Recommendation Systems

GRADIENT DESCENT & SINGULAR VALUE DECOMPOSITION

Review …

- Last time
  - Clustering - mapreduce
  - Recommendation Systems

- Today
  - Decompositions of the Utility Matrix
  - Gradient Descent
  - Singular Value Decomposition

Assignment 1

- Interactive nodes
- ppn > 16
- Running out of memory

Dimensionality reduction

- Utility Matrix, M, is low rank → Singular Value Decomposition

- $M \;\rightarrow n \times m$
- $M = UV$
- $U \rightarrow n \times d, \quad V \rightarrow d \times m$

- How close is UV to M → root mean square error (RMSE)
  - Sqrt of Sum of difference over all nonblank entries

Incremental computation of UV

- Preprocess matrix M
- Start with an initial guess for U, V
- Iteratively update U, V to minimize the RMSE
  - Optimization problem

We are estimating U, V using partial values of M

Preprocessing M

- Normalize for user
  - Subtract average user rating
- Normalize for item
  - Subtract average item rating
- Both
  - Subtract average of user and item rating from m_{ij}

- Need to undo normalization while making predictions ...

Initializing U, V

- Need a good guess
- Some randomness helps
- Initialize all entries to the same value
  - 0 is a good choice if normalized
  - Else, $\sqrt{\frac{a}{d}}$ is a good value
- Ideally start with multiple initial guesses centered around 0

Optimizing

- Gradient descent
- First order approximation
- Update using steps proportional to the negative gradient of the objective function (RMSE)
- Stop when gradient is zero
- Inefficient for large matrices
  - Stochastic Gradient descent
  - Randomized SVD

Gradient Descent

Given a multivariate function $F(x)$, at point p

then, $F(b) < F(a)$, where

$$b = a - \gamma \nabla F(a)$$

for some sufficiently small γ

$$\min \; ||M - UV||$$

UV Decomposition

- $M, U, V,$ and $P = UV$
- Let us optimize for $x = u_{rs}$

$$p_{rj} = \sum_{k=1}^{d} u_{rk} v_{kj} = \sum_{k \neq s} u_{rk} v_{kj} + x v_{sj}$$

$$\mathcal{C} = \sum_{j} \left(m_{rj} - p_{rj}\right)^2 = \sum_{j} \left(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj} - x v_{sj}\right)^2$$

UV Decomposition

- First order optimality → $\partial \mathcal{C}/\partial x = 0$

$$\mathcal{C} = \sum_j \left(m_{rj} - p_{rj}\right)^2 = \sum_j \left(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj} - x v_{sj}\right)^2$$

$$\frac{\partial \mathcal{C}}{\partial x} = \sum_j -2 v_{sj} \left(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj} - x v_{sj}\right) = 0$$

$$x = \frac{\sum_j v_{sj} \left(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj}\right)}{\sum_j v_{sj}^2}$$

UV Decomposition

- Choose elements of U and V to optimize
  - In order
  - Some random permutation
  - Iterate
- Correct way
  - Use expression to compute $\partial \mathcal{C} / \partial \boldsymbol{x}$ at current estimate
    - Expensive when number of unknowns is large $(2\, n\, d)$
  - Use traditional gradient descent

Stochastic Gradient Descent

In cases where the objective function $\mathcal{C}(w)$ can be written in terms of local costs

$$\mathcal{C}(w) = \sum_{n} \mathcal{C}_i(w)$$

For the case of UV decomposition,

$$\mathcal{C} = \sum_{i,j} c(M_{ij}, U_{i*}, V_{*j})$$

Stochastic Gradient Descent

- Traditional gradient descent

$$w \leftarrow w - \lambda \sum_{n} \nabla C_i(w)$$

- In Stochastic GD, approximate true gradient by a single example:

$$w \leftarrow w - \lambda \nabla C_i(w)$$

Stochastic Gradient Descent

- Input: samples Z, initial values $\boldsymbol{U}_0, \boldsymbol{V}_0$
- while not converged do
  - Select a sample $(i, j) \in Z$ uniformly at random
    - $\boldsymbol{U}'_{i*} \leftarrow \boldsymbol{U}_{i*} - \lambda_n N \frac{\partial}{\partial \boldsymbol{U}_{i*}} c(\boldsymbol{M}_{ij}, \boldsymbol{U}_{i*}, \boldsymbol{V}_{*j})$
    - $\boldsymbol{V}_{*j} \leftarrow \boldsymbol{V}_{*j} - \lambda_n N \frac{\partial}{\partial \boldsymbol{V}_{*j}} c(\boldsymbol{M}_{\boldsymbol{ij}}, \boldsymbol{U}_{i*}, \boldsymbol{V}_{*j})$
    - $U_{i*} \leftarrow U'_{i*}$

Singular Value Decomposition

Goal: Given a $m \times n$ matrix A, for large m, n, we seek to compute a rank-k approximation, with $k \ll n$,

$$A \approx U \Sigma V^* = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^*$$

