Machine Learning: Think Big and Parallel

Day 2

Inderjit S. Dhillon
Dept of Computer Science
UT Austin

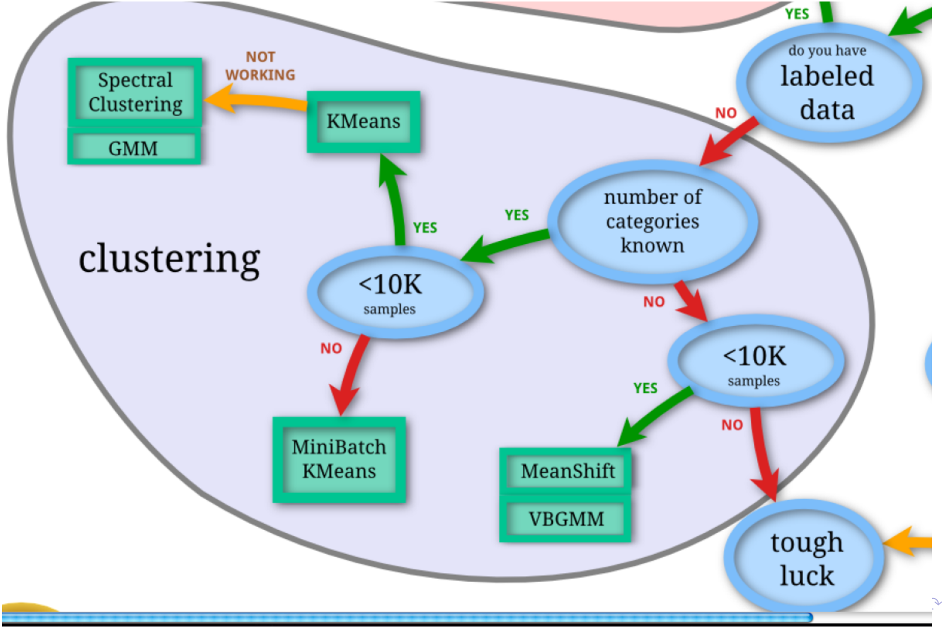
CS395T: Topics in Multicore Programming
Oct 3, 2013

Outline

- Scikit-learn: Machine Learning in Python
- Supervised Learning — day1
  - Regression: Least Squares, Lasso
  - Classification: kNN, SVM
- Unsupervised Learning — day2
  - Clustering: k-means, Spectral Clustering
  - Dimensionality Reduction: PCA, Matrix Factorization for Recommender Systems

Clustering

Clustering

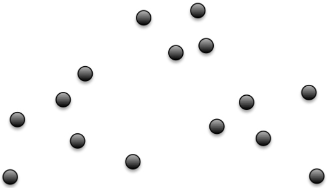

YES
do you have
labeled
data
NO
NOT
WORKING
Spectral
Clustering
GMM
KMeans
clustering
YES
YES
number of
categories
known
<10K
samples
NO
NO
MiniBatch
KMeans
<10K
samples
YES
NO
MeanShift
VBGMM
tough
luck

Clustering:
k-means Clustering

Clustering

Goal is to group "similar" instances together

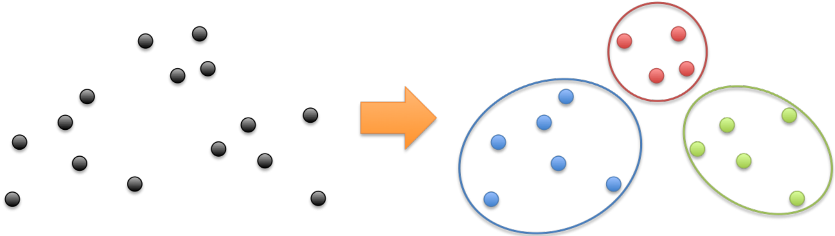
- Given data points $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, N$
- But no labels – unsupervised learning
- Useful for exploratory data analysis



Clustering

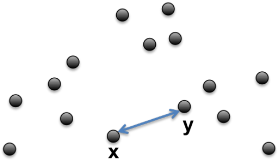
Goal is to group "similar" instances together

- Given data points $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, N$
- But no labels – unsupervised learning
- Useful for exploratory data analysis



Clustering

Need a measure of similarity (or distance) between two points $\boldsymbol{x}$ and $\mathbf{y}$



Popular distance metrics:

- Squared Euclidean distance $d(\boldsymbol{x}, \mathbf{y}) = \|\boldsymbol{x} - \mathbf{y}\|_2^2$
- Cosine similarity $d(\boldsymbol{x}, \mathbf{y}) = (\boldsymbol{x}^T \mathbf{y}) / \|\boldsymbol{x}\| \|\mathbf{y}\|$
- Manhattan distance $d(\boldsymbol{x}, \mathbf{y}) = \|\boldsymbol{x} - \mathbf{y}\|_1$

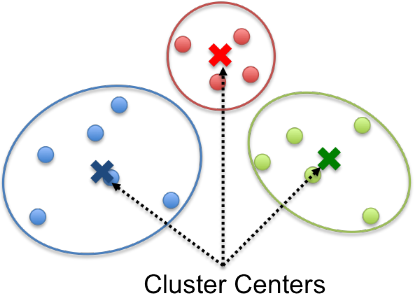
Clustering results are crucially dependent on the distance metric

k-means Clustering

Find k clusters that minimizes the objective:

$$J = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \|\boldsymbol{x} - \mathbf{m}_i\|_2^2$$

- $\mathcal{C}_i$: the set of points in cluster i
- $\mathbf{m}_i$: the mean(center) of cluster i
- Objective is non-convex and problem is NP-hard in general



Note: for $k = 1$, $J = \sum \|\boldsymbol{x} - \mathbf{m}\|_2^2$

$$\Rightarrow \text{solution is } \mathbf{m}^* = \frac{1}{N} \sum \boldsymbol{x}$$

k-means Algorithm (Batch)

