Collaborative Filtering

Mathieu Lagrange

Based on slides by Lester Mackey and Aleksandr Simma

CENTRALE NANTES

March 6, 2019

Course website: https://github.com/mathieulagrange/datasim

Outline

1. Problem Formulation
   - Centering
   - Shrinkage
2. Preliminaries
   - Naive Bayes
   - KNN
3. Classification/Regression
   - SVD
   - Factor Analysis
4. Low Dimensional Matrix Factorization
   - References

What is Collaborative Filtering?

Group of users

Group of items

What is Collaborative Filtering?

Group of users

Group of items

- Observe some user-item preferences
- Predict new preferences:

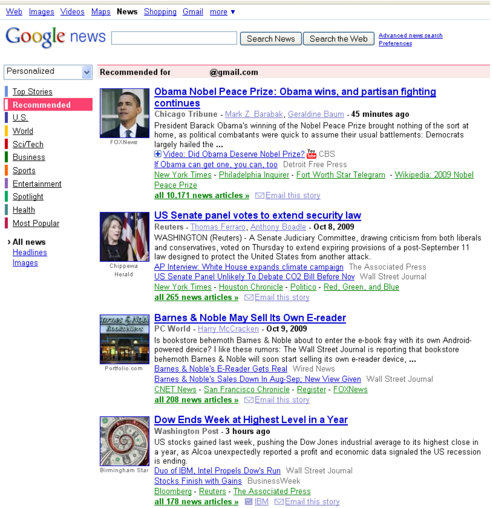
Does Bob like strawberries???

Collaborative Filtering in the Wild...

Amazon.com recommends products based on purchase history

Linder et al., 2003

Collaborative Filtering in the Wild...



- **Google News** recommends new articles based on click and search history
- Millions of users, millions of articles

Das et al., 2007

Collaborative Filtering in the Wild...

Netflix predicts other "Movies You'll ♡" based on past numeric ratings (1-5 stars)

- Recommendations drive 60% of Netflix's DVD rentals
- Mostly smaller, independent movies (Thompson 2008)

http://www.netflix.com

Collaborative Filtering in the Wild...

- **Netflix Prize:**
  Beat Netflix recommender system, using Netflix data → Win \$1 million
- **Data:**
  480,000 users
  18,000 movies
  100 million observed ratings = only 1.1% of ratings observed

"The Netflix Prize seeks to substantially improve the accuracy of predictions about how much someone is going to love a movie based on their movie preferences."

What is Collaborative Filtering?

Insight: Personal preferences are correlated

- If Jack loves A and B, and Jill loves A, B, and C, then Jack is more likely to love C

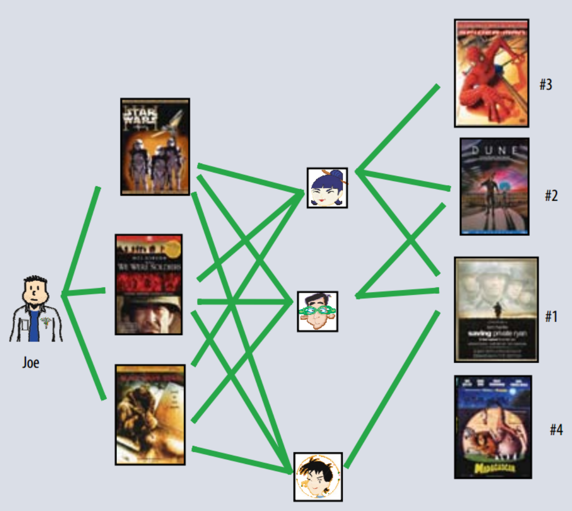
Collaborative Filtering Task

- Discover patterns in observed preference behavior (e.g. purchase history, item ratings, click counts) across community of users
- Predict new preferences based on those patterns

Does not rely on item or user attributes (e.g. demographic info, author, genre)

- Content-based filtering: complementary approach

What is Collaborative Filtering?



What is Collaborative Filtering?

Given:

- Users $u \in \{1, \ldots, U\}$
- Items $i \in \{1, \ldots, M\}$
- Training set $\mathcal{T}$ with observed, real-valued preferences r_{ui} for some user-item pairs (u, i)
  - r_{ui} = e.g. purchase indicator, item rating, click count ...

Goal: Predict unobserved preferences

- Test set $\mathcal{Q}$ with pairs (u, i) not in $\mathcal{T}$

View as matrix completion problem

- Fill in unknown entries of sparse preference matrix

$$\mathbf{R} = \underbrace{\left.\begin{bmatrix} ? & ? & 1 & \ldots & 4 \\ 3 & ? & ? & \ldots & ? \\ ? & 5 & ? & \ldots & 5 \end{bmatrix}\right\} U \text{ users}}_{M \text{ items}}$$

What is Collaborative Filtering?

