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: $k$NN, SVM

- Unsupervised Learning — day2
  - Clustering: $k$-means, Spectral Clustering
  - Dimensionality Reduction: PCA, Matrix Factorization for Recommender Systems
Clustering
Clustering

- Spectral Clustering
- GMM
- KMeans
- MiniBatch KMeans
- MeanShift
- VBGMM
- <10K samples
- number of categories known
- do you have labeled data
- <10K samples
- tough luck
Clustering:

$k$-means Clustering
Clustering

Goal is to group “similar” instances together

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

Goal is to group “similar” instances together

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

Need a measure of similarity (or distance) between two points \( x \) and \( y \)

Popular distance metrics:

- Squared Euclidean distance \( \|x - y\|_2^2 \)
- Cosine similarity \( \frac{x^T y}{\|x\| \|y\|} \)
- Manhattan distance \( \|x - y\|_1 \)

Clustering results are crucially dependent on the distance metric
**k-means Clustering**

Find $k$ clusters that minimizes the objective:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - m_i\|^2$$

- $C_i$: the set of points in cluster $i$
- $m_i$: the mean (center) of cluster $i$

Objective is non-convex and problem is NP-hard in general

Note: for $k = 1$, $J = \sum \|x - m\|^2$

$\Rightarrow$ solution is $m^* = \frac{1}{N} \sum x$
**k-means Algorithm (Batch)**

**Input:** data points $x \in \mathbb{R}^d$, number of clusters $k$

**Output:** cluster assignment $C_i$ 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 $i$
   \[
   m_i = \frac{1}{n_i} \sum_{x \in C_i} x
   \]
4. For each $x$, find its new cluster index:
   \[
   \pi(x) = \arg \min_{1 \leq i \leq k} \| x - m_i \|_2^2
   \]
5. Update clusters:
   \[
   C_i = \{ x | \pi(x) = i \}
   \]
6. end while
**k-means Clustering**

1. Initial cluster assignment

2. Update cluster means

3. Assign to nearest cluster

4. Update cluster cluster means
Convergence of $k$-means

Let the objective at $t$-th iteration be $J(t) = \sum_{i=1}^{k} \sum_{x \in C_i^{(t)}} \| x - m_i^{(t)} \|^2$

\[
J(t) = \sum_{i=1}^{k} \sum_{x \in C_i^{(t)}} \| x - m_i^{(t)} \|^2 \\
\geq \sum_{i=1}^{k} \sum_{x \in C_i^{(t)}} \| x - m_{\pi(x)}^{(t)} \|^2 = \sum_{i=1}^{k} \sum_{x \in C_i^{(t+1)}} \| x - m_i^{(t)} \|^2 \\
\geq \sum_{i=1}^{k} \sum_{x \in C_i^{(t+1)}} \| x - m_i^{(t+1)} \|^2 = J(t+1)
\]

- Each step decreases the objective — guaranteed to converge
- But not necessarily to the global minimum
**Input:** data points $x \in \mathbb{R}^d$, number of clusters $k$

**Output:** cluster assignment $C_i$ of data points, $i = 1, 2, \ldots, k$

1: Initialize means $m_i$ and $n_i = 0$, $i = 1, 2, \ldots, k$

2: **while** not converged **do**

3: Pick a data point $x$ and determine cluster $\pi(x)$

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

4: Update mean $m_{\pi(x)}$

$$n_{\pi(x)} = n_{\pi(x)} + 1 \quad \text{and} \quad m_{\pi(x)} = m_{\pi(x)} + \frac{1}{n_{\pi(x)}} (x - m_{\pi(x)})$$

5: **end while**
**k-means with Bregman Divergences**

Bregman divergences:

\[ d_\Phi(x, y) = \Phi(x) - \Phi(y) - \langle x - y, \nabla \Phi(y) \rangle, \]

where \( \Phi \) is strictly convex & differentiable

Examples of \( d_\Phi(x, y) \):

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

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

\[
\frac{1}{N} \sum_{i=1}^{N} x_i = \arg \min_c \sum_{i=1}^{N} d_\Phi(x_i, c)
\]
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

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

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

Most importantly,

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

for a special $F = [f_1, \ldots, f_k]$, 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

\[ \text{RatioCut}(V_1, \ldots, V_k) = \text{trace}(F^T L F), \]

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** \( V_1, \ldots, V_k \) from \( F^* \) by distance-based clustering algorithms (e.g. \( k \)-means)
Spectral Clustering vs. $k$-means

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

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

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

- $k$-means algorithm can only find linear decision boundaries
- Spectral clustering allows us to find non-convex boundaries
Variants of Graph Laplacian