Input: data points $\boldsymbol{x} \in \mathbb{R}^d$, number of clusters k
Output: cluster assignment $\mathcal{C}_i$ of data points, $i = 1, 2, \ldots, k$

1: Randomly partition the data into k clusters
2: **while** not converged **do**
3: Compute mean of each cluster i

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \boldsymbol{x}$$

4: For each $\boldsymbol{x}$, find its new cluster index:

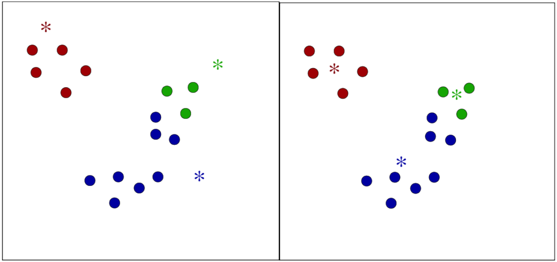
$$\pi(\boldsymbol{x}) = \arg \min_{1 \le i \le k} \|\boldsymbol{x} - \mathbf{m}_i\|_2^2$$

5: Update clusters:

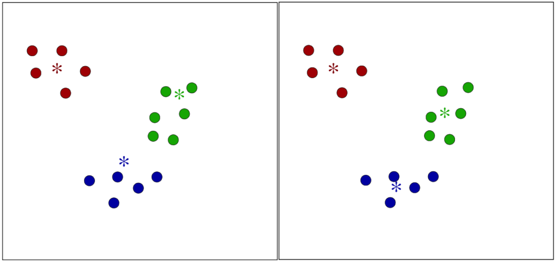
$$\mathcal{C}_i = \{\boldsymbol{x} | \pi(\boldsymbol{x}) = i\}$$

6: **end while**

k-means Clustering



1. Initial cluster assignment 2. Update cluster means



3. Assign to nearest cluster 4. Update cluster means

Convergence of k-means

Let the objective at t-th iteration be $J^{(t)} = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_i^{(t)}} \|\boldsymbol{x} - \mathbf{m}_i^{(t)}\|^2$

$$
\begin{aligned}
J^{(t)} &= \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_i^{(t)}} \|\boldsymbol{x} - \mathbf{m}_i^{(t)}\|^2 \\
&\geq \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_i^{(t)}} \|\boldsymbol{x} - \mathbf{m}_{\pi(\boldsymbol{x})}^{(t)}\|^2 = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_i^{(t+1)}} \|\boldsymbol{x} - \mathbf{m}_i^{(t)}\|^2 \\
&\geq \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_i^{(t+1)}} \|\boldsymbol{x} - \mathbf{m}_i^{(t+1)}\|^2 = J^{(t+1)}
\end{aligned}
$$

- Each step decreases the objective — guaranteed to converge
- But not necessarily to the global minimum

k-means Algorithm (Online)

Input: data points $\boldsymbol{x} \in \mathbb{R}^d$, number of clusters k
Output: cluster assignment $\mathcal{C}_i$ of data points, $i = 1, 2, \ldots, k$

1: Initialize means $\mathbf{m}_i$ and $n_i = 0$, $i = 1, 2, \ldots, k$
2: **while** not converged **do**
3: Pick a data point $\boldsymbol{x}$ and determine cluster $\pi(\boldsymbol{x})$

$$\pi(\boldsymbol{x}) = \arg\min_{1 \leq i \leq k} \|\boldsymbol{x} - \mathbf{m}_i\|_2^2$$

4: Update mean $\mathbf{m}_{\pi(\boldsymbol{x})}$

$$n_{\pi(\boldsymbol{x})} = n_{\pi(\boldsymbol{x})} + 1 \quad \text{and} \quad \mathbf{m}_{\pi(\boldsymbol{x})} = \mathbf{m}_{\pi(\boldsymbol{x})} + \frac{1}{n_{\pi(\boldsymbol{x})}}(\boldsymbol{x} - \mathbf{m}_{\pi(\boldsymbol{x})})$$

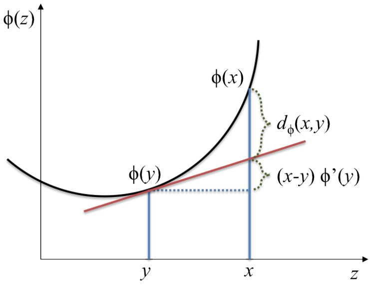
5: **end while**

k-means with Bregman Divergences

Bregman divergences:

$$d_\Phi(\boldsymbol{x}, \mathbf{y}) = \Phi(\boldsymbol{x}) - \Phi(\mathbf{y}) - \langle \boldsymbol{x} - \mathbf{y}, \nabla\Phi(\mathbf{y})\rangle,$$

where Φ is strictly convex & differentiable



Examples of $d_\Phi(\boldsymbol{x}, \mathbf{y})$:

- Squared Euclidean distance: $\|\boldsymbol{x} - \mathbf{y}\|_2^2$
- KL-divergence: $\sum_i x_i \log(\frac{x_i}{y_i})$
- Itakura-Saito distance: $\sum_i \left(\frac{x_i}{y_i} - \log(\frac{x_i}{y_i}) - 1\right)$

For Bregman divergences, the arithmetic mean is the best predictor:

$$\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i = \arg\min_{\mathbf{c}} \sum_{i=1}^{N} d_\Phi(\boldsymbol{x}_i, \mathbf{c})$$

Clustering:
Spectral Clustering

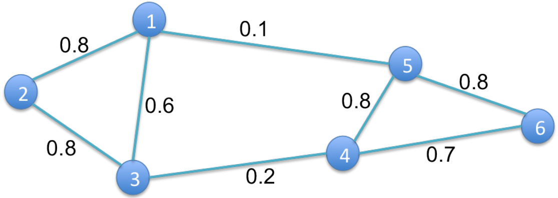
Spectral Clustering

Given:

- Number of clusters k
- Graph $G = (\mathcal{V}, \mathcal{E})$
  - Set of nodes: $\mathcal{V} = \{1, \cdots, n\}$
  - Set of edges: $\mathcal{E} = \{e_{ij} | i, j \in \mathcal{V}\}$ — similarity between nodes
  - Weighted adjacency matrix $W \in \mathbb{R}^{n \times n}$

$$W_{ij} = \begin{cases} e_{ij}, & \text{if there is an edge between nodes } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$$

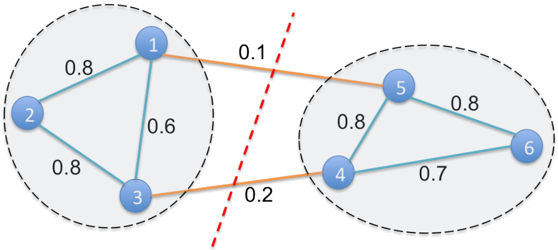
    W is symmetric if G is an undirected graph
  - Degree matrix: a diagonal matrix D where $D_{ii} = \sum_{j=1}^{n} W_{ij}$



Spectral Clustering

Goal:

- Partition $\mathcal{V}$ into k disjoint clusters: $\mathcal{V}_1, \ldots, \mathcal{V}_k$
- Within-cluster: large weights
- Between-cluster: small weights

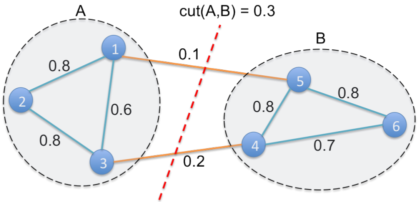


An ideal but trivial case: G has exactly k connected components

Graph Cut

- Small cut between clusters

$$\text{cut}(A, B) = \frac{1}{2} \sum_{i \in A, j \in B} W_{ij}$$



- Balance of cluster sizes $|\mathcal{V}_i|$
- Objective:

$$\text{RatioCut}(\mathcal{V}_1, \ldots, \mathcal{V}_k) = \sum_{i=1}^{k} \frac{\text{cut}(\mathcal{V}_i, \mathcal{V} \setminus \mathcal{V}_i)}{|\mathcal{V}_i|}$$

- Goal: minimize $\text{RatioCut}(\mathcal{V}_1, \ldots, \mathcal{V}_k)$

Graph Laplacian

Laplacian: $L = D - W$

- L: symmetric and positive semi-definite
- Eigenvalues: $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$
- # of connected components in G = # of 0 eigenvalues of L
- For all $\mathbf{f} \in \mathbb{R}^n$,

$$\mathbf{f}^T L \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{n} W_{ij}(f_i - f_j)^2$$

Most importantly,

$$\text{RatioCut}(A_1, \ldots, A_k) = \text{trace}(F^T L F)$$

for a special $F = [\mathbf{f}_1, \ldots, \mathbf{f}_k]$, where $F_{ij} = \begin{cases} 1/\sqrt{|\mathcal{V}_j|}, & \text{if } i \in \mathcal{V}_j \\ 0, & \text{otherwise} \end{cases}$

Relaxation of Cut Minimization

In general, minimizing RatioCut is NP-hard!
However, based on

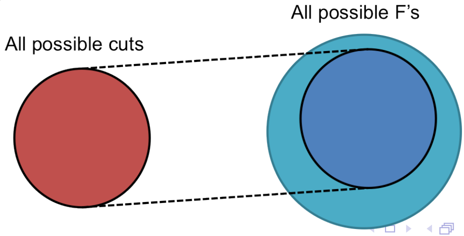
$$\text{RatioCut}(\mathcal{V}_1, \ldots, \mathcal{V}_k) = \text{trace}(F^T L F),$$

we have the following relaxation:

- Solve

  $$F^* = \arg \min_{F \in \mathbb{R}^{n \times k}} \text{trace}(F^T L F)$$

  which are exactly the first k eigenvectors of L
- Recover $\mathcal{V}_1, \ldots, \mathcal{V}_k$ from F^* by distance-based clustering algorithms (e.g. k-means)



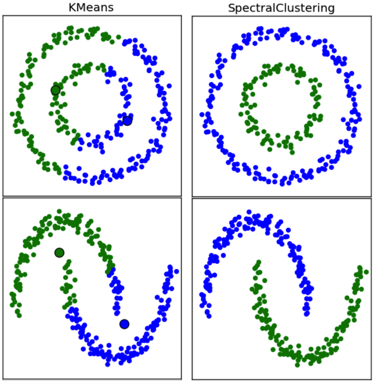
Spectral Clustering vs. k-means

Clustering data points $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1, \ldots, N$

- First construct kernel matrix
  e.g. Gaussian kernel:

$$W_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma}$$

- k-means algorithm can only find linear decision boundaries
- Spectral clustering allows us to find non-convex boundaries



Variants of Graph Laplacian

Normalized Laplacian:

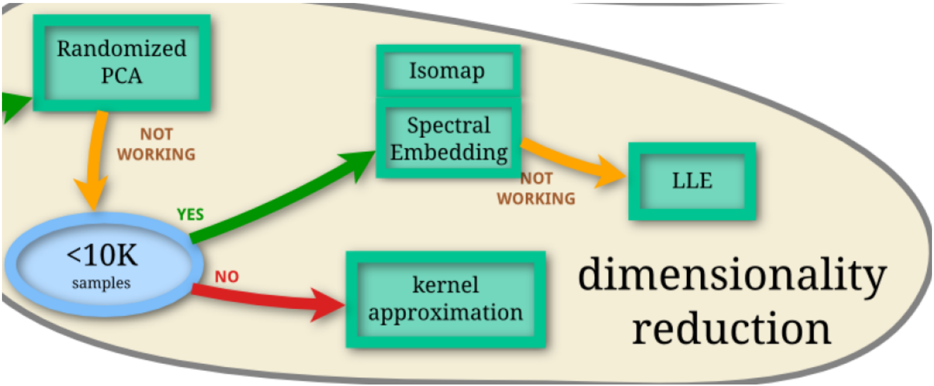
- $L = I_n - D^{-1/2} W D^{-1/2}$
- $\text{NormalizedCut}(\mathcal{V}_1, \dots, \mathcal{V}_k) = \sum_{i=1}^{k} \frac{\text{cut}(\mathcal{V}_i, \mathcal{V} \setminus \mathcal{V}_i)}{\text{vol}(\mathcal{V}_i)}$, where $\text{vol}(\mathcal{V}_i) = \sum_{j \in \mathcal{V}_i} D_{jj}$

