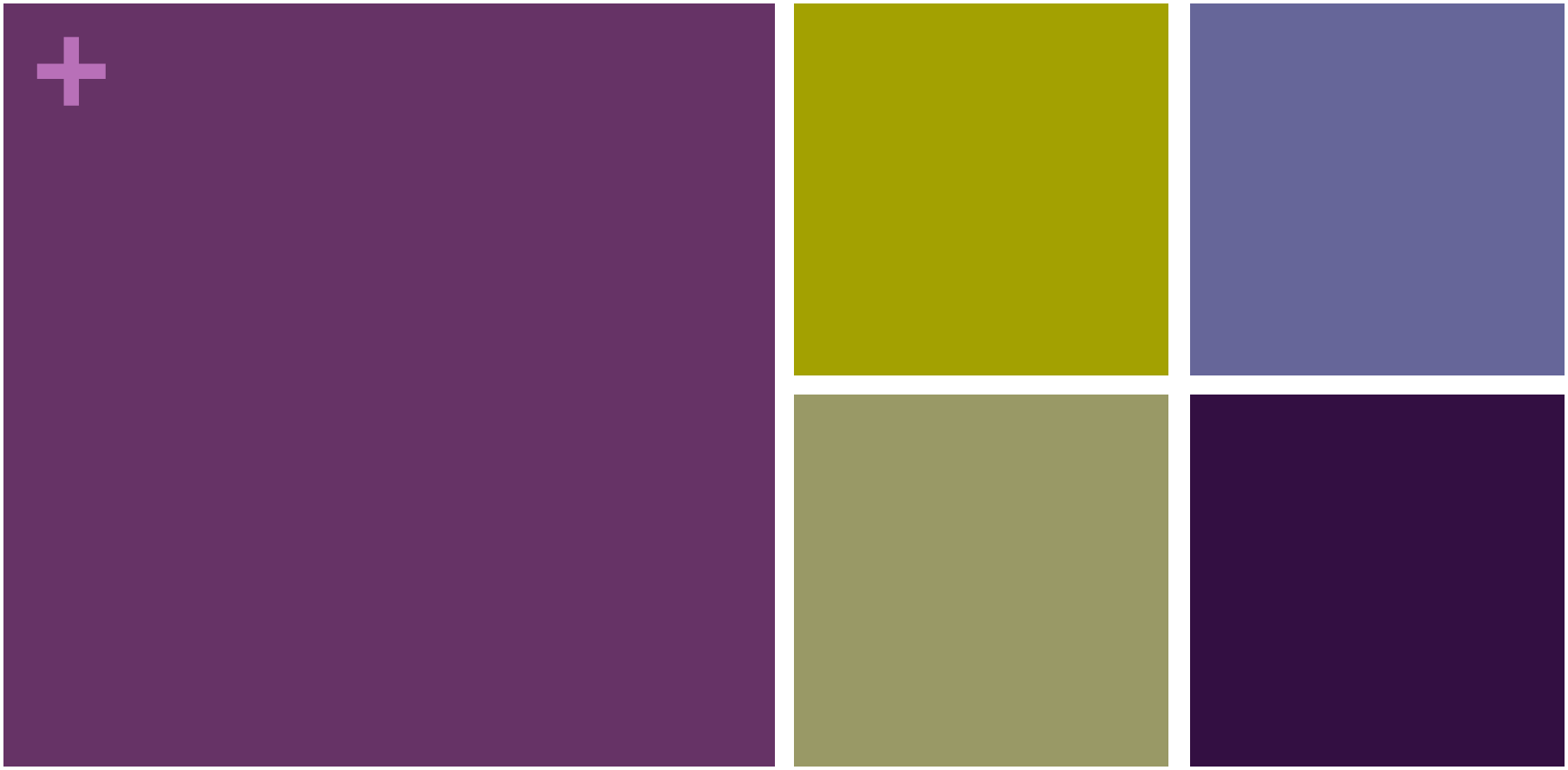


Recommender Systems Factorization and Gradient Descent

Professor Robin Burke

Spring 2019

Outline

- Syllabus status
- Factorization via gradient descent
  - adding regularization
- Alternating least squares
- User and item biases
  - FunkSVD

