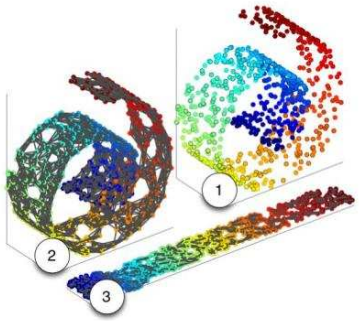
Permutation and b-Matching in Machine Learning

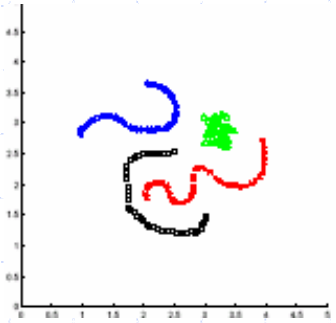
Tony Jebara
Machine Learning Lab
Columbia University

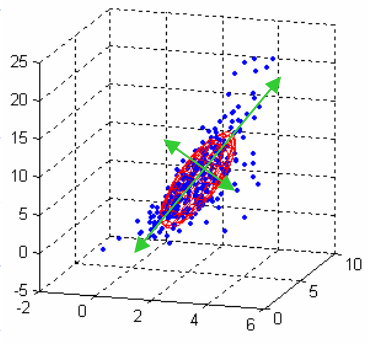
joint work with B. Huang, R. Kondor, B. Shaw, V. Shchogolev, P. Shivaswamy

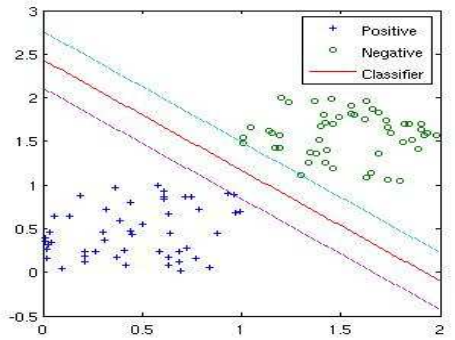
Machine Learning & Optimization

- Statistical, Data-Driven Computational Models
- Machine Learning Problems:









Embedding **Clustering** **Compression** **Classification**

- Learn by writing a cost function over parameters, labels, or embedding based on likelihood, classification margin, dimensionality reduction, clustering cost and *optimize* via:

Linear Programming
Semidefinite Programming
Quadratic Programming
Gradient/Variational Methods

Matching and B-Matching

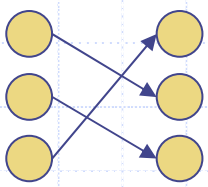
- We explore these 2 combinatorial optimization algorithms from graph theory and find ways to use them in learning.
- Maximum Weight Matching = Linear Assignment Problem
  Given weight matrix, find permutation matrix. $O(N^3)$

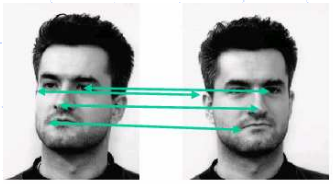
husband

wife $\begin{bmatrix} \$1 & \$6 & \$3 \\ \$4 & \$2 & \$4 \\ \$4 & \$2 & \$5 \end{bmatrix}$ $\rightarrow$ **Kuhn-Munkres Hungarian Algorithm** $\rightarrow$ $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$\max_P tr\left(P^T A\right)$$

$$\sum_i P_{ij} = \sum_j P_{ij} = 1,\ P_{ij} \in \{0,1\}$$

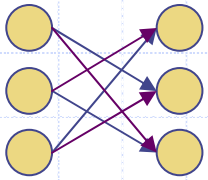




- B-Matching generalizes to multi-matchings (Mormon). $O(bN^3)$

$$\max_P tr\left(P^T A\right)$$

$$\sum_i P_{ij} = \sum_j P_{ij} = b,\ P_{ij} \in \{0,1\}$$



Matching and B-Matching

- Balanced versions of nearest neighbor & k-nearest-neighbor

$$A = \begin{bmatrix} 27 & 89 & 6 & 43 & 21 & 79 \\ 25 & 20 & 99 & 23 & 38 & 6 \\ 88 & 30 & 58 & 58 & 78 & 60 \\ 74 & 66 & 42 & 76 & 68 & 5 \\ 14 & 28 & 52 & 53 & 46 & 42 \\ 1 & 47 & 33 & 64 & 57 & 30 \end{bmatrix}$$

$$\max_P tr\left(P^T A\right)$$

$$where\, P_{ij} \in \{0,1\}$$

Matchings $O\left(bN^3\right)$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\sum_i P_{ij} = \sum_j P_{ij} = 1$$

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\sum_i P_{ij} = \sum_j P_{ij} = 3$$

Neighbors $O\left(bN^2\right)$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\sum_j P_{ij} = 1$$

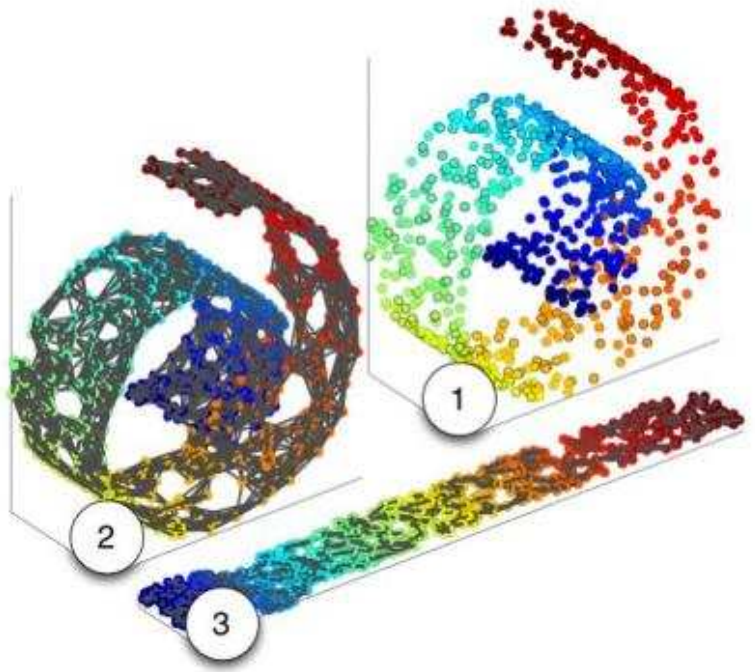
$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\sum_j P_{ij} = 3$$

B-Matched Embedding

- Replace k-nearest-neighbor in machine learning algorithms
- Example: Semidefinite Embedding (Weinberger & Saul)

1) Get matrix A by computing affinities between all pairs of points
2) Find k-nearest neighbors graph
3) Use SDP to find P.D. matrix K which preserves distances on graph yet is as stretched as possible. Eigen-decomposition of K finds embedding of points in low dimension that preserve distance structures in high dimensional data.

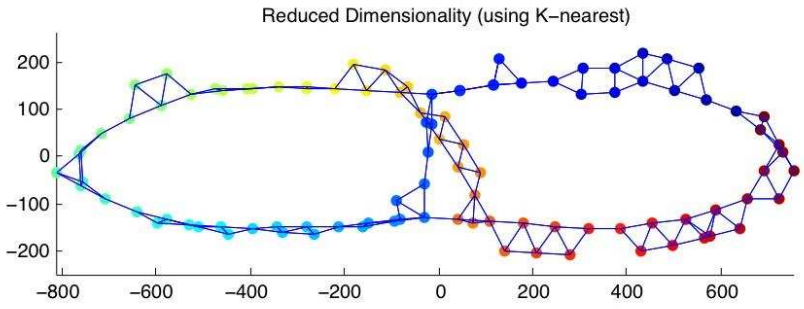


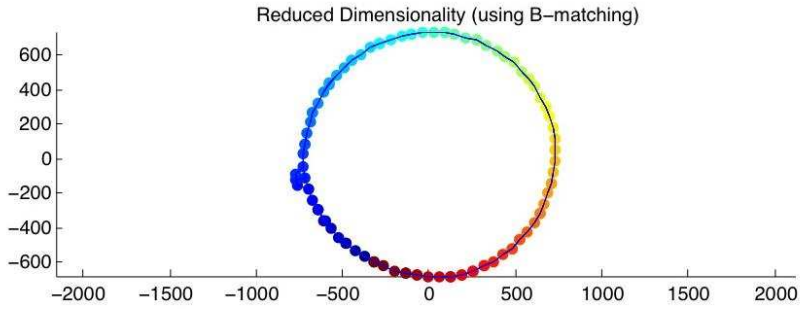
Maximize $\mathrm{Tr}(K)$ subject to $K \succeq 0$, $\sum_{ij} K_{ij} = 0$,
and $\forall\, i,j$ such that $h_{ij}=1$ or $[h^\top h]_{ij}=1$,
$K_{ii} + K_{jj} - K_{ij} - K_{ji} = A_{ii} + A_{jj} - A_{ij} - A_{ji}$.

B-Matched Embedding

- Visualization example: images of rotating tea pot.