Normalized Laplacian:

- \( L = I_n - D^{-1/2}WD^{-1/2} \)
- \( \text{NormalizedCut}(V_1, \ldots, V_k) = \sum_{i=1}^{k} \frac{\text{cut}(V_i, V \setminus V_i)}{\text{vol}(V_i)} \), where
  \( \text{vol}(V_i) = \sum_{j \in V_i} D_{jj} \)

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

- Randomized PCA
- Isomap
- Spectral Embedding
- LLE

<10K samples

决策路径：
- YES: kernel approximation
- NO: NOT WORKING
- NOT WORKING: dimensionality reduction
Dimensionality Reduction: Principal Component Analysis
Principal Component Analysis

$N$ observations: $\{x_i \in \mathbb{R}^D : i = 1 \ldots, N\}$

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 \(\{x_n\}\):

\[
\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n
\]

\[
S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T
\]

\(w\): the direction of the space

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

\[
\frac{1}{N} \sum_{n=1}^{N} \left( w^T x_n - w^T \bar{x} \right)^2 \equiv w^T S w.
\]
PCA: Projection to one dimensional space ($M = 1$)

Goal: maximize the variance of the projected data $\{\text{Proj}_w(x_n)\}$:

$$\arg \max_{w_1: \|w_1\| = 1} w_1^T S w_1$$

- Lagrangian $L(w_1, \lambda_1) = w_1^T S w_1 + \lambda_1 (1 - w_1^T w_1)$
- $\nabla L(w_1, \lambda_1) = 0$ implies that $S w_1^* = \lambda_1 w_1^*$.
- $w_1^*$ is the eigenvector of $S$ corresponding to the largest eigenvalue $\lambda_1^*$, also called the 1-st principal component.
- In general, the $k$-th principal component $w_k^*$ is the eigenvector of $S$ corresponding to the $k$-th largest eigenvalue $\lambda_k^*$.

Dimension reduction:
- $W = [w_1^*, \ldots, w_M^*]$: formed by $M$ principal components.
- $\text{Proj}_W(x) = W^T x$: the projected vector in $M$ dimensional space.
PCA: An Example

A set of digit images

The mean vector $\bar{x}$ 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 $M$:

Original | $M = 1$ | $M = 10$ | $M = 50$ | $M = 250$
--- | --- | --- | --- | ---

Eigenvalue Spectrum:

![Graph](image)
Dimensionality Reduction: Matrix Factorization
Matrix Factorization

A motivating example: recommender systems
Problem Formulation
Latent Feature Space
Existing Methods
# Recommender Systems

## Rating Matrix

<table>
<thead>
<tr>
<th>Users</th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Items</th>
<th>Movie 10</th>
<th>Movie 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hsiang-Fu</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Cho-Jui</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Si Si</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indrjith</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kai-Yang</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Donghyuk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Koge</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: The user ratings are represented in the matrix.*
Matrix Factorization Approach $A \approx WH^T$
Matrix Factorization Approach $A \approx WH^T$
Matrix Factorization Approach

\[
\min_{W \in \mathbb{R}^{m \times k}, H \in \mathbb{R}^{n \times k}} \sum_{(i,j) \in \Omega} (A_{ij} - w_i^T h_j)^2 + \lambda \left( \|W\|_F^2 + \|H\|_F^2 \right),
\]

- \( \Omega = \{(i,j) \mid A_{ij} \text{ is observed}\} \)
- 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 \( W \), \( w_i \)
- the \( j^{th} \) item \( \Rightarrow \) \( j^{th} \) row of \( H \), \( h_j \)
- \( w_i^T h_j \): measures the interaction between \( i^{th} \) user and \( j^{th} \) item.
Latent Feature Space

- Titanic
- Up
- Three Idiots
- Mask
- Men In Black
- Avengers
- Die Hard
- The Dark Knight
- Twilight
- Silent Hill
- Saw
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: $\min_{w_i \in \mathbb{R}^k} \sum_{j \in \Omega_i} (A_{ij} - w_i^T h_j)^2 + \lambda \|w_i\|^2$

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

$$
\begin{pmatrix}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{pmatrix}
= 
\begin{pmatrix}
w_1^T \\
w_2^T \\
w_3^T
\end{pmatrix}
\begin{pmatrix}
H^T
\end{pmatrix}
$$
SGM: Stochastic Gradient Method

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

- \(R_{ij} \leftarrow A_{ij} - w_i^T h_j\)
- \(w_i \leftarrow w_i - \eta(\lambda w_i - R_{ij} h_j),\)
- \(h_j \leftarrow h_j - \eta(\lambda h_j - R_{ij} w_i).\)

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 \(\eta.\)
Coordinate Descent

Update a variable at a time:

$$w_{it} \leftarrow \left( \sum_{j \in \Omega_i} (A_{ij} - w_i^T h_j + w_{it} h_{jt}) h_{jt} \right) / \lambda + \sum_{j \in \Omega_i} h_{jt}^2 .$$

- 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:**
  - \(k\text{NN}, \text{SVM}\), Perceptron, Logistic Regression, Naive Bayes, Decision Trees, Random Forest, AdaBoost, ...

