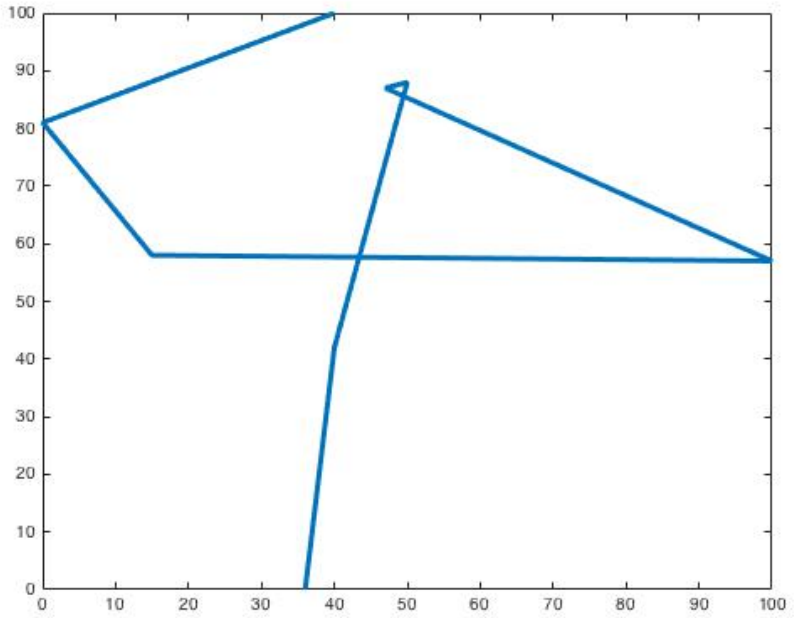
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

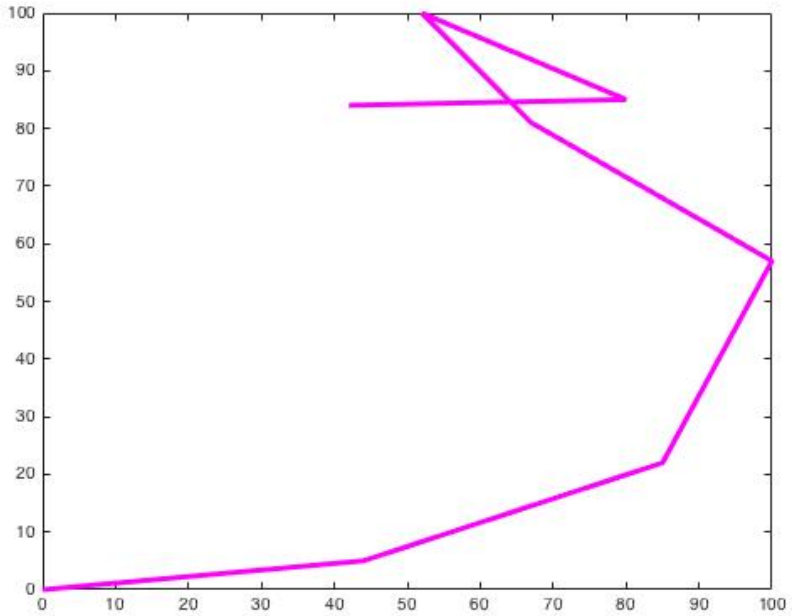
PENDIGITS - guess the number:)