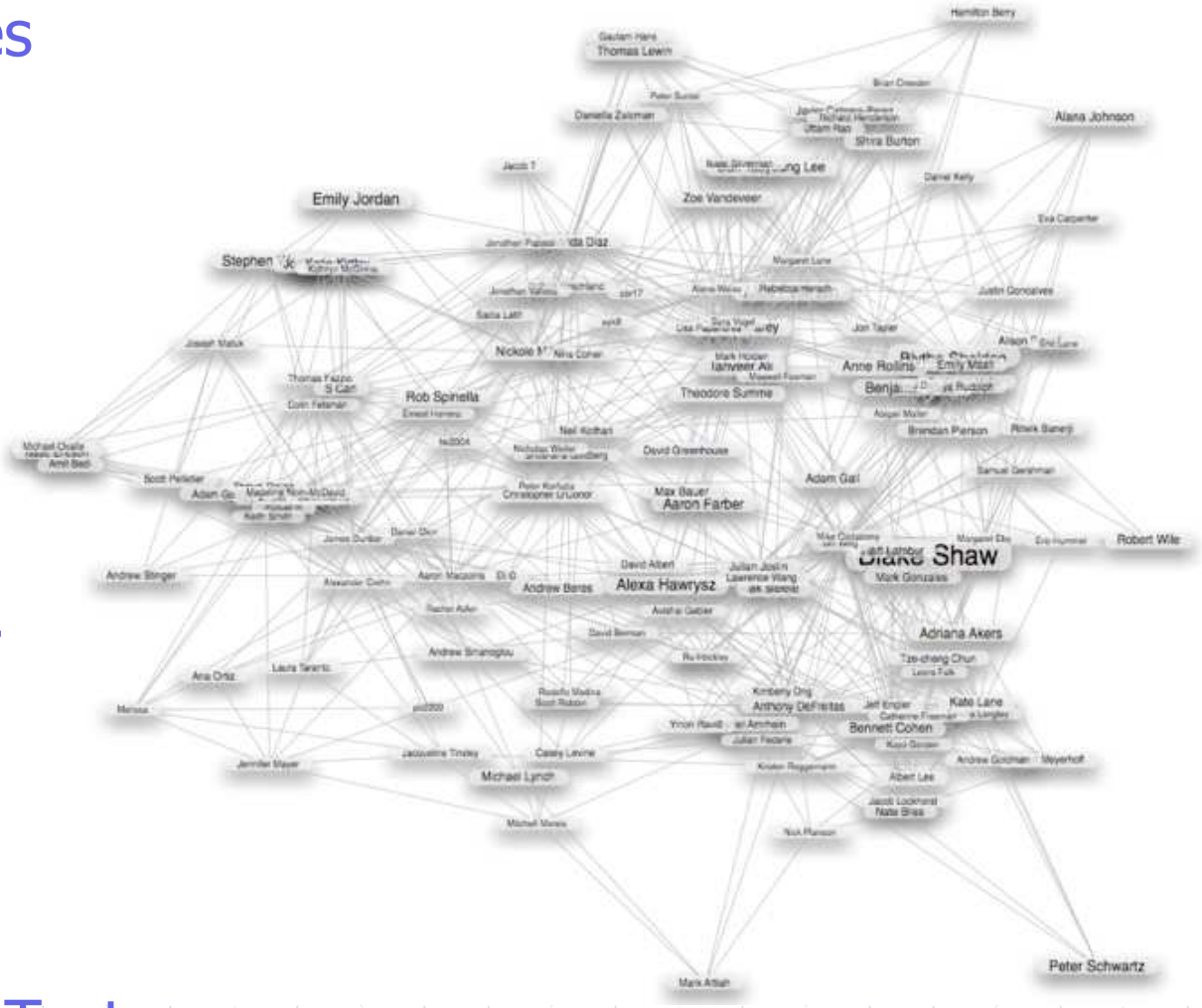
- Get affinity $A_{ij} = \exp(-||X_i - X_j||^2)$ between pairs of images
- Should get ring but noisy images confuse kNN. Greedily connects nearby images without balancing in-degree and out-degree. Get folded over ring even for various values of b (k)
- B-Matching gives clean ring for many values of b.





B-Matched Embedding

- Example: CUtunes
- 2d visualization
- 150 users
- 10,000 artists
- Use divergence between user's artist download frequency vector to find inter-user affinity matrix A
- Get b-matched social network
- Embed via SDE
- Visualize Musical Tastes



Spectral Clustering

- Many clustering algorithms exist in machine learning: kmeans, expectation-maximization, linkage, etc. But: local minima, make parametric cluster assumptions
- More general: converting data to graph, cluster via cut
- Given graph (V,E), weight matrix A, normalized cut is NP $ncut(B) = \frac{\sum_{i\in B, j\in V/B} A_{ij}}{\sum_{i\in B, j\in V} A_{ij}} + \frac{\sum_{i\in V/B, j\in B} A_{ij}}{\sum_{i\in V/B, j\in V} A_{ij}}$
- Spectral Graph Theory: relax discrete optimization over y node indicator vector to continuous:

$D_{ii} = \sum_j A_{ij} \quad d = \sum_i D_{ii} \quad d_B = \sum_{i\in B} D_{ii} \quad y(i) = \begin{cases} \sqrt{d_{V/B}/d_B d} & if\ i \in B \\ -\sqrt{d_{V/B}/d_B d} & if\ i \notin B \end{cases}$

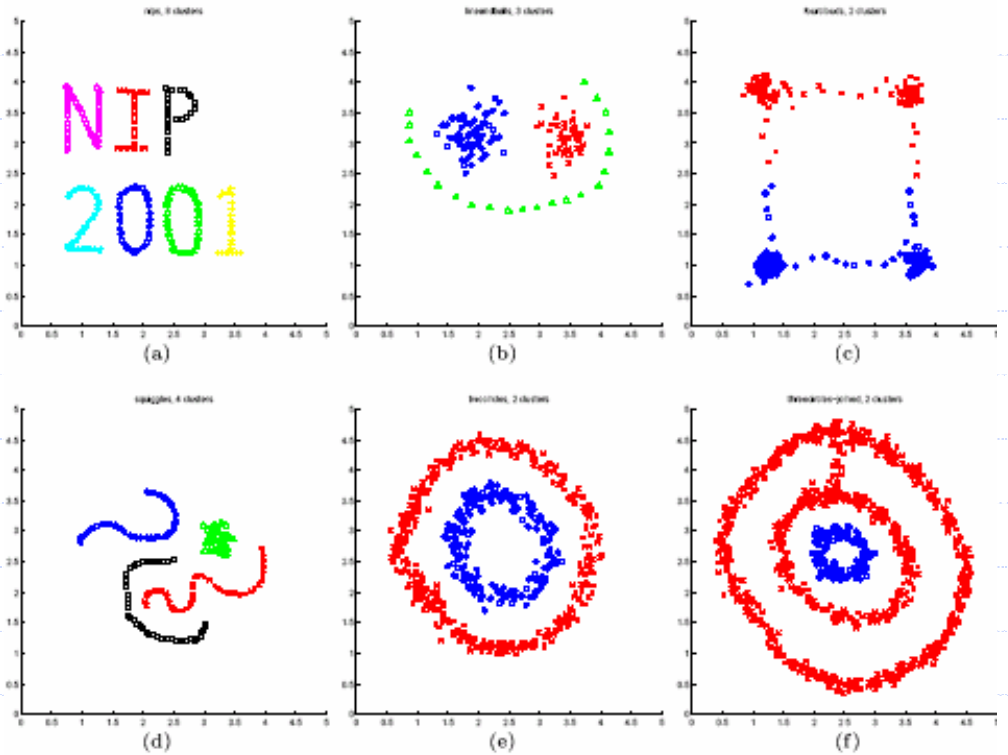
- Relaxed *global* y optimization (Shi & Malik):

$\min_y y^T (D-A) y \quad such\ that \quad y^T D y = 1 \quad and \quad y^T D e = 0$

- Solve for y as 2nd smallest eigenvector of: $(D-A)y = \lambda D y$

Spectral Clustering

- Currently many slight variants of spectral clustering
- Each has varying performance
- One theme is to stabilize the clustering (Ng, Weiss, Jordan) find eigenvectors of normalized Laplacian $L=D^{-1/2}AD^{-1/2}$
- Still imperfect but works if clusters are well separated



B-Matched Spectral Clustering

- Try to improve spectral clustering using B-Matching
- Assume w.l.og. two clusters of roughly equal size
- If we knew the labeling y = [+1 +1 +1 -1 -1 -1]
  the "ideal" affinity matrix A =

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

  in other words...

$$A = \tfrac{1}{2}\left(yy^T + 1\right)$$

  and spectral clustering and eigendecomposition is perfect
- The "ideal" affinity is a B-Matching with b=N/2
- Stabilize affinity by finding the closest B-Matching to it:

$$\min_P \|A - P\|^2 \ such\ that \sum_i P_{ij} = \sum_j P_{ij} = \tfrac{N}{2} \ and \ P_{ij} \in \{0,1\}$$

- Then, spectral cluster B-Matching or use it to prune A

$$A^{new} = P \quad or \quad A^{new} = P \circ A$$

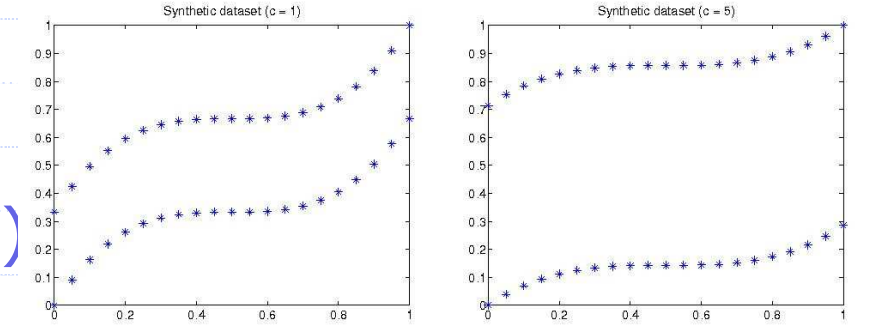
- Also, instead of B-Matching, can do kNN (lazy strawman).