- **Clustering:**
  - \(k\text{-means}, \text{Spectral Clustering}\), Affinity Propagation, Mean-Shift, DBSCAN, Hierarchical Clustering, ...

- **Dimensionality Reduction:**
  - (kernel/sparse) \textbf{PCA}, \textbf{MF}, \textbf{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

\[
\begin{pmatrix}
P_1 & P_2 & P_3 \\
h_1 & h_2 & h_3 \\
w_1^T & w_2^T & w_3^T \\
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{pmatrix}
\]
DSGD: Distributed SGM

\[
\begin{bmatrix}
\mathbf{w}_1^T \\
\mathbf{w}_2^T \\
\mathbf{w}_3^T
\end{bmatrix}
= \begin{bmatrix}
\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3
\end{bmatrix}
\begin{bmatrix}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{bmatrix}
\]
DSGD: Distributed SGM
Parallel Coordinate Descent

Feature-wise Update: CCD++

Rank-one decomposition:

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

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

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

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

\((u^*, v^*)\) is a rank-one approximation of \( \hat{R} \)

- Apply the CCD iteration \( T \) times to obtain \((u^*, v^*)\)
- CCD: item/user-wise update
When $T = 2$

Feature-wise Update: CCD++
Feature-wise Update: CCD++

When $T = 2$

$W$

$H^T$

 Netflix with $k = 40$
Feature-wise Update: CCD++

When $T = 2$

$O(2^T + 1)$ faster than CCD

Netflix with $k = 40$
Feature-wise Update: CCD++

When \( T = 2 \)
Feature-wise Update: CCD++

When $T = 2$

$W$

$H^T$

Cycle through $k$ feature dimensions $O(2^TT + 1)$ faster than CCD

netflix with $k = 40$
Feature-wise Update: CCD++

When $T = 2$

$W$ $H^T$

Netflix with $k = 40$
Feature-wise Update: CCD++

When $T = 2$

$W$

$H^T$

When $T = 2$ Netflix with $k = 40$
Feature-wise Update: CCD++

When $T = 2$

When $T = 2$ faster than CCD

$O((2T+1)^2)$
Feature-wise Update: CCD++

When $T = 2$

- $W$
- $H^T$

Netflix with $k = 40$
Feature-wise Update: CCD++

When $T = 2$

$O(2^T + 1)$ faster than CCD

Netflix with $k = 40$
Feature-wise Update: CCD++

When $T = 2$

$W$ $H^T$

Netflix with $k = 40$ faster than CCD.
Feature-wise Update: CCD++

When $T = 2$

$W$

$H^T$
Feature-wise Update: CCD++

When $T = 2$

- Cycle through $k$ feature dimensions
- $O\left(\frac{2T}{T+1}\right)$ faster than CCD
Problems of Different Scales

\[ W, H, \text{ and } R \text{ fit in the memory of a single computer} \]

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

\[ W, H \text{ or } R \text{ exceeds memory capacity of one computer} \]

- 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 $(u^*, v^*)$.
- Fact: each $u_i$ can be updated independently.

Partition $u$ and $v$ into $p$ sub-vectors.

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

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

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

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.

$$
\begin{bmatrix}
R_{11} & R_{12} & R_{13} \\
R_{21} \\
R_{31}
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
C_1
\end{bmatrix}

\begin{bmatrix}
R_{21} & R_{22} & R_{23} \\
R_{32}
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
C_2
\end{bmatrix}

\begin{bmatrix}
R_{31} & R_{32} & R_{33}
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
C_3
\end{bmatrix}

W \Rightarrow \begin{bmatrix} W^1 & W^2 & W^3 \end{bmatrix}^T

H \Rightarrow \begin{bmatrix} H^1 & H^2 & H^3 \end{bmatrix}^T
Distributed update: computer $C_r$:
- obtains $(u^r, v^r)$ using CCD:
  - computes $u^r$ and broadcasts it
  - computes $v^r$ and broadcasts it
- updates $(\bar{w}_t^r, \bar{h}_t^r) \leftarrow (u^r, v^r)$
References