Syllabus status

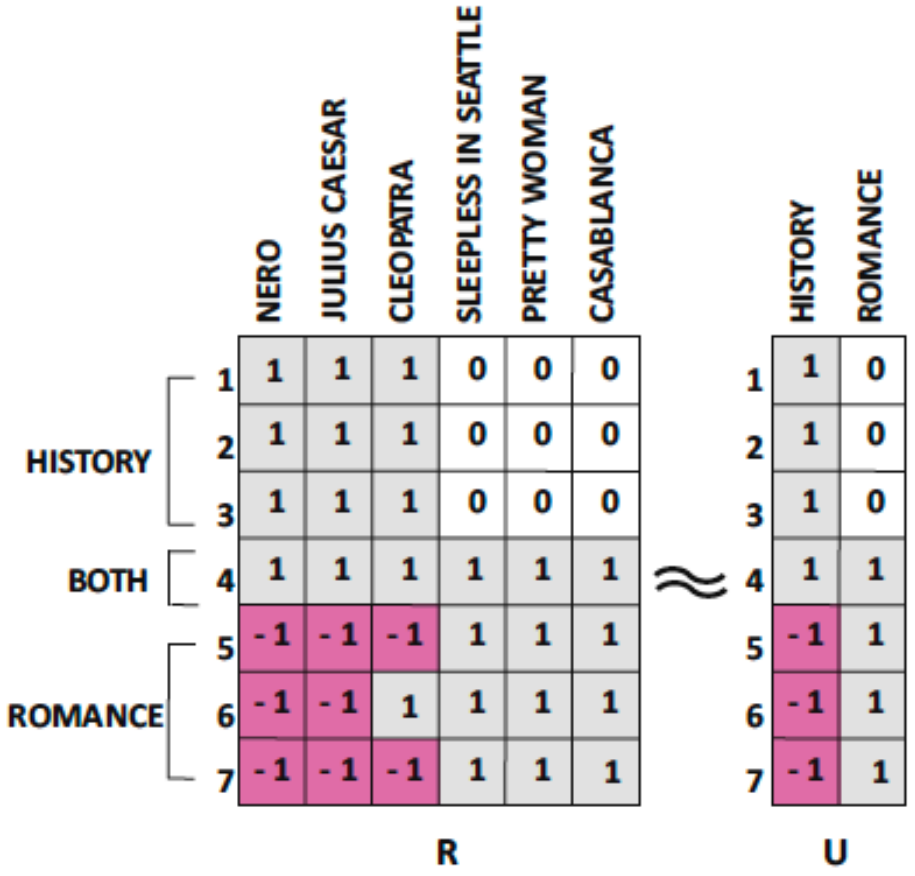
- We are about 1 week behind
  - big deal?
- Content-based recommendation is now delayed considerably
- Next weeks
  - 2/19: Domain presentations
  - 2/26: Project brainstorming
  - 3/5: Evaluation
- I will issue a new syllabus next week

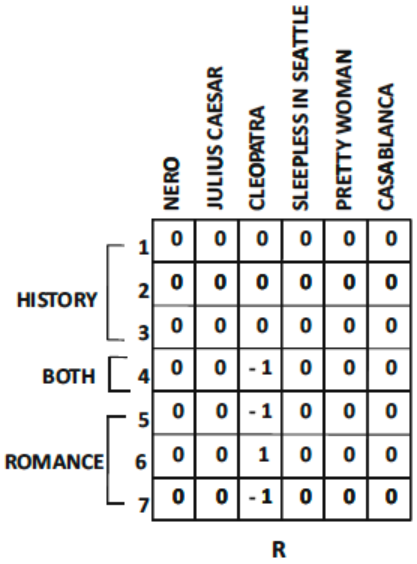
Low rank approximation

- We can approximate R by any set of vectors
  - $R \approx UV^T$
  - With some error $||R - UV^T||^2$