B-Matched Spectral Clustering

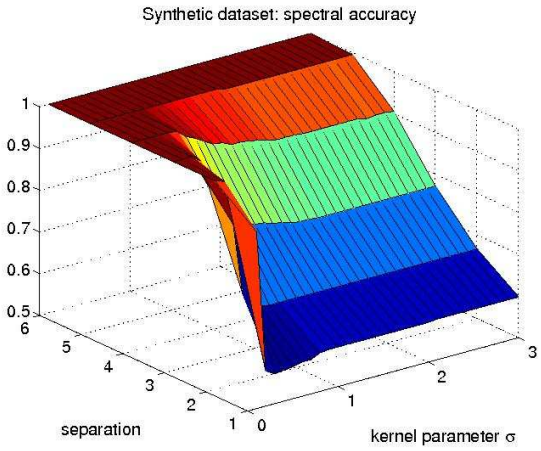
- Synthetic experiment
- Have 2 S-shaped clusters
- Explore different spreads
- Affinity $A_{ij}=\exp(-||X_i-X_j||^2/\sigma^2)$
- Do spectral clustering on

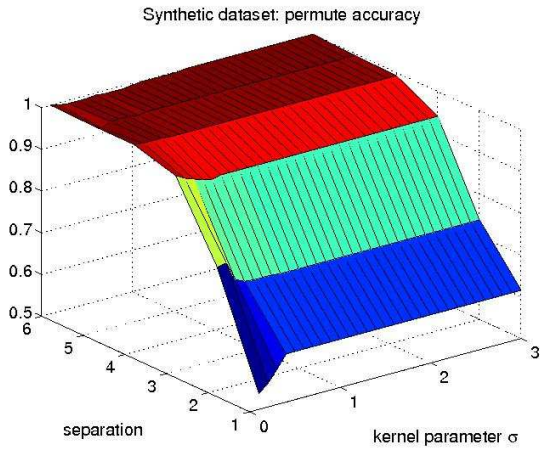
$$A \quad or \quad P \quad or \quad P \circ A$$

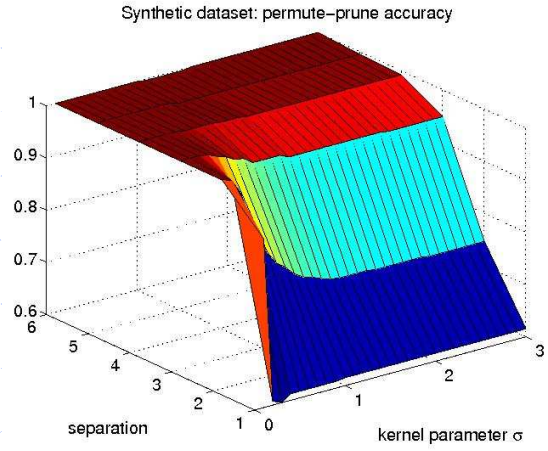
- Evaluate cluster labeling accuracy



(a) Curve separation $c = 1$. (b) Curve separation $c = 5$.