Signed Laplacian:

- $L = \bar{D} - W$, where $\bar{D}_{ii} = \sum_{j=1}^{n} |W_{ij}|$
- Handle "signed" similarity graphs with both positive and negative edge weights

Dimensionality Reduction

Dimensionality Reduction



Dimensionality Reduction: Principal Component Analysis

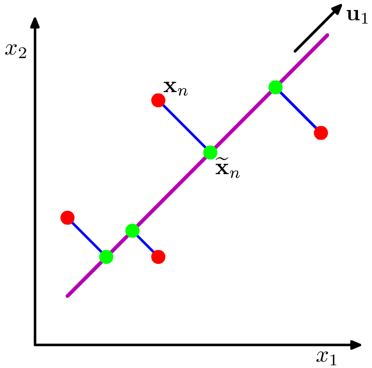
Principal Component Analysis

N observations: $\{\boldsymbol{x}_i \in \mathcal{R}^D : i = 1 \ldots, N\}$

Goal:

- Project data onto a space with dimensional $M < D$
- Maximize the variance of the projected data

Example:



PCA: Projection to one dimensional space ($M = 1$)

Empirical mean and variance of $\{\boldsymbol{x}_n\}$:

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$$

$$S = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T$$

$\boldsymbol{w}$: the direction of the space

- $\|\boldsymbol{w}\|_2 = 1$ as the length is not important.
- $Proj_{\boldsymbol{w}}(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{x}_n, \quad \forall n = 1, \ldots, N$
- $Proj_{\boldsymbol{w}}(\bar{\boldsymbol{x}}) = \boldsymbol{w}^T \bar{\boldsymbol{x}}$
- The variance of $\{Proj_{\boldsymbol{w}} \boldsymbol{x}_n\}$:

$$\frac{1}{N}\sum_{n=1}^{N}\left(\boldsymbol{w}^T \boldsymbol{x}_n - \boldsymbol{w}^T \bar{\boldsymbol{x}}\right)^2 \equiv \boldsymbol{w}^T S \boldsymbol{w}.$$

PCA: Projection to one dimensional space ($M = 1$)

Goal: maximize the variance of the projected data $\{Proj_{\boldsymbol{w}}(\boldsymbol{x}_n)\}$:

$$\arg \max_{\boldsymbol{w}_1 : \|\boldsymbol{w}_1\| = 1} \quad \boldsymbol{w}_1^T S \boldsymbol{w}_1$$

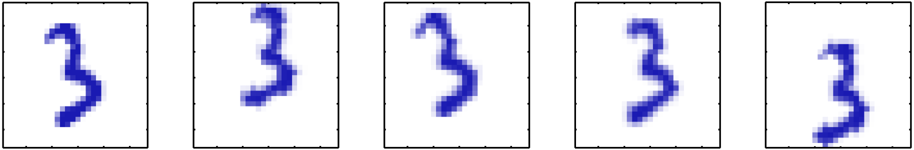
- Lagrangian $L(\boldsymbol{w}_1, \lambda_1) = \boldsymbol{w}_1^T S \boldsymbol{w}_1 + \lambda_1 \left(1 - \boldsymbol{w}_1^T \boldsymbol{w}_1\right)$
- $\nabla L(\boldsymbol{w}_1, \lambda_1) = 0$ implies that $S\boldsymbol{w}_1^* = \lambda_1 \boldsymbol{w}_1^*$.
- $\boldsymbol{w}_1^*$ is the eigenvector of S corresponding the largest eigenvalue λ_1^*, also called the 1-st principal component.
- In general, the k-th principal component $\boldsymbol{w}_k^*$ is the eigenvector of S corresponding to the k-th largest eigenvalue λ_k^*.

Dimension reduction:

- $W = [\boldsymbol{w}_1^*, \ldots, \boldsymbol{w}_M^*]$: formed by M principal components.
- $Proj_W(\boldsymbol{x}) = W^T \boldsymbol{x}$: the projected vector in M dimensional space.

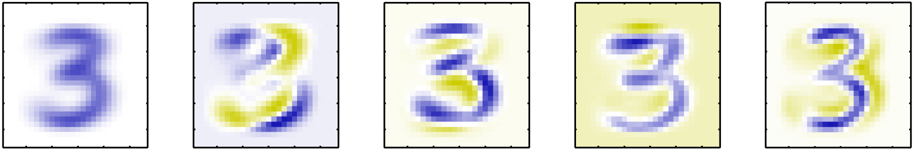
PCA: An Example

A set of digit images



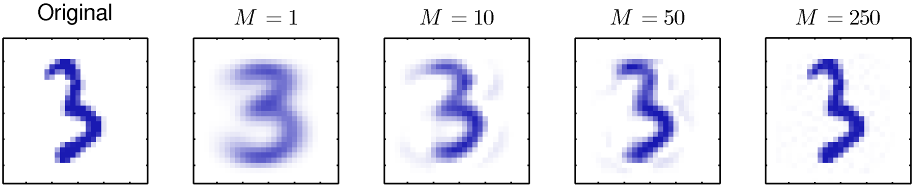
The mean vector $\bar{\boldsymbol{x}}$ and the first 4 principal components:

Mean | $\lambda_1 = 3.4 \cdot 10^5$ | $\lambda_2 = 2.8 \cdot 10^5$ | $\lambda_3 = 2.4 \cdot 10^5$ | $\lambda_4 = 1.6 \cdot 10^5$

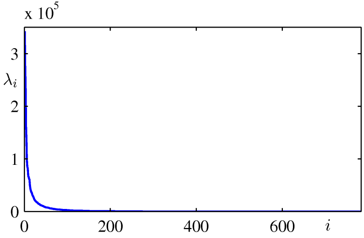


PCA: An Example

Various M:



Eigenvalue Spectrum:



(a)

Dimensionality Reduction: Matrix Factorization

Matrix Factorization

Matrix Factorization

- A motivating example: recommender systems
- Problem Formulation
- Latent Feature Space
- Existing Methods

Recommender Systems

Rating Matrix

Items

Users

	Movie 1	Movie 2								Movie 10	Movie 11
Hsiang-Fu	1			5			3		5		2
Cho-Jui		2		3			5		2	5	
Si Si					3	?	5		3		
Inderjit	2		5			3		4		2	
Kai-Yang				5			5				1
Donghyuk		5			1				5		
Naga	1			1				2			4

Matrix Factorization Approach $A \approx WH^T$

H^T

-0.07	-0.11	-0.53	-0.46	-0.06	-0.05	-0.53	-0.07	-0.35	-0.19	-0.14
0.13	-0.42	0.45	0.17	-0.25	-0.17	-0.18	0.27	-0.59	0.05	0.14
-0.21	-0.43	-0.23	0.16	0.08	0.17	0.57	-0.39	-0.37	-0.08	-0.15

W

-8.72	0.03	-1.03
-7.56	-0.79	0.62
-4.07	-3.95	2.55
-3.52	3.73	-3.32
-7.78	2.34	2.33
-2.44	-5.29	-3.92
-1.78	1.90	-1.68

1			5			3		5		2
	2		3			5		2	5	
				3		5		3		
2		5			3		4		2	
			5			5				1
	5			1				5		
1			1				2			4

Matrix Factorization Approach $A \approx WH^T$

H^T

-0.07	-0.11	-0.53	-0.46	-0.06	-0.05	-0.53	-0.07	-0.35	-0.19	-0.14
0.13	-0.42	0.45	0.17	-0.25	-0.17	-0.18	0.27	-0.59	0.05	0.14
-0.21	-0.43	-0.23	0.16	0.08	0.17	0.57	-0.39	-0.37	-0.08	-0.15

W

-8.72	0.03	-1.03
-7.56	-0.79	0.62
-4.07	-3.95	2.55
-3.52	3.73	-3.32
-7.78	2.34	2.33
-2.44	-5.29	-3.92
-1.78	1.90	-1.68

1			5			3		5		2
	2		3			5		2	5	
				3	?	5		3		
2		5			3		4		2	
			5			5				1
	5			1				5		
1			1				2			4

Matrix Factorization Approach

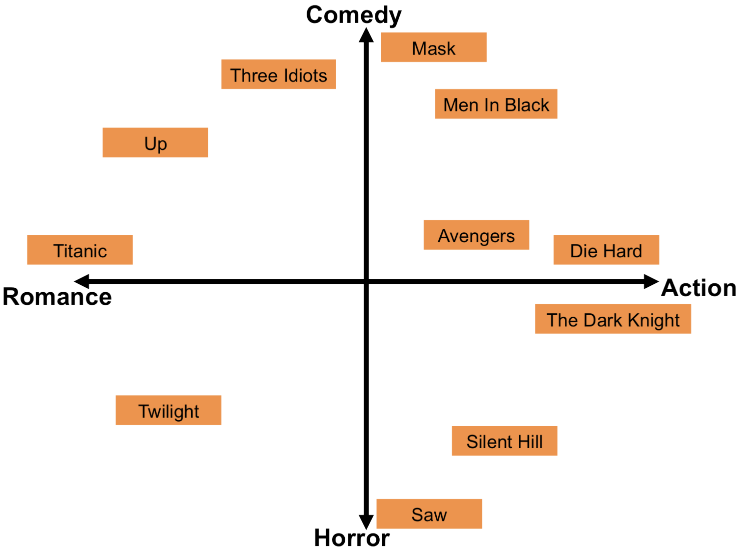
$$\min_{\substack{W \in \mathbb{R}^{m \times k} \\ H \in \mathbb{R}^{n \times k}}} \sum_{(i,j) \in \Omega} (A_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j)^2 + \lambda \left(\|W\|_F^2 + \|H\|_F^2 \right),$$

- $\Omega = \{(i,j) \mid A_{ij} \text{ is observed}\}$
- Regularized terms to avoid over-fitting

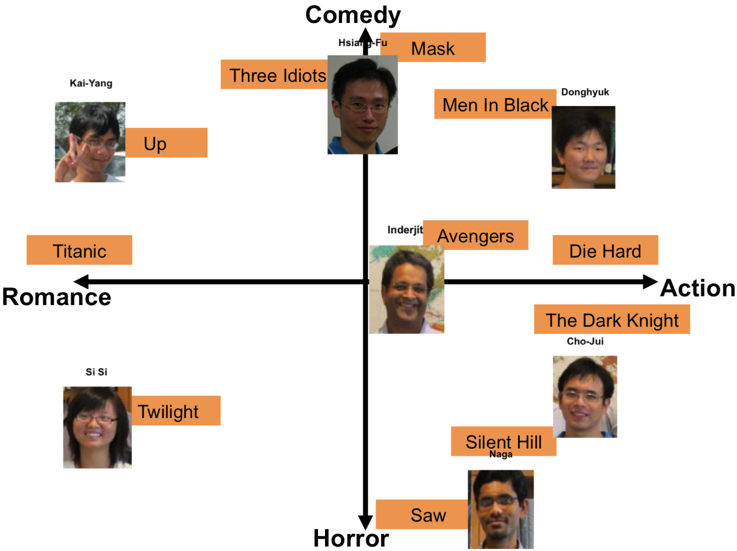
Matrix factorization maps users/items to latent feature space $\mathbb{R}^k$

- the i^{th} user $\Rightarrow$ i^{th} row of W, $\boldsymbol{w}_i^T$,
- the j^{th} item $\Rightarrow$ j^{th} row of H, $\boldsymbol{h}_j^T$.
- $\boldsymbol{w}_i^T \boldsymbol{h}_j$: measures the interaction between i^{th} user and j^{th} item.