4

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

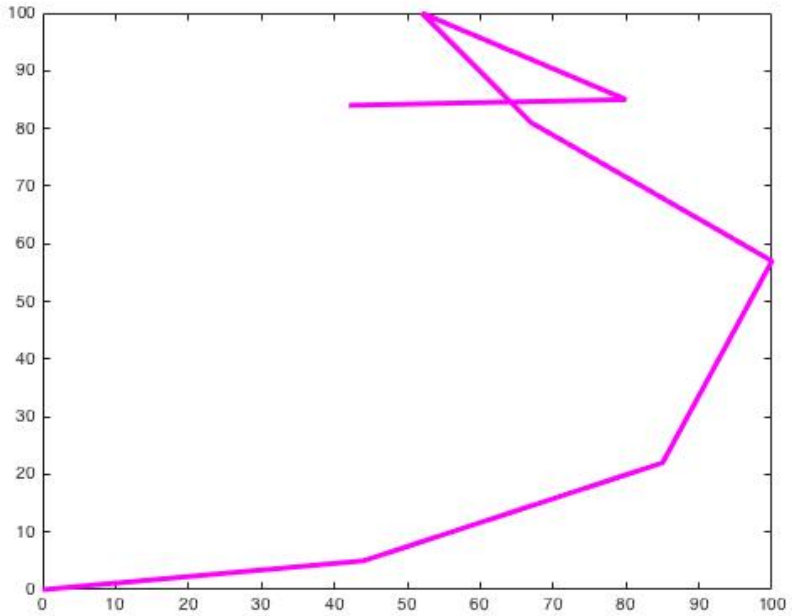
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

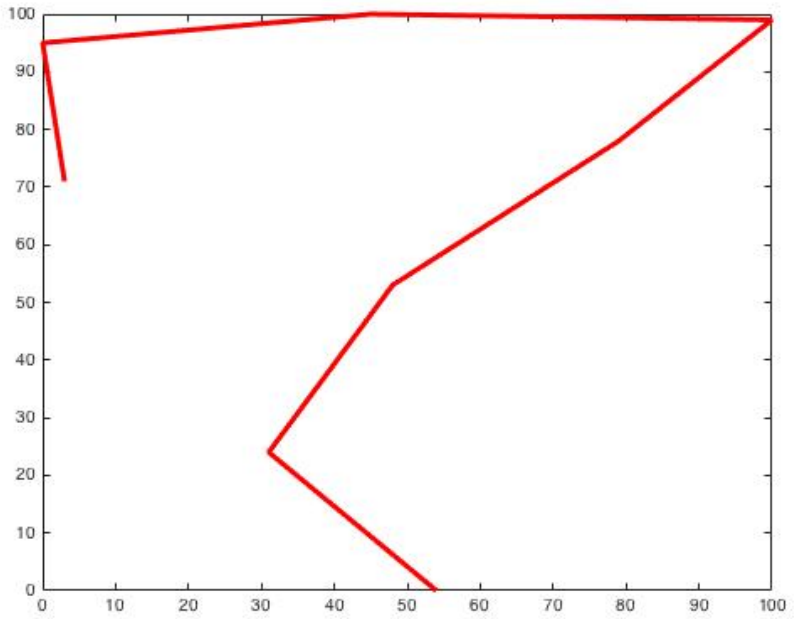
PENDIGITS - guess the number:)



9

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

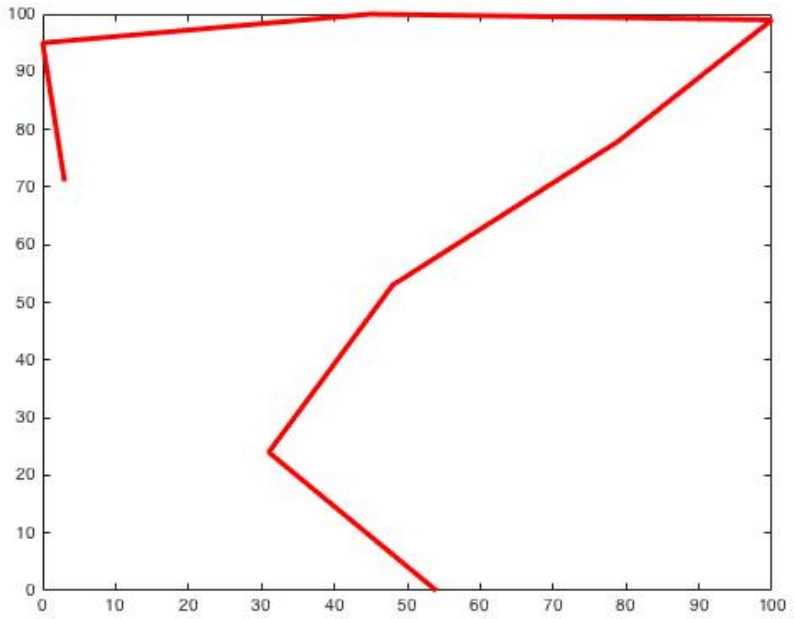
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