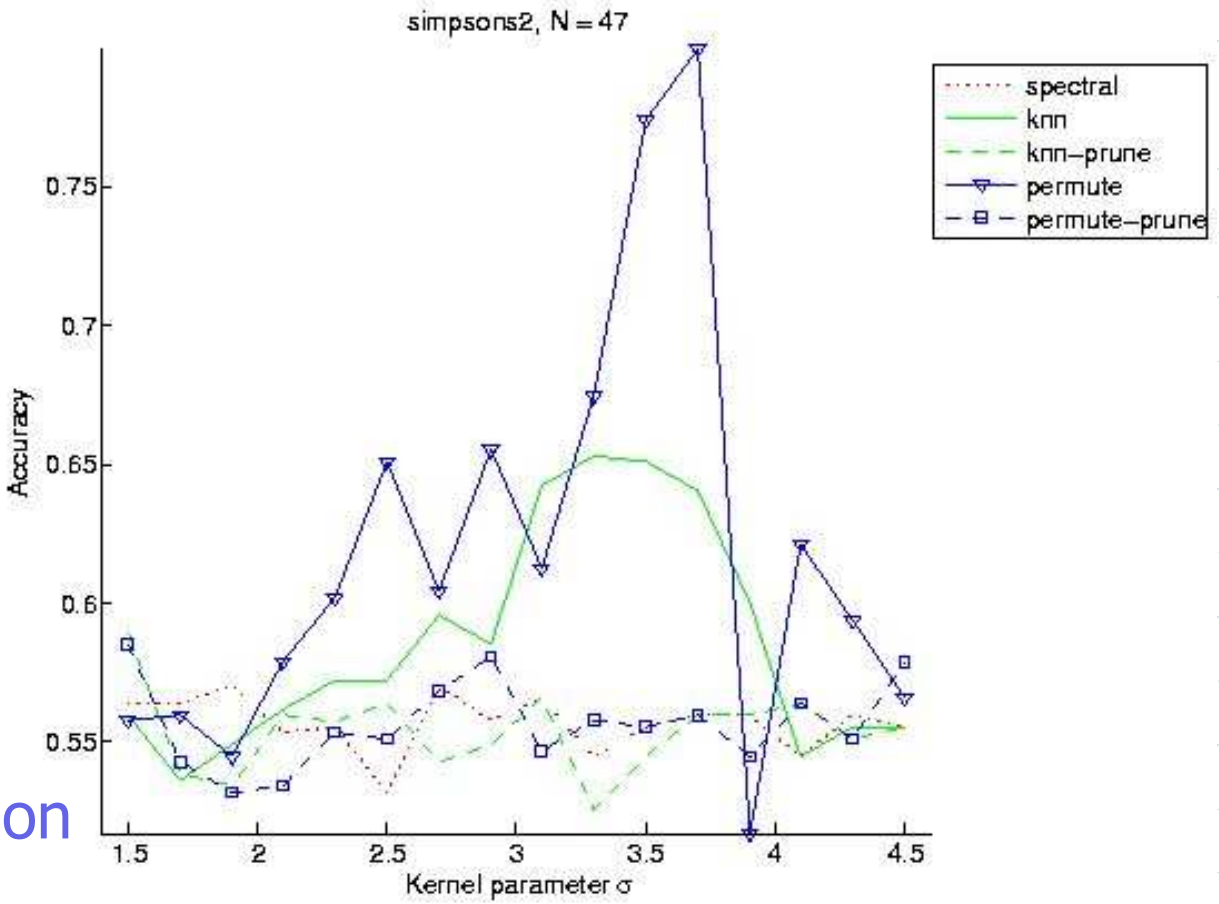






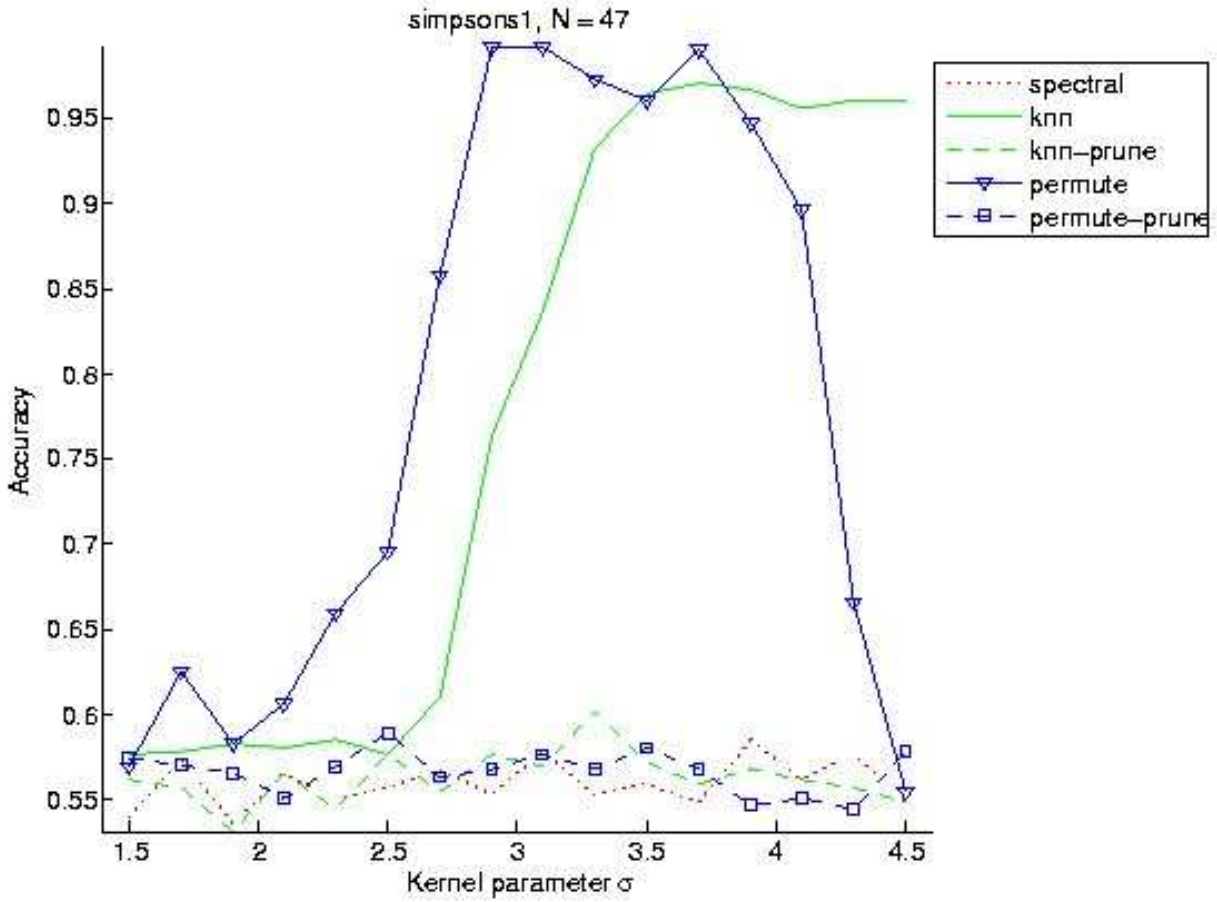
B-Matched Spectral Clustering

- Clustering images from real video with 2 scenes in it.
- Accuracy is how well we classify both scenes
- Evaluate also with kNN
- Only adjacent frames have high affinity
- BMatching does best since it boosts connection to far frames