Latent Feature Space



Latent Feature Space



Other Factorizations

Nonnegative Matrix Factorization

$$\min_{W,H} \|A - WH^T\|_F^2 + \lambda \|W\|_F^2 + \lambda \|H\|_F^2$$

- Each entry is positive
- A is either fully or partially observed
- Goal: find the nonnegative latent factors

Existing Methods

ALS: Alternating Least Squares

Fix either H or W and optimize the other:

LS sub-problem: $\min_{\boldsymbol{w}_i \in \mathcal{R}^k} \sum_{j \in \Omega_i} (A_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j)^2 + \lambda \|\boldsymbol{w}_i\|^2$

- it has closed form solution.
- An iteration: update W/H once
- $O(|\Omega|k^2 + (m+n)k^3)$

$$\begin{pmatrix} & H^T & \end{pmatrix}$$

$$\begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \boldsymbol{w}_3^T \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

SGM: Stochastic Gradient Method

$$\begin{pmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \boldsymbol{h}_3 \end{pmatrix}$$

$$\begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \boldsymbol{w}_3^T \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

SGM update: pick $(i, j) \in \Omega$

- $R_{ij} \leftarrow A_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j$
- $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta(\lambda \boldsymbol{w}_i - R_{ij} \boldsymbol{h}_j)$,
- $\boldsymbol{h}_j \leftarrow \boldsymbol{h}_j - \eta(\lambda \boldsymbol{h}_j - R_{ij} \boldsymbol{w}_i)$.

An iteration : $|\Omega|$ updates

- Time per iteration: $O(|\Omega| k)$,
  better than $O(|\Omega| k^2)$ for ALS
- Convergence is sensitive to the learning rate η.

Coordinate Descent

Update a variable at a time:

$$w_{it} \leftarrow \frac{\sum_{j \in \Omega_i} (A_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j + w_{it} h_{jt}) h_{jt}}{\lambda + \sum_{j \in \Omega_i} h_{jt}^2}.$$

- Subproblem is just a single-variate quadratic problem
- $\Omega_i = \{j : (i,j) \in \Omega\}$
- Can be done in $O(|\Omega_i|)$

Update Sequence:

- Item/user-wise update:
  - pick a user i or an item j
  - update the i-th row of W or the j-th column of H
- Feature-wise update:
  - pick a feature index $t \in \{1, \ldots, k\}$
  - update t-column of W and H alternatively

Thoughts on Parallelization

List of Methods in Scikit-learn

- Regression:
  - **Linear**, **Ridge**, **Lasso**, Elastic Net, Bayesian Regression, Support Vector Regression, ...
- Classification:
  - k**NN**, **SVM**, Perceptron, Logistic Regression, Naive Bayes, Decision Trees, Random Forest, AdaBoost, ...
- Clustering:
  - **k-means**, **Spectral Clustering**, Affinity Propagation, Mean-Shift, DBSCAN, Hierarchical Clustering, ...
- Dimensionality Reduction:
  - (kernel/sparse) **PCA**, **MF**, **NMF**, Truncated SVD (LSA), Dictionary Learning, Factor Analysis, Independent Component Analysis, ...

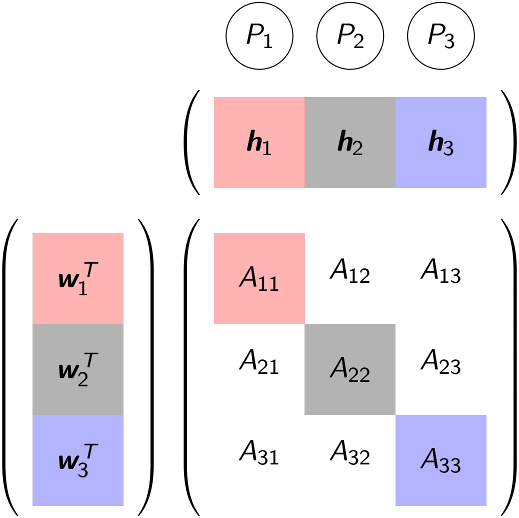
Potential Projects

Goal: **A fully parallelized version of Scikit-learn**

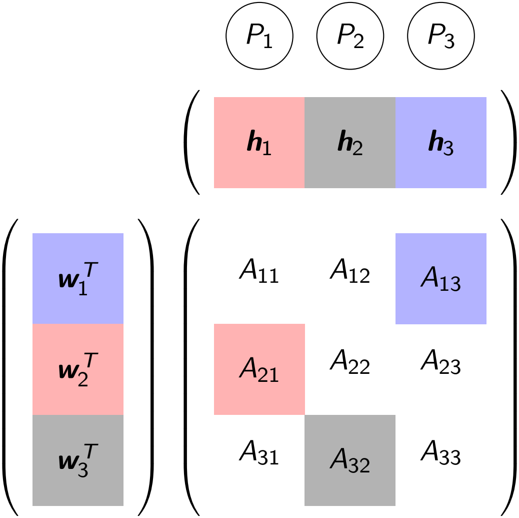
- Regression:
  - parallel solvers for **Lasso/Ridge**
- Classification:
  - parallel solvers for **SVM, Logistic Regression**
- Clustering:
  - parallel **k-means**
- Dimensionality Reduction:
  - parallel **MF/NMF** for recommender system