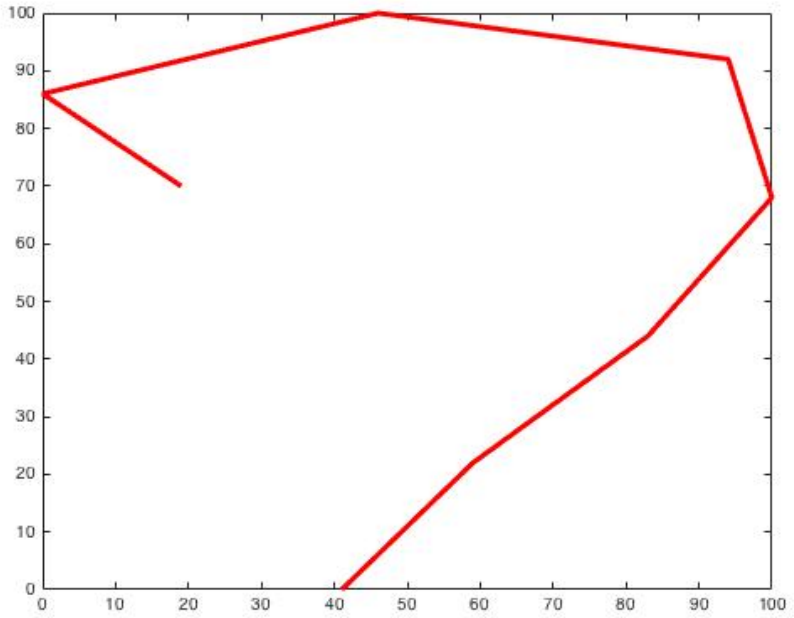
PENDIGITS - guess the number:)



7 (predicted 3)

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

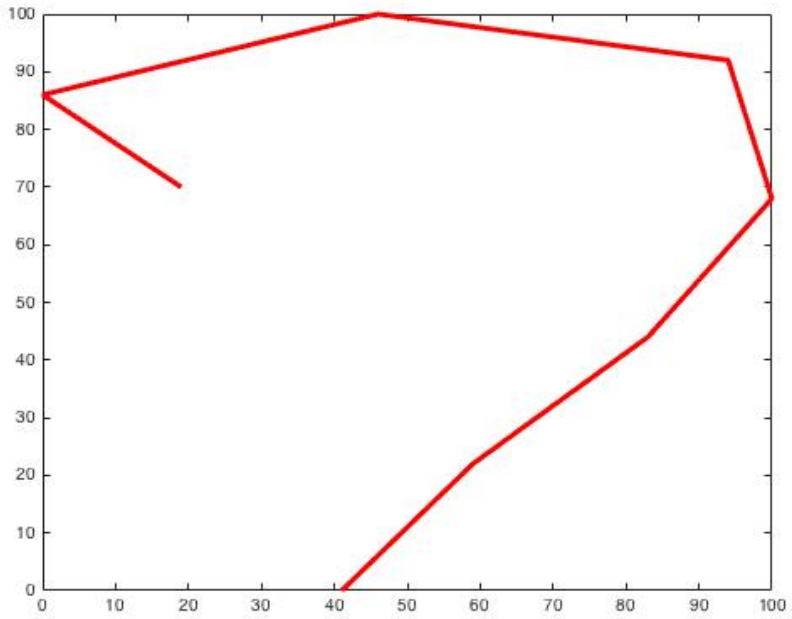
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