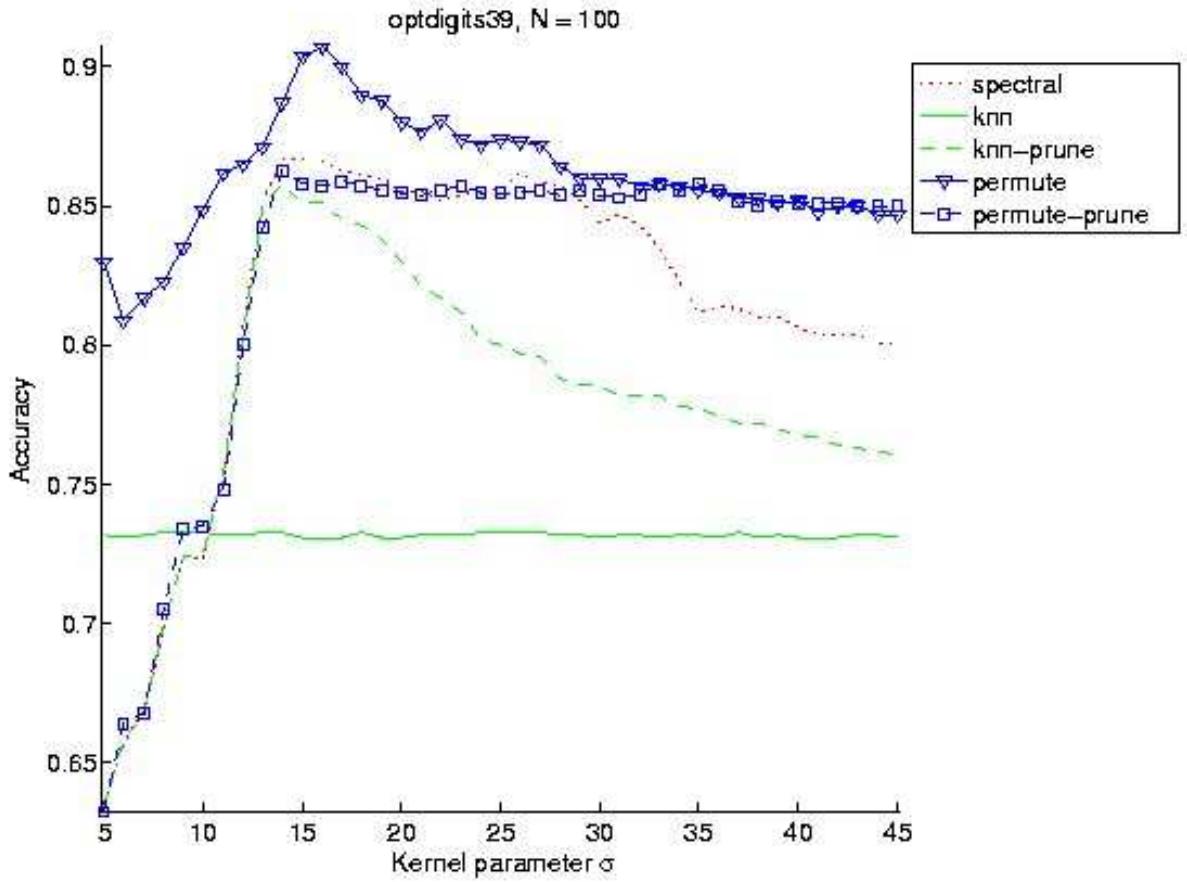
B-Matched Spectral Clustering

- Clustering images from same video but 2 other scenes



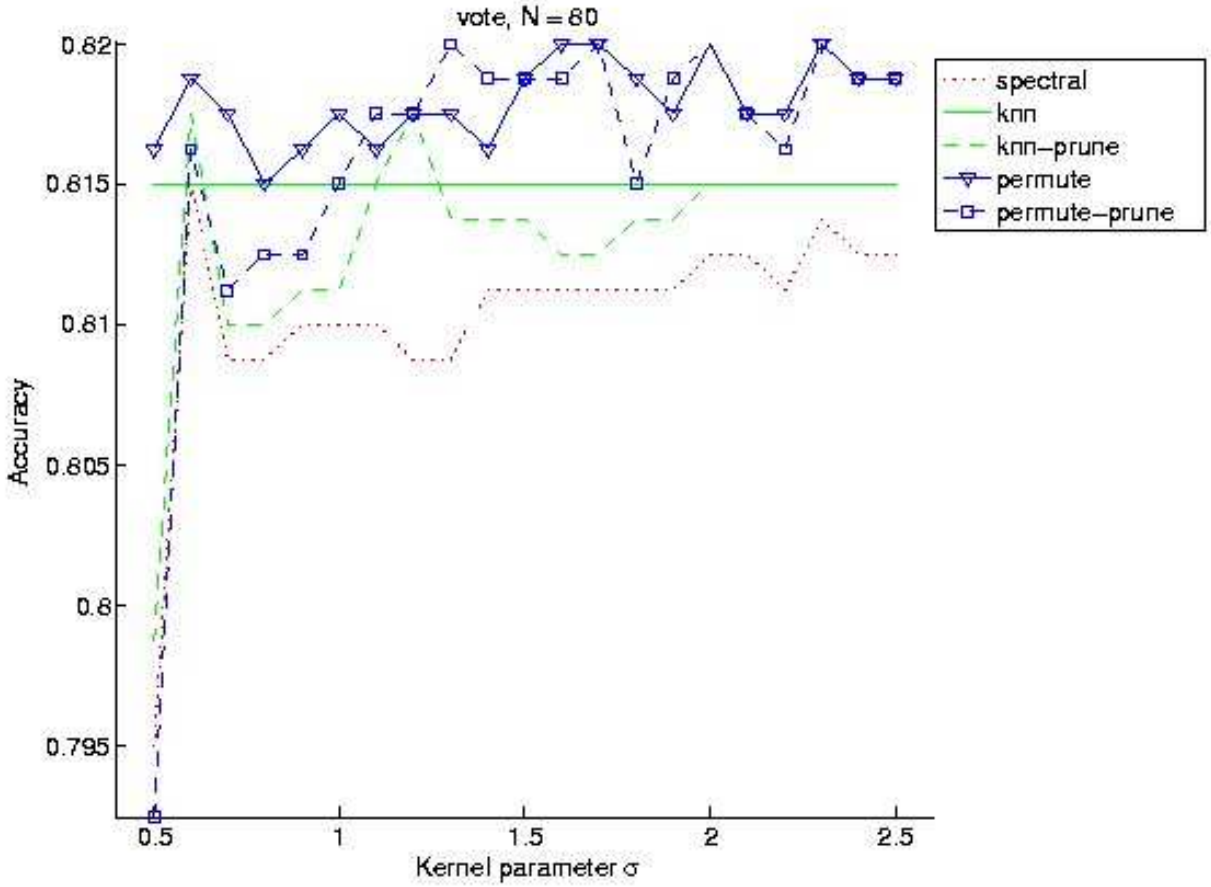
B-Matched Spectral Clustering

- Unlabeled classification via clustering of UCI Optdigits data



B-Matched Spectral Clustering

- Unlabeled classification via clustering of UCI Vote dataset

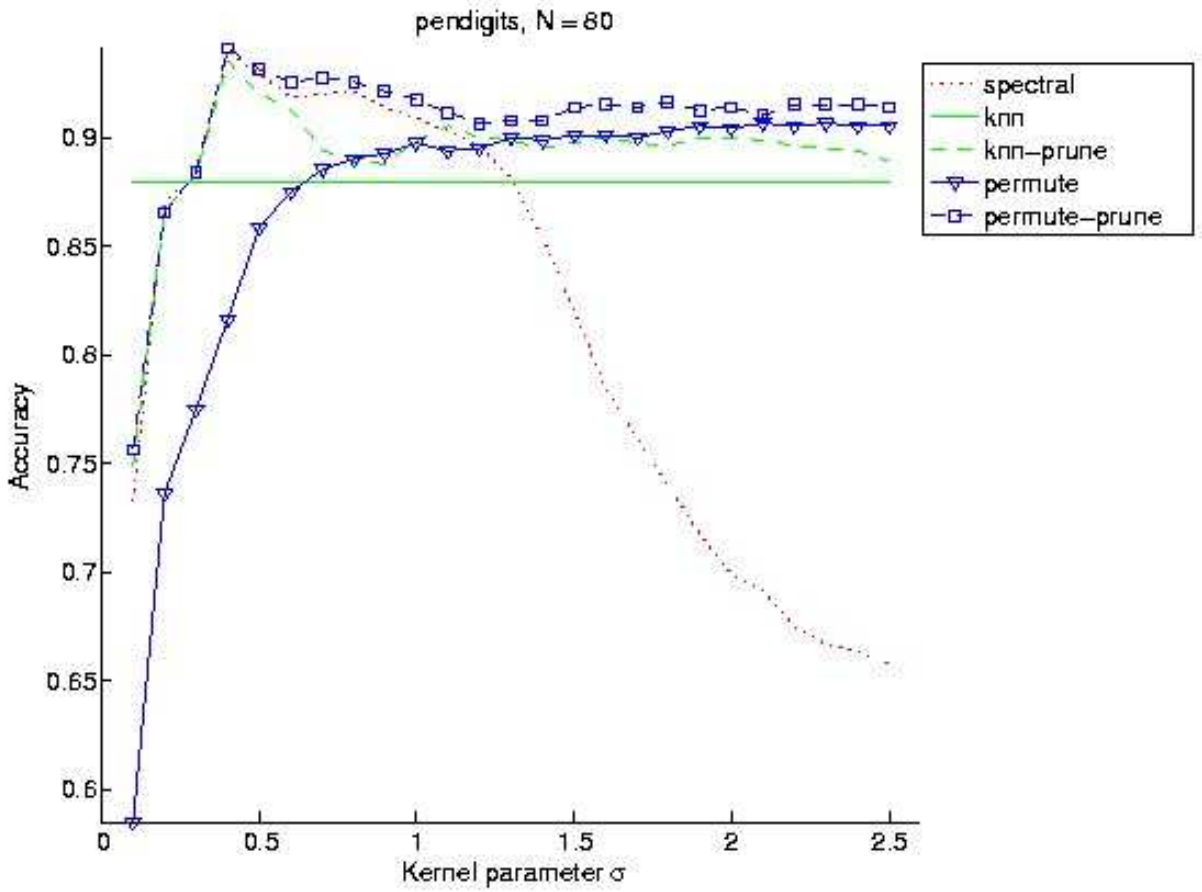


B-Matched Spectral Clustering

- Classification accuracy via clustering of UCI Pendigits data

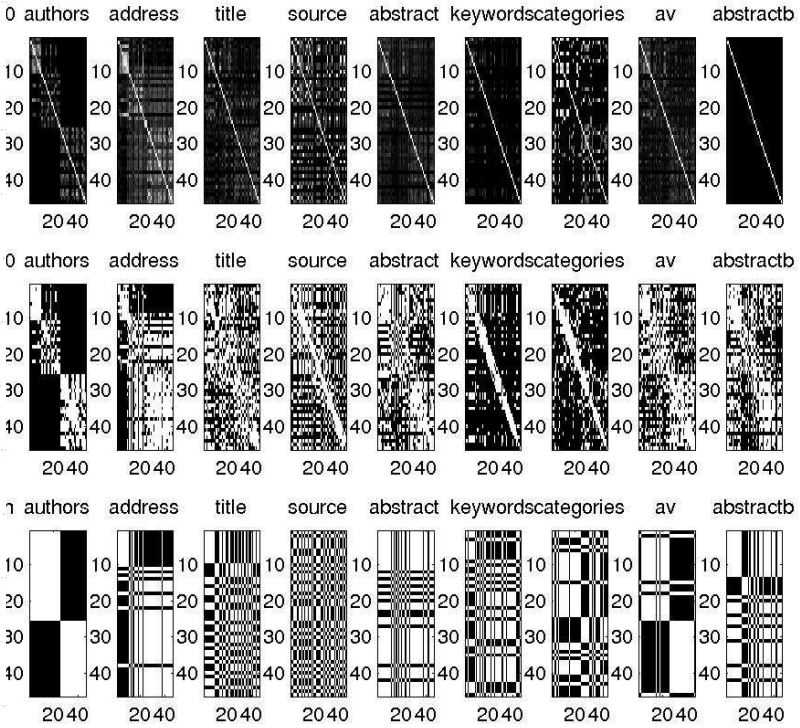
- Here, using the B-Matching to just prune A is better

  kNN always seems a little worse…



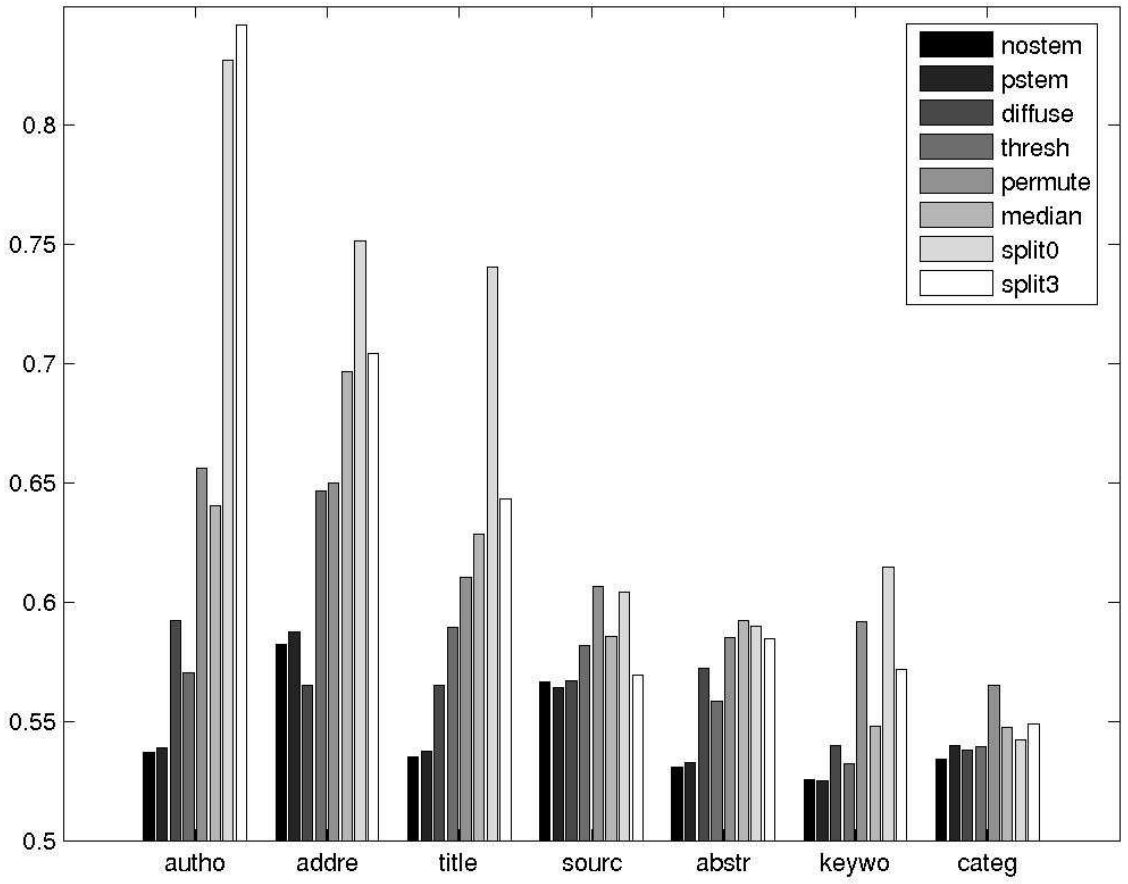
B-Matched Spectral

- Applied method to KDD 2005 Challenge. Predict authorship in anonymized BioBase pubs database
- Challenge gives many splits of N~100 documents
- For each split, find NxN matrix saying if documents i & j were authored by same person (1) or different person (0)
- Documents have 8 fields
- Compute affinity A for each field via text frequency kernel
- Find B-Matching P
- Get spectral clustering y and compute ½ (yy^T+1)