L2 matrix norm: square all entries and sum

Recommendation Example





Factors

- Each row $\mathbf{u_i}$ of U is a "user factor"
- Each row $\mathbf{v_j}$ of V is an "item factor"
- $\check{r}_{ij} = \mathbf{u_i} \cdot \mathbf{v_j}$
  - This is the estimated rating
  - Very efficient to compute
- Might be explanatory
  - But typically the factors are not readily interpretable
  - Other formulations (NMF, PLSA) slightly better

$$r_{ij} \approx \sum_{s=1}^{k} u_{is} \cdot v_{js}$$
$$= \sum_{s=1}^{k} (Affinity\ of\ user\ i\ to\ concept\ s) \times (Affinity\ of\ item\ j\ to\ concept\ s)$$

Unconstrained Matrix Factorization

- Perform the $R = UV^T$ factorization without any restrictions on the results
  - Must pick the number of factors

$$\text{Minimize } J = \frac{1}{2}||R - UV^T||^2$$

$$\text{subject to:}$$

$$\text{No constraints on } U \text{ and } V$$

- Cannot be performed with missing entries
  - Norm undefined
- Need a different objective

With missing values

- e_{ij} is the observed error on an entry we have

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2$$

subject to:

No constraints on U and V

- No closed form solution to this
  - But it is convex!
  - So there must be a minimum
  - A set of u_{is} and v_{js} values that minimizes the error

Gradient descent solution

- Requires that we calculate the gradient with respect to the decision variables u_{iq} and v_{jq}
  - Where i ranges over all users, j ranges over all items
  - q ranges over all factors
  - Total of (mk + nk) variables in the gradient vector

Chain rule

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

$$= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

$$= \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

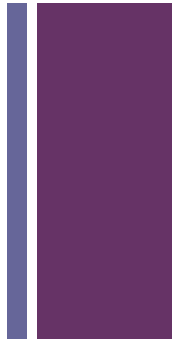
Algorithm

Algorithm *GD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i,j) : r_{ij} \text{ is observed}\}$;
  while not(convergence) **do**
  begin
    Compute each error $e_{ij} \in S$ as the observed entries of $R - UV^T$;
    for each user-component pair (i,q) **do** $u_{iq}^+ \Leftarrow u_{iq} + \alpha \cdot \sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq}$;
    for each item-component pair (j,q) **do** $v_{jq}^+ \Leftarrow v_{jq} + \alpha \cdot \sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq}$;
    for each user-component pair (i,q) **do** $u_{iq} \Leftarrow u_{iq}^+$;
    for each item-component pair (j,q) **do** $v_{jq} \Leftarrow v_{jq}^+$;
    Check convergence condition;
  end
end

Descent controlled by learning rate

Simultaneous update

Example



Efficiency

- We run through the entire data set to compute the gradient at the given point
- All of the information from the data about the direction to the solution point
- We have to do this over and over
- Fairly slow
  - think millions of user/item/rating triples

Latent factor space

- You can compute a solution for any number of factors
- What is the right number?
- Very hard to determine in advance
- Trial and error / grid search

A factorization with a very small number of latent factors is

- A: A high-bias solution (likely to underfit)
- B: A high-variance solution (likely to overfit)
- C: Neither

Your colleague says "let's do matrix factorization with the number of latent factors = the number of items". You say:

- A: That's a great idea. It'll be very accurate. Might take a long time to compute though
- B: That's the dumbest idea I've ever heard. It will never converge.
- C: That's the dumbest idea I've ever heard. It will totally overfit the data.
- D: That's a great idea. We will be able to compute U and V very easily.

Stochastic gradient descent

- Let's assume that there is a lot of redundancy in the data
  - that's what we mean by latent factors after all
- Then many r_{ij} values have the same information
- Let's update on a single rating at a time
- Stochastic version
  - Randomly choose r_{ij} (without replacement)
  - Update
  - Repeat

Algorithm

Algorithm *SGD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i,j) : r_{ij} \text{ is observed}\}$;
  while not(convergence) **do**
  begin
    Randomly shuffle observed entries in S;
    for each $(i,j) \in S$ in shuffled order **do**
    begin
      $e_{ij} \Leftarrow r_{ij} - \sum_{s=1}^{k} u_{is} v_{js}$;
      for each $q \in \{1 \ldots k\}$ **do** $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq}$;
      for each $q \in \{1 \ldots k\}$ **do** $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq}$;
      for each $q \in \{1 \ldots k\}$ **do** $u_{iq} = u_{iq}^{+}$ and $v_{jq} = v_{jq}^{+}$;
    end
    Check convergence condition;
  end
end

Single rating update

Why do this?