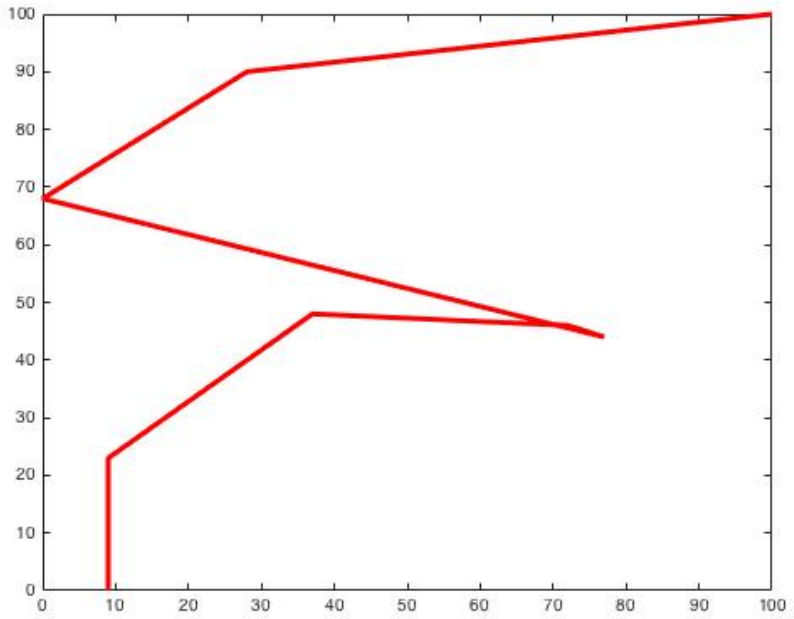
PENDIGITS - guess the number:)



9 (predicted 7)

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

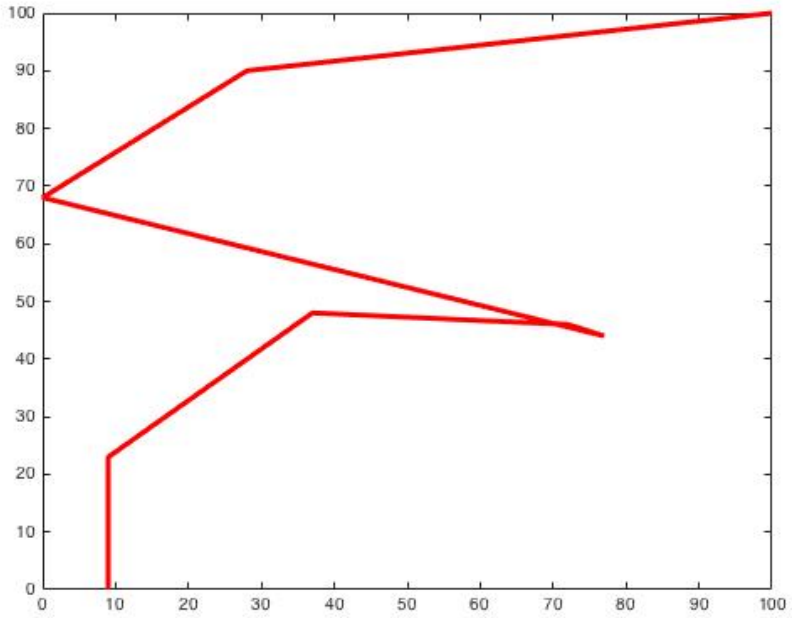
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