B-Matched Spectral Clustering

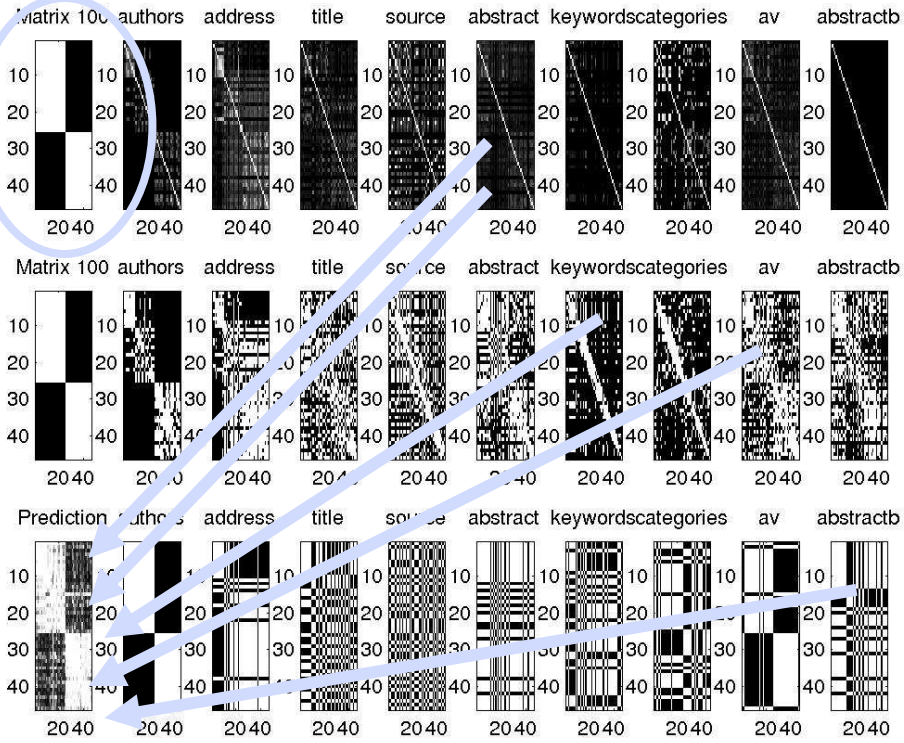
- Accuracy is evaluated using the labeled true same-author & not-same-author matrices
- Explored many processings of the A matrices. Best accuracy was by using the spectral clustered values of the P matrix found via B-Matching



B-Matched Spectral Clustering

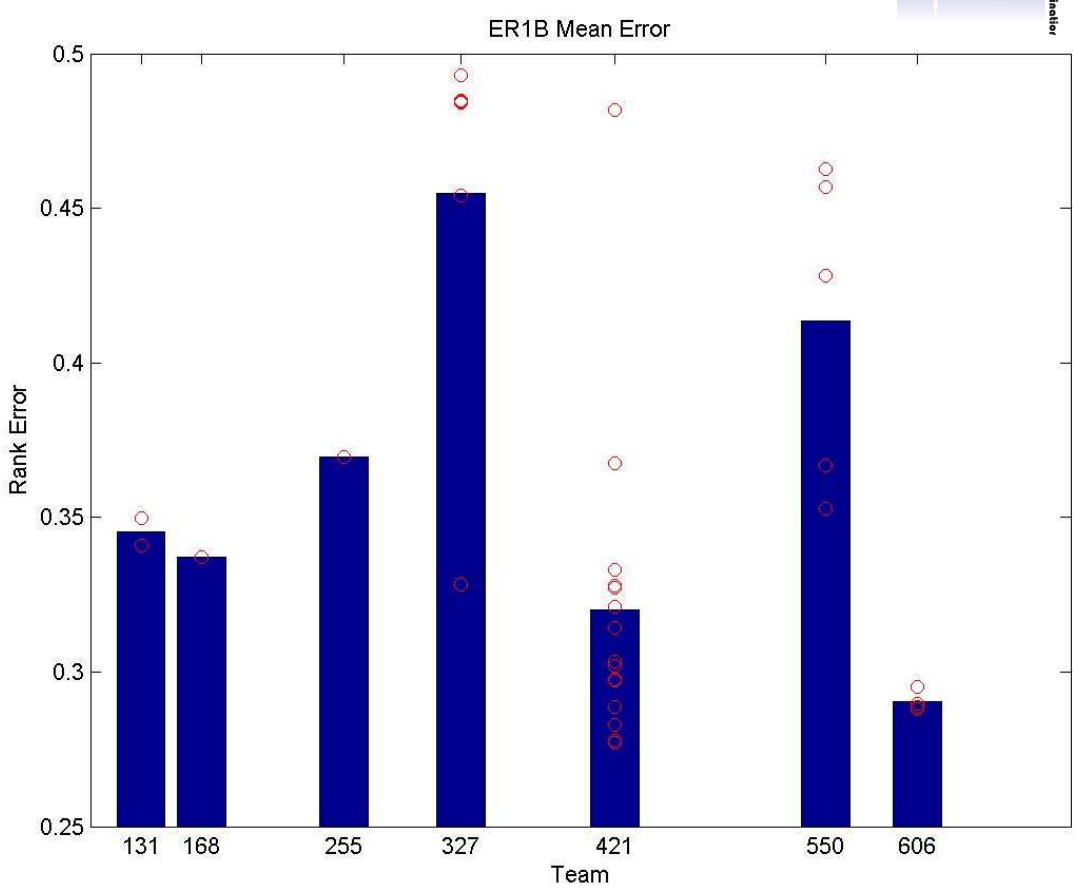
- Merge all the 3x8 matrices into a single hypothesis using an SVM and a quadratic kernel. SVM is trained on labeled data (same author, not same author matrices).

- For each split, we get a single matrix of same-author and not-same-author which was uploaded to KDD Challenge anonymously



B-Matched Spectral Clustering

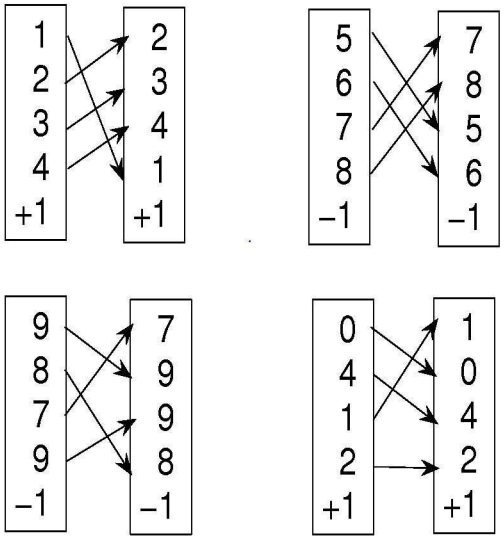
- Total of 7 funded teams attempted this KDD Challenge task and were evaluated by a rank error
- Double-blind submission and evaluation
- Our method had lowest average error



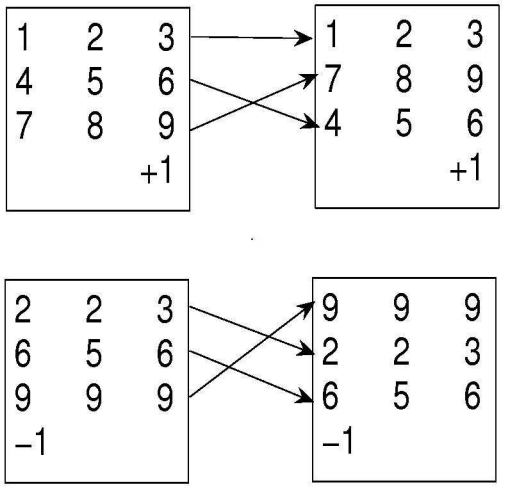
Permutation Invariant SVMs

- Suppose we want a classifier from a dataset of

some labeled vectors



some labeled matrices



- BUT, adversary has shuffled / permuted rows of each input
- Want an SVM that is invariant to this… an SVM on *sets*
- Equivalently, an SVM that undoes these permutations

Permutation Invariant SVMs

- When does SVM on sets or point clouds arise in practice?
- Example: images as bags of pixels or point clouds.
- Classically, image modeled as a vector of intensity values:

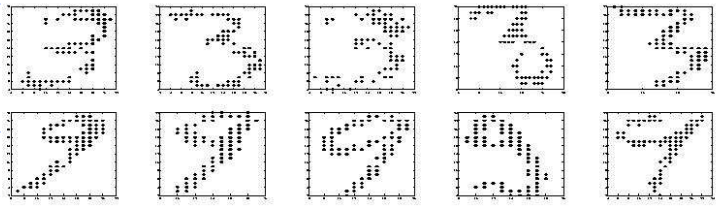
  $X_i = [\ i_1, \ldots , i_N\ ]$
- Instead, can model each image as a set of pixels:

  $X_i = \{\ (x_1,y_1), \ldots , (x_N, y_N)\ \}$

  $X_i = \{\ (x_1,y_1,i_1), \ldots , (x_N, y_N, i_N)\ \}$
- Order the points appear is irrelevant

positive

negative



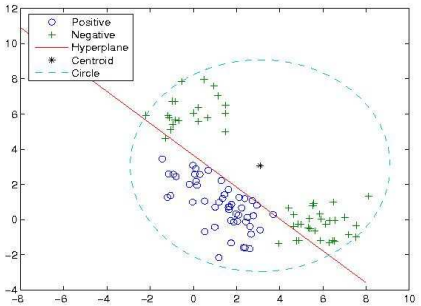
- IN: Training data = $X_1,\ldots,X_N$ with binary label $Y_1,\ldots,Y_N$

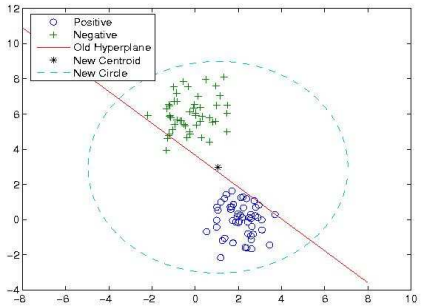
  Each $X_i=\{\ldots\}$ is a point set with arbitrary order
- OUT: Classifier w, undone permutations $A_1,\ldots,A_N$

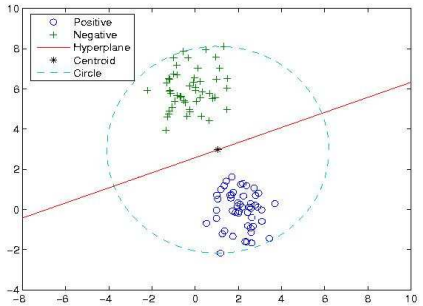
Permutation Invariant SVMs

- Use standard SVM classifier but improve its performance
- Try to minimize its VC-dimension < D^2/M^2+1 by permuting data towards centroid & away from margin
- π-SVM Algorithm:

1. Solve SVM QP on from $(\mathbf{x_i}, \mathbf{y_i})_{\mathbf{i=1}}^{\mathbf{n}}$ to find $(\mathbf{w^j}, \mathbf{b^j})$
2. Solve bounding sphere QCQP from $(\mathbf{x_i}, \mathbf{y_i})_{\mathbf{i=1}}^{\mathbf{n}}$ to find centroid $\mathbf{c^j}$
3. Let $\mathbf{A^{ij}} \leftarrow \mathrm{LAP}(\lambda y_i \mathbf{w^j x_i^\top} + \mathbf{c^j x_i^\top})$ for each i. Permute $\mathbf{x_i}$ with $\mathbf{A^{ij}}$, that is, $\mathbf{x_i} \leftarrow \mathbf{A^{ij} x_i}$.
4. Go Back to 1.



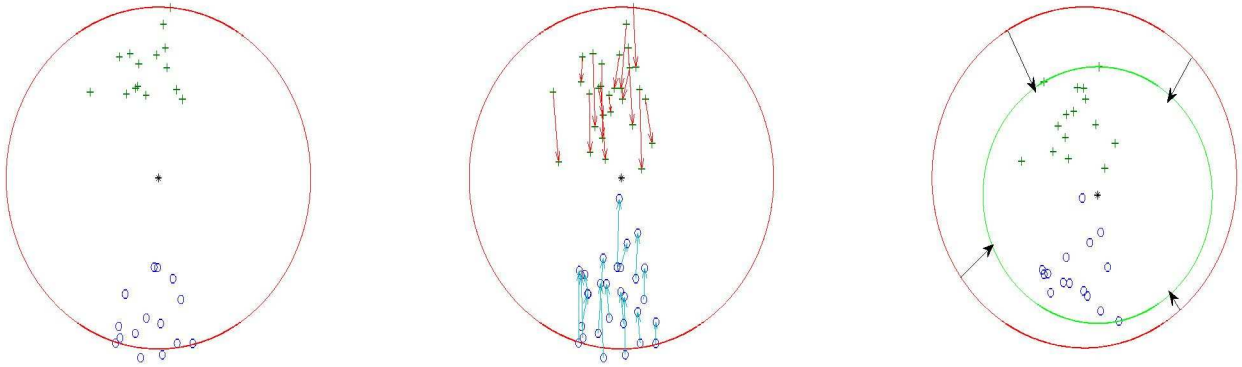




Permutation Invariant SVMs

- LAP is linear assignment problem (Hungarian Algorithm)
  $\min_A ||Z-AX||^2$ find optimal permutation matrix A to pull a matrix X to a target matrix Z

- Radius reduced by pulling all $\mathbf{x_i}$ close to the cetroid

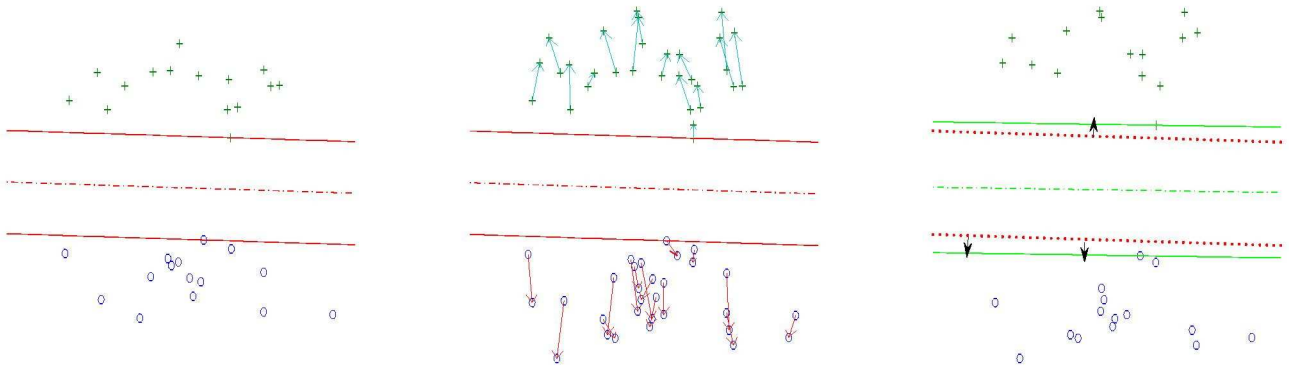


- $\mathrm{LAP}(\mathbf{x_i c}^\top)$

Permutation Invariant SVMs

- LAP is linear assignment problem (Hungarian Algorithm)
  $\min_A ||Z-AX||^2$ find optimal permutation matrix A to pull a matrix X to a target matrix Z

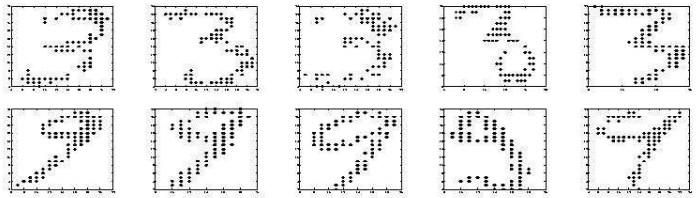
- Margin increased by pushing away $\mathbf{x_i}$ on the correct side



- $\text{LAP}(y_i \mathbf{x_i} \mathbf{w}^\top)$
- Aim: To pull towards margin, to push away from margin $\rightarrow$ optimize a weighted combination of costs.... $\text{LAP}(\lambda y_i \mathbf{w} \mathbf{x_i}^\top + \mathbf{c} \mathbf{x_i}^\top)$