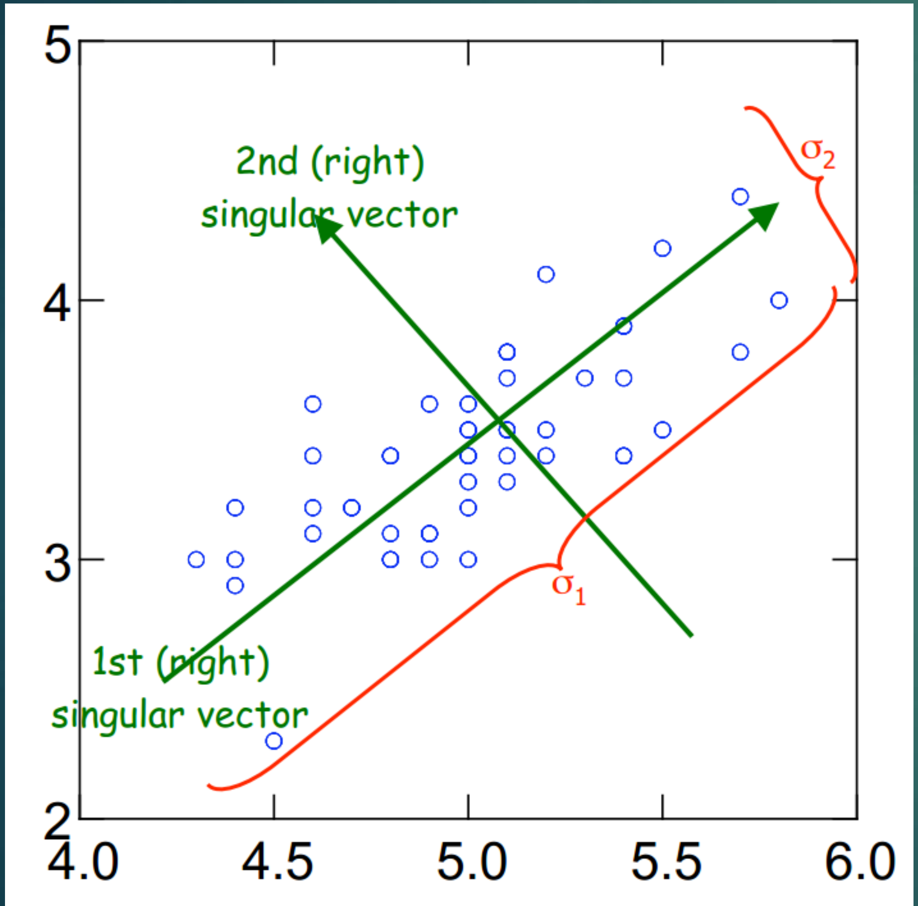
$m \times n$ $\quad$ $m \times k$ $\quad$ $k \times k$ $\quad$ $k \times n$

$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$ are the (approximate) singular values of A
$\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_k$ are orthonormal, the (approximate) left singular vectors of A, and
$\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k$ are orthonormal, the (approximate) right singular vectors of A.

Singular Value Decomposition

- Closely related problems:
  - Eigenvalue decomposition $A \approx V \Lambda V^*$
  - Spanning columns or rows $A \approx C\ U\ R$
- Applications:
  - **Principal Component Analysis**: Form an empirical covariance matrix from some collection of statistical data. By computing the singular value decomposition of the matrix, you find the directions of maximal variance
  - **Finding spanning columns or rows**: Collect statistical data in a large matrix. By finding a set of spanning columns, you can identify some variables that "explain" the data. (Say a small collection of genes among a set of recorded genomes, or a small number of stocks in a portfolio)
  - **Relaxed solutions to k-means clustering**: Relaxed solutions can be found via the singular value decomposition
  - **PageRank**: primary eigenvector

Singular values, intuition



- Blue circles are m data points in $2D$
- The SVD of the $m \times 2$ matrix
  - V_1: 1st (right) singular vector: direction of maximal variance,
  - σ_1: how much of data variance is explained by the first singular vector
  - V_2: 2nd (right) singular vector: direction of maximal variance, after removing projection of the data along first singular vector.
  - σ_2: measures how much of the data variance is explained by the second singular vector

Goal: Given a $m \times n$ matrix A, for large m, n, we seek to compute a rank-k approximation, with $k \ll n$,

$$A \approx U \Sigma V^* = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^*$$

$m \times n$ $\quad$ $m \times k$ $\quad$ $k \times k$ $\quad$ $k \times n$

$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$ are the (approximate) singular values of A
$\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_k$ are orthonormal, the (approximate) left singular vectors of A, and
$\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k$ are orthonormal, the (approximate) right singular vectors of A.

Randomized SVD

1. Draw an $n \times k$ Gaussian random matrix, Ω
2. Form the $m \times k$ sample matrix $Y = A\Omega$
3. Form an $m \times k$ orthonormal matrix Q such that $Y = QR$
4. Form the $k \times n$ matrix $B = Q^*A$
5. Compute the SVD of the small matrix $B = \hat{U}\Sigma V^*$
6. Form the matrix $U = Q\hat{U}$

Computational Costs ?

2,4 → k –Matrix-Vector product
3,5,6 → dense operations on matrices $m \times k, k \times n$

Computational Costs

- If A can fit in RAM
  - Cost dominated by $2mnk$ flops required for steps 2,4
- If A cannot fit in RAM
  - Standard approaches suffer
  - Randomized SVD is successful as long as
    - Matrices of size $m \times k$ and $k \times n$ must fit in RAM
- Parallelization
  - Steps 2,4 permit k-way parallelization

Probabilistic Error Analysis

The error of the method is defined as

$$e_k = \left\|A - \hat{A}_k\right\|$$

e_k is a random variable whose theoretical minimum value is

$$\sigma_{k+1} = \min(\|A - A_k\| : A_k \text{ has rank } k)$$

Ideally, we would like e_k to be close to σ_{k+1} with high probability

Not true, the expectation of $\frac{e_k}{\sigma_{k+1}}$ is large and has very large variance

Oversampling …

Oversample a little. If p is a small integer (think $p = 5$), then we often can bound e_{k+p} by something close to σ_{k+1}

$$\mathbb{E}\|A - \hat{A}_{k+p}\| \leq \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{\frac{1}{2}}$$