Scientific Computation

Spring, 2019

Lecture 14

Notes

- Extra office hour this week: Thursday, 4-5pm, MLC

Plan for last few weeks:

- Lectures 15 and 16: Discrete Fourier transforms and nonlinear time series analysis
- Lectures 17 and 18: Numerical differentiation
- Lectures 19 and 20: Compiling Python code, parallel computing with Python, suggested topics from class

Data analysis

- With PCA, the goal was to extract information from a dataset
- Now, we want to think about filling in information *missing* from a dataset
- There are many different approaches, we will look at just one based on low-rank matrix factorization
- Is this actually useful? Very important in computer graphics, vision, machine learning, and recommender systems
- We will focus on the latter example

Recommender systems

- How do Netflix, Amazon, Facebook, etc… decide what to recommend to you?
- They collect:
  - information about what you like (and dislike)
  - information about what everyone else likes
- And then attempt to predict what you will spend time and/or money on
- How is this data organized? – one example is a ratings matrix

Recommender systems

- **How is this data organized? – one example is a ratings matrix**

	Suits	Sex Education	Friends	Stranger Things	Killing Eve
Don	5	?	1	?	?
Liz	0	4	5	?	?
Kamala	2	?	3	?	5
Beto	1	5	4	5	?

- **How do we fill in the missing entries?**
- **Organizing idea: Need ratings for high-level concepts (e.g. genre) rather than individual movies**
- **Note: in practice, ratings matrices are much, much larger**

Recommender systems

- Let's first think about a simpler case – how do we fill in the missing entries in this matrix: $\mathrm{D} = \begin{bmatrix} 1 & 2 \\ \times & 6 \\ 2 & \times \end{bmatrix}$
- We need to set criteria to assess how well we fill in the data
- Basic idea: Fill in the data without increasing the "information" or introducing "new trends"

Recommender systems

- Let's first think about a simpler case – how do we fill in the missing entries in this matrix: $D = \begin{bmatrix} 1 & 2 \\ \times & 6 \\ 2 & \times \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 2 & 4 \end{bmatrix}$
- We need to set criteria to assess how well we fill in the data
- Basic idea: Fill in the data without increasing the "information" or introducing "new trends"
- If we maximized variance like in PCA, we would be doing the opposite and inventing trends
- We could think about minimizing variance, but a simpler closely-related idea is to minimize the *rank*
    - Rank(D) = dimension of column space of D = dimension of row space of D
    - In the example above, the rank-1 estimate would be as above

Rank minimization

$$D = \begin{bmatrix} 1 & 2 \\ \times & 6 \\ 2 & \times \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 2 & 4 \end{bmatrix}$$

- **With this approach, we have estimated “ratings” without introducing new information or trends**
- **Basic idea of matrix factorization-based recommender systems:**
  - **Complete the ratings matrix so that:**
    - **Changes to existing entries are “small”**
    - **The rank of the resulting matrix is minimized**

Rank minimization

$$D = \begin{bmatrix} 1 & 2 \\ \times & 6 \\ 2 & \times \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 2 & 4 \end{bmatrix}$$

- **With this approach, we have estimated "ratings" without introducing new information or trends**
- **Basic idea of matrix factorization-based recommender systems:**
  - **Complete the ratings matrix so that:**
    - **Changes to existing entries are "small"**
    - **The rank of the resulting matrix is minimized**
  - **We will look at an approach based on the SVD**
    - **A SVD is a sum of rank-1 matrices**
    - **Rank(A) = number of non-zero singular values of A**

Rank minimization

- **This is an optimization problem, and we first need to define a *cost function***
- **Motivating example:**

  Find "completed" ratings matrix, A**, such that rank(**A**) is minimized and** A = R **for all valid entries in** R

- **The rank is discrete, and it is easier to solve continuous optimization problems with differentiable cost functions (NP hard to minimize rank while leaving existing ratings unchanged)**
- **Instead of minimizing rank, we can maximize the sum of the singular values of** A **(the rank is the number of non-zero singular values):**

Find A such that $|A|_*$ is minimized and $A_{i,j} = R_{i,j}$ for $(i, j) \in \Omega$.
Ω is the set of indices for which ratings have been specified in R, and $|A|_*$ is the sum of the singular values of A.

Rank minimization

- **For numerical calculations, the missing ratings are typically replaced with a number, say, -1000 or something that cannot be mistaken for an actual rating**

- **We then need an auxiliary matrix to help "remove" these numbers:**

$$E_{i,j} = \begin{cases} 0, & (i,j) \in \Omega \\ R_{i,j} - A_{i,j} & (i,j) \notin \Omega \end{cases}$$

- **So, our optimization problem requires:** R - A - E = 0

- **How do we enforce this constraint? With a matrix of *Lagrange multipliers***, Y, **which must be found as part of the problem**

- **Our provisional cost function (Lagrangian) is:**

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M} \sum_{j=1}^{N} Y_{ij} \left(R_{ij} - A_{ij} - E_{ij} \right)$$

Rank minimization

- **Our provisional cost function (Lagrangian) is:**

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right)$$

- **We add one more *regularizing* term:**

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

  - **This reduces the likelihood of large ratings being placed in** A
  - **And introduces a new parameter,** μ **, that must be determined**
  - **But it also allows us to leverage a powerful optimization result (shown later)**