Example: Parallel Matrix Factorization for Recommender Systems

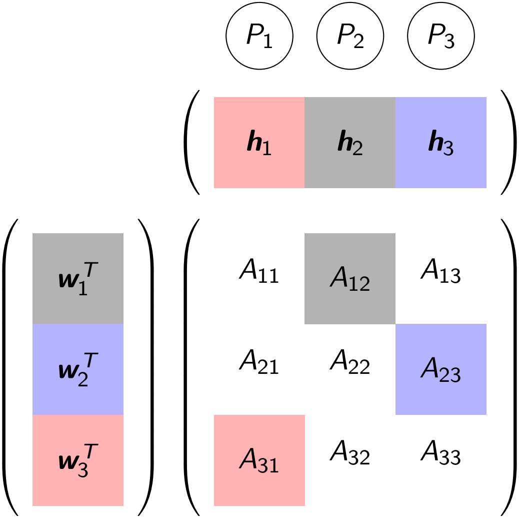
DSGD: Distributed SGM



DSGD: Distributed SGM



DSGD: Distributed SGM



Parallel Coordinate Descent

Feature-wise Update: CCD++

Rank-one decomposition:

$$WH^T = [\cdots \bar{\boldsymbol{w}}_t \cdots][\cdots \bar{\boldsymbol{h}}_t \cdots]^T = \sum_{t=1}^{k} \bar{\boldsymbol{w}}_t \bar{\boldsymbol{h}}_t^T$$

CCD++: picks a latent feature t and updates $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$

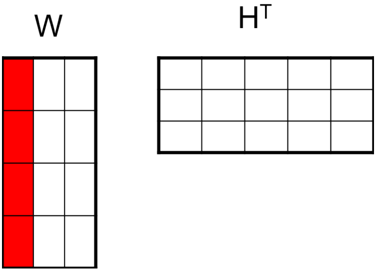
$$\min_{\boldsymbol{u}\in\mathbb{R}^m, \boldsymbol{v}\in\mathbb{R}^n} \sum_{(i,j)\in\Omega} \left(\hat{R}_{ij} - u_i v_j\right)^2 + \lambda(\|\boldsymbol{u}\|^2 + \|\boldsymbol{v}\|^2).$$

- $R_{ij} = A_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j$
- $\hat{R}_{ij} = R_{ij} + \bar{w}_{ti}\bar{h}_{tj}, \ \forall (i,j) \in \Omega$

  $(\boldsymbol{u}^*, \boldsymbol{v}^*)$ is a rank-one approximation of $\hat{R}$
- Apply the CCD iteration T times to obtain $(\boldsymbol{u}^*, \boldsymbol{v}^*)$
- CCD: item/user-wise update