- If the data is very large
  - And has a lot of redundancy
- Many updates will point in the same direction
- Stochastic version moves more quickly through the space
  - Although noisily
- Possible to do something in between
  - Randomly chosen mini-batch
- Possible to adjust learning rate through the process
  - Bold driver algorithm

Visualization

- https://www.youtube.com/watch?v=HvLJUsEc6dw

Regularization

- Need to avoid overfitting
  - As we saw with SLIM
- If R is sparse
  - The factors could fit to the noise in the data
  - Lower performance on test data
- Solution
  - Create a bias towards lower-valued factors
  - Typically $\frac{\lambda}{2}(\|U\|^2 + \|V\|^2)$
    - Alternatively, different λ for U and V
- This changes our gradients

Gradients

Regularization of U

Regularization of V

- New objective

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

$$= \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

- New gradients

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1\ldots m\}, q \in \{1\ldots k\}$$

$$= \sum_{j:(i,j)\in S}(e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1\ldots m\}, q \in \{1\ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1\ldots n\}, q \in \{1\ldots k\}$$

$$= \sum_{i:(i,j)\in S}(e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1\ldots n\}, q \in \{1\ldots k\}$$

In the gradient with respect to uiq, there is no term for the regularization of V. Why is this?

$$= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

- A: Error in the textbook. We should add a λv_{jq} to this expression.
- B: The $||V||^2$ is constant with respect to changes in u_{iq}
- C: Error in the textbook. The regularization term shouldn't appear in the gradient calculation.

Last week, we talked about the difference between L1 regularization and L2 regularization. This algorithm uses L2 regularization. What is the impact of this choice?

- A: It will tend to make the elements of the U and V matrices small
- B: It will tend to make the U and V matrices sparse
- C: It ensures that the gradient of the loss function can be computed
- D: A and B
- E: A and C

Regularized version

- Can use batch or stochastic gradient descent
- Need to pick lambda
  - Not easy to do this in a principled way
- Try different values and see what works
  - "grid search"

Bayesian Probabilistic Matrix Factorization

- Salakhutdinov & Mnih. ICML, 2008.
- Does this by learning the lambdas and the factors at the same time
  - Using Markov chain monte carlo (MCMC)
  - Implemented in some libraries, but not in LKPy
- Side benefit
  - The values in the matrices can be intrepreted as probabilities
  - P(u|s) = the probability that user u is associated with latent factor s

Alternating least squares

- Instead of solving for U and V at the same time
- Solve for them separately
  - First U
    - Holding V constant
  - Then V
    - Holding U constant
  - Repeat back and forth – alternating!

Alternating least squares

- Start with random U and V matrices
- Multiple instances of least squares optimization

$$\sum_{i:(i,j)\in S}(r_{ij} - \sum_{s=1}^{k} u_{is}v_{js})^2$$

- Change which set of vectors are constant
  - And which are learned
- Less sensitive to initial conditions and parameter settings
- Can use regularization (ridge regression usually)
- A bit less efficient than SGD
  - Need to invert matrices for least squares solution

Coordinate descent

- Talked a little about this with SLIM
- Can also be applied to ALS
- Pick only a single u_{iq} variable to solve for
  - The whole optimization objective is just a quadratic function
    - $y = ax^2+bx+c$
  - Minimum = -b/2a
  - Treat this as the update for u_{iq}
    - Not a gradient! Move straight to the minimum
- Work through all the variables as in regular gradient descent
- May have to do this several times

LKPy implementation

- Note LAPACK package used
  - http://www.netlib.org/lapack/explore-html/dc/de9/group__double_p_osolve_ga9ce56acceb70eb6484a768eaa841f70d.html
- Not coordinate descent

Have you ever programmed in Fortran?

- A: Yes
- B: No
- C: What's Fortran?