Rank minimization

- **We can now state our "full" optimization problem:**
  Find A**,** E**,** Y**,** μ, **such that the Lagrangian (below) is minimized:**

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

- **Next step: construct method to solve the optimization problem.**

Rank minimization

- **We can now state our "full" optimization problem:**
  Find A**,** E**,** Y**,** μ**, such that the Lagrangian (below) is minimized:**

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

- **Next step: construct method to solve the optimization problem.**

- **We could use any standard gradient-based optimization method (e.g. a optimizer in scipy.minimize)**
  - **This requires providing the derivatives of the Lagrangian with respect to the entries of** A**,** E**,** Y**, and** μ

- **We will take a different approach here. We use an iterative approach and update the four parameters sequentially. This works particularly well for large ratings matrices with low rank**

Rank minimization

- **We can now state our "full" optimization problem:**
  Find A, E, Y, μ, such that the Lagrangian (below) is minimized:

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

- **Step 0: Provide initial guesses for A, E, Y, μ**

- **Given estimates for these parameters at iteration k, there is a four-stage process to obtain improved estimates at iteration k+1**

- **Stage 1:** $\mathrm{A}_{k+1} = \arg\min_{\mathrm{A}} \mathscr{L}\left(\mathrm{A}, \mathrm{E}_k, \mathrm{Y}_k, \mu_k\right)$

- **"Find A such that the Lagrangian is minimized with , E, Y, μ held fixed to their values at iteration k"**
- **A recent result from optimization theory – this problem is solved with:**

$$svd(R - E_k + \tfrac{1}{\mu_k} Y_k) = U\Sigma V^T$$

$$A_{k+1} = U\tilde{\Sigma}_{\mu_k} V^T$$

Rank minimization

- **Stage 1:** $A_{k+1} = \arg\min_A \mathscr{L}(A, E_k, Y_k, \mu_k)$

- **"Find A such that the Lagrangian is minimized with , E, Y, μ held fixed to their values at iteration k"**
- **A recent result from optimization theory – this problem is solved with:**

$$svd(R - E_k + \frac{1}{\mu_k} Y_k) = U \Sigma V^T$$

$$A_{k+1} = U \tilde{\Sigma}_{\mu_k} V^T$$

- **Here, we are iteratively "reducing" the singular values of A (and reducing its rank) with:** $\tilde{\sigma}^{(i)} = max(\sigma^{(i)} - \frac{1}{\mu_k}, 0)$

- **So singular values of** $(R - E_k + \frac{1}{\mu_k} Y_k)$ **which are smaller than 1/μ are set to zero, and all others are reduced by 1/μ**

Rank minimization

- **We can now state our "full" optimization problem:**
  Find A, E, Y, μ, such that the Lagrangian (below) is minimized:

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

- **Step 0: Provide initial guesses for A, E, Y, μ**

- **Given estimates for these parameters at iteration k, there is a four-stage process to obtain improved estimates at iteration k+1**

- **Stage 2:** $\mathrm{E}_{k+1} = \arg\min_{\mathrm{E}} \mathscr{L}\left(\mathrm{A}_{k+1}, \mathrm{E}, \mathrm{Y}_k, \mu_k\right)$

- **Differentiating the Lagrangian with respect to E and setting the result to zero:**

$$E_{k+1} = \frac{1}{2\mu_k} Y_k + R - A_{k+1}, \;\; (i,j) \notin \Omega$$

Rank minimization

- **We can now state our "full" optimization problem:**
  Find A, E, Y, μ, such that the Lagrangian (below) is minimized:

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

- **Step 0: Provide initial guesses for A, E, Y, μ**

- **Given estimates for these parameters at iteration k, there is a four-stage process to obtain improved estimates at iteration k+1**

- **Stage 3:** $Y_{k+1} = Y_k + \mu_k (D - A_{k+1} - E_{k+1})$.

- **This is a *relaxation* of Y based on how well constraint is satisfied and is a companion result to what we used for stage 1**

Rank minimization

- **We can now state our "full" optimization problem:**
  Find A**,** E**,** Y**,** μ, **such that the Lagrangian (below) is minimized:**

$$\mathcal{L} = |A|_* + \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij}\left(R_{ij} - A_{ij} - E_{ij}\right) + \mu|R - A - E|_F$$

- **Step 0: Provide initial guesses for** A**,** E**,** Y**,** μ

- **Given estimates for these parameters at iteration** k**, there is a four-stage process to obtain improved estimates at iteration** k+1

- **Stage 4:** $\mu_{k+1} = \rho\mu_k,\ \rho > 1$

- **At larger** μ**, our approximation is closer to the true minimum of the Lagrangian, so we gradually increase** μ **each iteration. Increase more gradually if there are more missing entries**

Rank minimization

- See *rec.py* for Python implementation of this method
- Applying it to our small example:

	Suits	Sex Education	Friends	Stranger Things	Killing Eve
Don	5	1	1	1	2
Liz	0	4	5	4	3
Kamala	2	4	3	4	5
Beto	1	5	4	5	5

Recommender systems

- There are a broad range of matrix factorization approaches
  - For example, estimate r column vectors and r row vectors that minimize an error based on Frobenius norm

- Different optimization methods can then be applied for each of these approaches
  - Stochastic gradient descent is popular for very large ratings matrices

- Underlying idea, as in PCA, and linear amplification problems: singular values and corresponding eigenvectors provide essential information about key trends "hidden" in data