π–SVM: Results

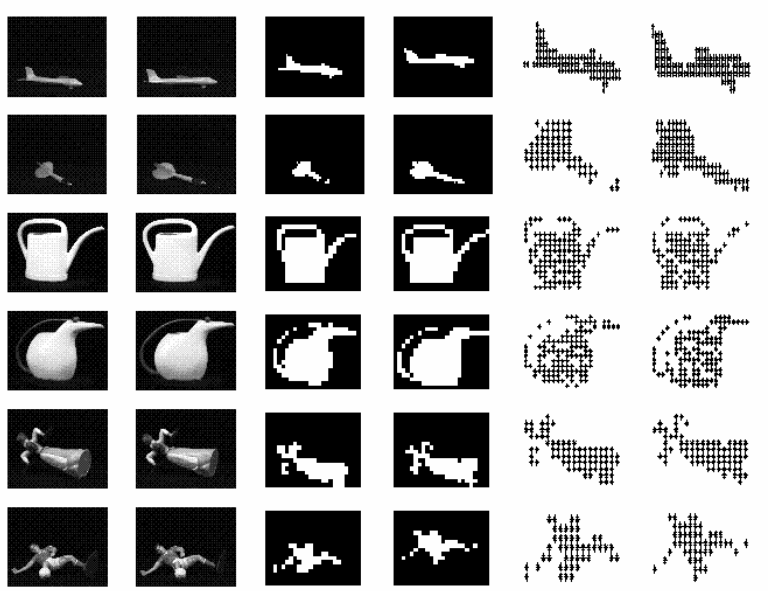
- NIST Digits 3 versus 9
- 100 training examples
- Classify images as point-sets with 70 dots each dealing with permutations and matching problems
- Or vectorize images and use standard SVM



Method	Accuracy
π-SVM RBF	97.00
Sorted RBF	88.00
Sorted Linear	92.00
Random RBF	69.00
Random Linear	80.00
Kondor/Jebara	94.00
SVM RBF (image)	95.00
SVM Linear (image)	91.00

π-SVM: Results

- Classify two types of images:
  - Planes versus Darts
  - Two different Jugs
  - Two different Dolls
- Translate & rotate in 3D
- About 5 training views per class
- Convert bright pixels in image into cloud of 120 (x,y) points vs. classifying raw images as vectors (SVM-RBF)



	Plane/Dart		Jugs		Dolls	
	Acc	Var	Acc	Var	Acc	Var
π-SVM	97.22	0.002	95.83	0.001	98.61	0.006
RBF	66.67	0.014	93.75	0.001	96.52	0.009
Linear	66.67	0.014	91.67	0.001	95.83	0.009
Sorted	95.83	0.003	81.94	0.008	96.52	0.006
Random	88.89	0.004	72.91	0.008	65.28	0.007
Kondor/Jebara	90.97	0.057	71.52	0.003	86.11	0.015

π-SVM: Results

- Standard UCI Classification Problems
- Permute input features dimensions for each input randomly

$X_1 = [\ 1\ 5\ 3\ 8\ ] \rightarrow X_1 = [\ 5\ 8\ 1\ 3\ ]$

$X_{13} = [\ 3\ 4\ 1\ 0\ ] \rightarrow X_{13} = [\ 0\ 4\ 1\ 3\ ]$

- See how well we can rectify this by estimating the permutations that give a max margin & min radius SVM.
- Almost recovers the accuracy of uncorrupted data

	Ionosphere	Pima	Heart
π-SVM	85.14	69.87	85.18
Actual	85.42	76.75	86.67
Sorted	76.61	67.19	74.89
Random	68.24	65.60	52.36

- Runtime per iteration (typically 5-10 total):
  SVM $O(N^3)$ + Centroid $O(N^3)$ + LAP Permutations $O(NM^3)$

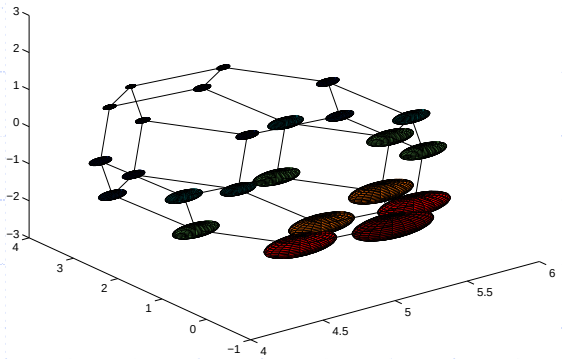
Faster Permutation Algorithms

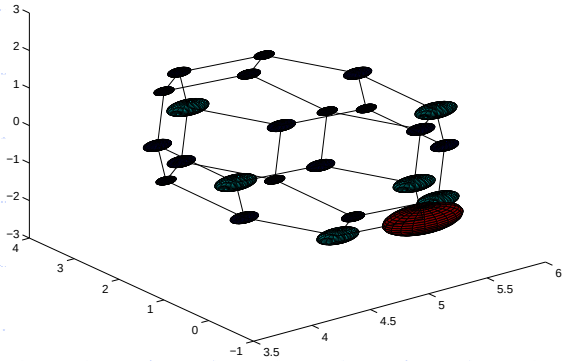
- Can we go beyond linear LAP functions such as:
  $\max_P \mathrm{tr}(P^T K)$ but instead have nonlinear $\max_P f(P)$
- Yes, but these are NP… require O(N!) time
- Can we do better?
- Yes: implement FFT on permutation group.
- Speeds up by square rooting time.

- Can we speed up beyond $O(bN^3)$?
- With sparse weighted graphs, $O(b\, N \min(N^2, |E| \log N))$
- Can we do better?
- Yes: implement B-Matching via Loopy Belief Propagation.
- Proof: guaranteed to converge

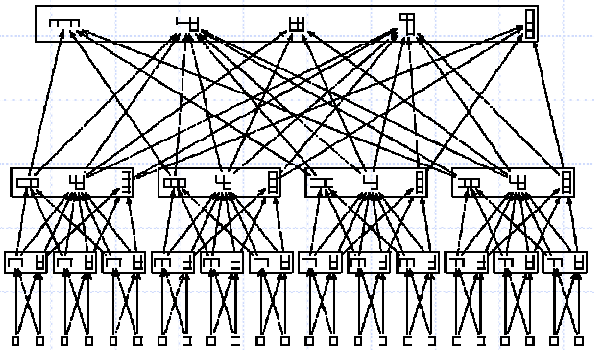
Permutation Group Theory

- Group-theory can be used to find the right (not adhoc) distance measures & affinities between elements of group.



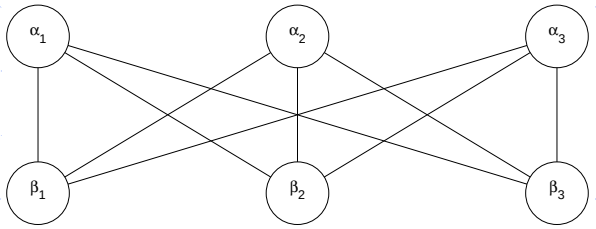


- Group-theoretic tools (Young tableux & irreducible representations) exist for permutations. Allow Fast Fourier Transform on group and square root speedup of O(N!)

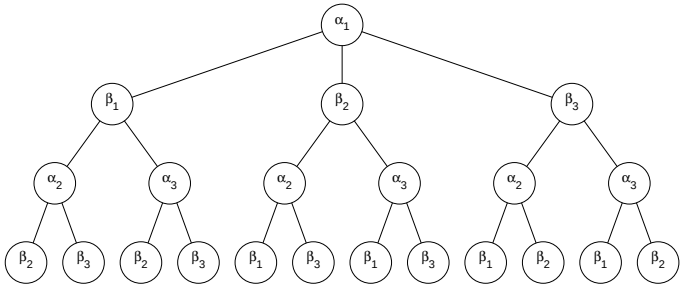


B-Matching Belief Propagation

- Loopy belief propagation: standard Bayes net inference tool
- Not guaranteed except for Gaussian graphical models.
- Pass max-product messages between cliques until settle
- Can implement matching as loopy BP on bipartite graph.
  $\alpha(i)$ row variable = j → $P_{ij}=1$
  $\beta(j)$ col variable = i → $P_{ij}=1$
  have cliques ϕ representing A_{ij} weights
  cliques φ for α and β agreement



- By unrolling the message passing in time, get a tree can *guarantee* convergence.



- Upper bound on # iterations of message passing < $O(1/\varepsilon)$
  ε = cost between 1st & 2nd best permutation

B-Matching Belief Propagation

- Belief propagation drastically outperforms fast classical maximum weight matching. Annealed version avoids degenerate cases when ε is tiny or zero. Gets ~ $O(N^{2.4})$

Average run time in seconds for random graphs or random K

N	128	256	512	1024
KM	2.971	19.79	189.6	2536
BP	3.033	13.36	58.20	356.6
Anneal	3.905	21.37	103.0	597.0

Average run time in seconds for non-random graphs or K

N	30	120	270	460	1080
KM	0.034	0.303	4.395	41.15	1076
BP	0.019	0.110	1.086	5.663	61.57
Anneal	0.114	0.735	7.403	30.00	202.8

Discussion

- Whenever you use nearest neighbor or k-nearest neighbor, try matching or b-matching (only slightly slower)
- Less greedy, more formal optimization problem, i.e. over permutation matrices
- Balances neighbors. Each point has b neighbors and is also a neighbor of b points. In-degree=Out-degree=b.
- Improves nonlinear dimensionality reduction / embedding
- Improves spectral clustering
- Allows permutationally invariant SVMs
- Various guaranteed speed ups via belief propagation or square-root speed up via FFT on permutation group.

Current work: Faster code versions.
Other applications of b-matching in learning.