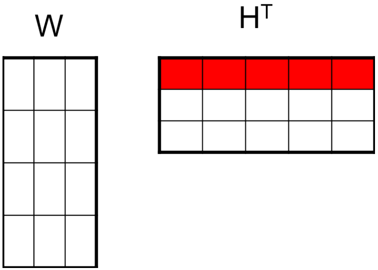
Feature-wise Update: CCD++

When $T = 2$



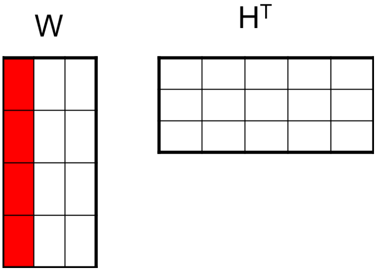
Feature-wise Update: CCD++

When $T = 2$



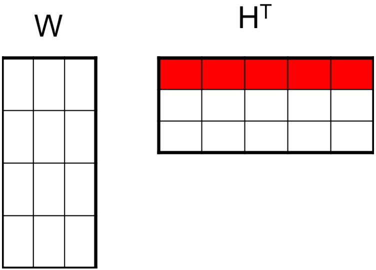
Feature-wise Update: CCD++

When $T = 2$



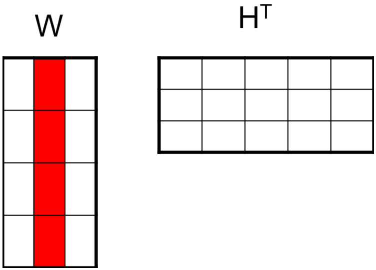
Feature-wise Update: CCD++

When $T = 2$



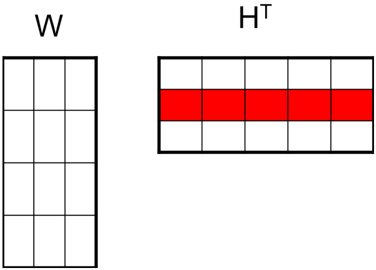
Feature-wise Update: CCD++

When $T = 2$



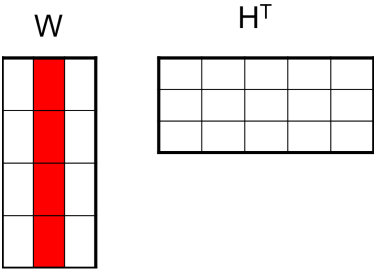
Feature-wise Update: CCD++

When $T = 2$



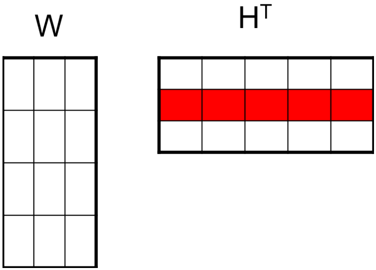
Feature-wise Update: CCD++

When $T = 2$



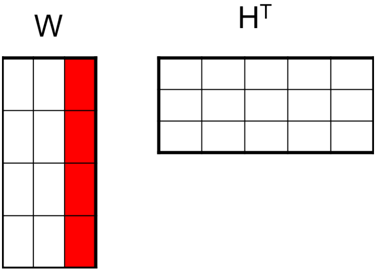
Feature-wise Update: CCD++

When $T = 2$



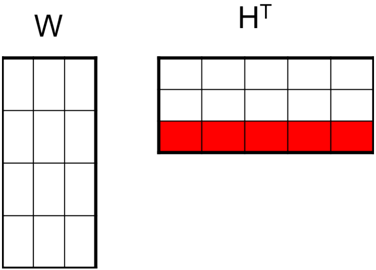
Feature-wise Update: CCD++

When $T = 2$



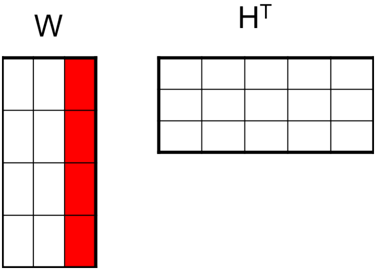
Feature-wise Update: CCD++

When $T = 2$



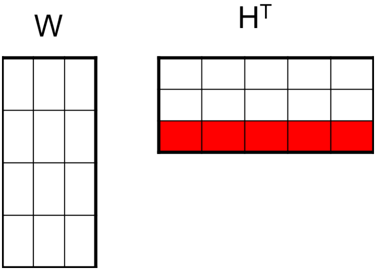
Feature-wise Update: CCD++

When $T = 2$



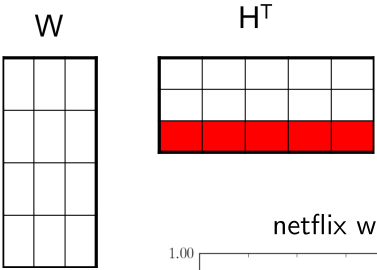
Feature-wise Update: CCD++

When $T = 2$

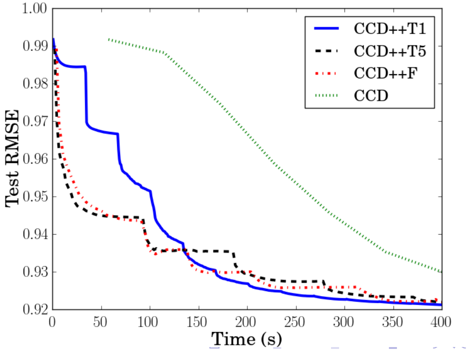


Feature-wise Update: CCD++

When $T = 2$



netflix with $k = 40$



- Cycle through k feature dimensions
- $O(\frac{2T}{T+1})$ faster than CCD

Problems of Different Scales

W, H, and R fit in the memory of a single computer

- **Multi-core systems** are an appropriate framework.
- All cores share the same memory space.
- Latest variables are always available to access.

W, H or R exceeds memory capacity of one computer

- Can still run on one computer, but leads to disk swap.
- **Distributed systems** are appropriate.
- Matrices are stored in memory of the distributed system $\Rightarrow$ only local data can be accessed fast.
- Require communication to access latest variables.

Parallelization of CCD++

- Key: to parallelize CCD to obtain $(\boldsymbol{u}^*, \boldsymbol{v}^*)$.
- Fact: each u_i can be updated independently.

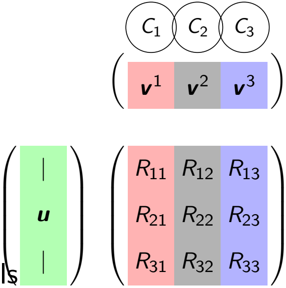
Partition $\boldsymbol{u}$ and $\boldsymbol{v}$ into p sub-vectors.

- $\boldsymbol{u} \Rightarrow \{\boldsymbol{u}^1, \ldots, \boldsymbol{u}^r, \ldots, \boldsymbol{u}^p\}$
- $\boldsymbol{v} \Rightarrow \{\boldsymbol{v}^1, \ldots, \boldsymbol{v}^r, \ldots, \boldsymbol{v}^p\}$

Run in parallel: the r^{th} core C_r:

- computes $(\boldsymbol{u}^*)^r$ and $(\boldsymbol{v}^*)^r$
- updates $\bar{\boldsymbol{w}}_t^r$ and $\bar{\boldsymbol{h}}_t^r$

See the paper Yu et al, 2013 for more details



CCD++ on Distributed Systems

W, H, R are distributed over the memory of different computers.

$$R \Rightarrow \quad \underset{C_1}{\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & & \\ R_{31} & & \end{pmatrix}} \quad \underset{C_2}{\begin{pmatrix} & R_{12} & \\ R_{21} & R_{22} & R_{23} \\ & R_{32} & \end{pmatrix}} \quad \underset{C_3}{\begin{pmatrix} & & R_{13} \\ & & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}}$$

$$W \Rightarrow \begin{pmatrix} W^1 & W^2 & W^3 \end{pmatrix}^T \qquad H \Rightarrow \begin{pmatrix} H^1 & H^2 & H^3 \end{pmatrix}^T$$

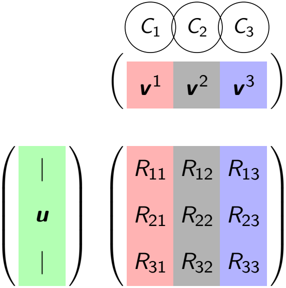
CCD++ on Distributed Systems

Distributed update: computer C_r:

- obtains $(\boldsymbol{u}^r, \boldsymbol{v}^r)$ using CCD:

  computes $\boldsymbol{u}^r$ and broadcasts it
  computes $\boldsymbol{v}^r$ and broadcasts it

- updates $(\bar{\boldsymbol{w}}_t^r, \bar{\boldsymbol{h}}_t^r) \leftarrow (\boldsymbol{u}^r, \boldsymbol{v}^r)$



References

[1] R. Gemulla, P. J. Haas, E. Nijkamp, and Y. Sismanis *Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent*. KDD, 2011.

[2] F. Niu, B. Recht, C. Re, and S. J. Wright *Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent*. NIPS, 2011.

[3] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin *A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems*. RecSys, 2013.

[4] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon *Parallel Matrix Factorization for Recommender Systems*. KAIS, 2013.