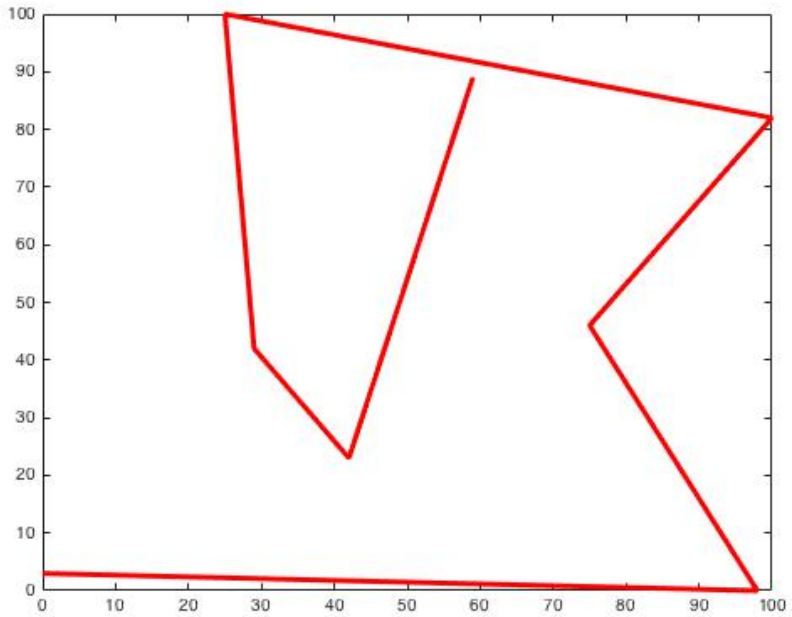
PENDIGITS - guess the number:)



4 (predicted 5)

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

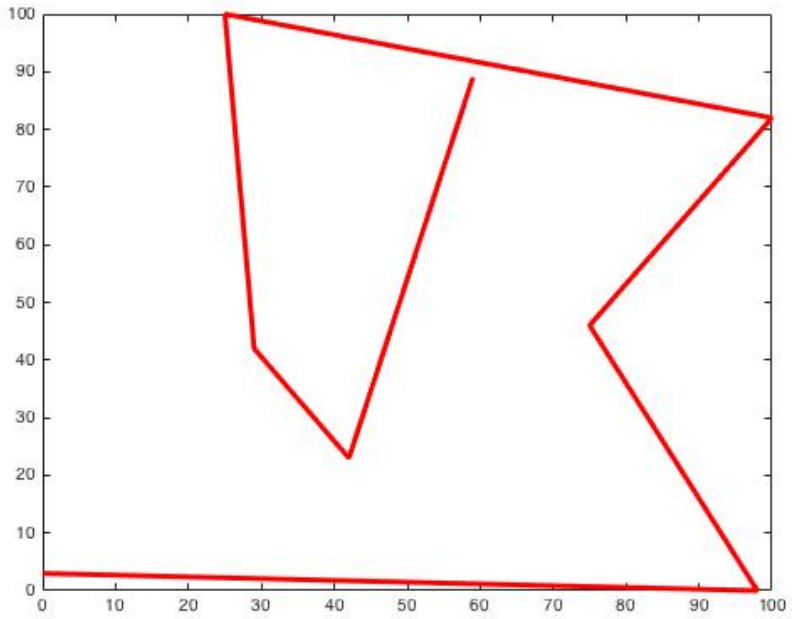
PENDIGITS - guess the number:)



?

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

PENDIGITS - guess the number:)



3 (predicted 9)

Blue numbers - predicted correctly, magenta - correct for some experiments, red - errors

Conclusions

- Hierarchical matrices provide an optimal strategy to perform kernel ridge regression.
- This approach proves most beneficial for datasets with low to moderate dimension.
- Interpretable method, based on linear systems.

Acknowledgement

This research was supported by the Exascale Computing Project (http://www.exascaleproject.org), a joint project of the U.S. Department of Energys Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nations exascale computing imperative.
Project Number: 17-SC-20-SC

THANK YOU!