Measuring success

- Interested in error on unseen test set Q, not on training set
- For each (u, i) let r_{ui} = true preference, $\hat{r}_{ui}$ = predicted preference
- Root Mean Square Error
  - $\text{RMSE} = \sqrt{\frac{1}{|Q|} \sum_{(u,i) \in Q} (r_{ui} - \hat{r}_{ui})^2}$
- Mean Absolute Error
  - $\text{MAE} = \frac{1}{|Q|} \sum_{(u,i) \in Q} |r_{ui} - \hat{r}_{ui}|$
- Ranking-based objectives
  - e.g. What fraction of true top-10 preferences are in predicted top 10?

Centering Your Data

- What?
  - Remove bias term from each rating before applying CF methods: $\tilde{r}_{ui} = r_{ui} - b_{ui}$
- Why?
  - Some users give systematically higher ratings
  - Some items receive systematically higher ratings
  - Many interesting patterns are in variation around these systematic biases
  - Some methods assume mean-centered data
    - Recall PCA required mean centering to measure variance around the mean

Centering Your Data

- What?
  - Remove bias term from each rating before applying CF methods: $\tilde{r}_{ui} = r_{ui} - b_{ui}$
- How?
  - Global mean rating
    - $b_{ui} = \mu := \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} r_{ui}$
  - Item's mean rating
    - $b_{ui} = b_i := \frac{1}{|R(i)|} \sum_{u\in R(i)} r_{ui}$
    - $R(i)$ is the set of users who rated item i
  - User's mean rating
    - $b_{ui} = b_u := \frac{1}{|R(u)|} \sum_{i\in R(u)} r_{ui}$
    - $R(u)$ is the set of items rated by user u
  - Item's mean rating + user's mean deviation from item mean
    - $b_{ui} = b_i + \frac{1}{|R(u)|} \sum_{i\in R(u)} (r_{ui} - b_i)$

Shrinkage

- What?
  - Interpolating between an estimate computed from data and a fixed, predetermined value
- Why?
  - Common task in CF: Compute estimate (e.g. a mean rating) for each user/item
  - Not all estimates are equally reliable
  - Some users have orders of magnitude more ratings than others
  - Estimates based on fewer datapoints tend to be noisier

$$\mathbf{R} = \begin{array}{lccccccc} & A & B & C & D & E & F & \text{User mean} \\ \textit{Alice} & 2 & 5 & 5 & 4 & 3 & 5 & 4 \\ \textit{Bob} & 2 & ? & ? & ? & ? & ? & 2 \\ \textit{Craig} & 3 & 3 & 4 & 3 & ? & 4 & 3.4 \end{array}$$

- Hard to trust mean based on one rating

Shrinkage

- What?
  - Interpolating between an estimate computed from data and a fixed, predetermined value
- How?
  - e.g. Shrunk User Mean:

$$\tilde{b}_u = \frac{\alpha}{\alpha + |R(u)|} * \mu + \frac{|R(u)|}{\alpha + |R(u)|} * b_u$$

  - μ is the global mean, α controls degree of shrinkage
  - When user has many ratings, $\tilde{b}_u \approx$ user's mean rating
  - When user has few ratings, $\tilde{b}_u \approx$ global mean rating

$\mathbf{R} =$

	A	*B*	*C*	*D*	*E*	*F*	User mean	Shrunk mean
Alice	2	5	5	4	3	5	4	3.94
Bob	2	?	?	?	?	?	2	2.79
Craig	3	3	4	3	?	4	3.4	3.43

Global mean $\mu = 3.58$, $\alpha = 1$

Classification/Regression for CF

Interpretation: CF is a set of M classification/regression problems, one for each item

- Consider a fixed item i
- Treat each user as incomplete vector of user's ratings for all items except i: $\vec{r}_u = (3, ?, ?, 4, ?, 5, ?, 1, 3)$
- Class of each user w.r.t. item i is the user's rating for item i (e.g. $1, 2, 3, 4,$ or 5)
- Predicting rating $r_{ui} \equiv$ Classifying user vector $\vec{r}_u$

Classification/Regression for CF

Approach:

- Choose your favorite classifier/regression algorithm
- Train separate predictor for each item
- To predict r_{ui} for user u and item i, apply item i's predictor to vector of user u's incomplete ratings vector

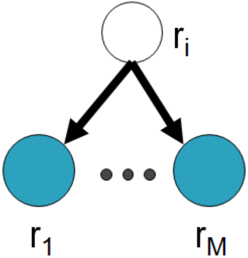
Pros:

- Reduces CF to a well-known, well-studied problem
- Many good prediction algorithms available

Cons:

- Predictor must handle missing data (unobserved ratings)
- Training M independent predictors can be expensive
- Approach may not take advantage of problem structure
  - Item-specific subproblems are often related

Naive Bayes Classifier



- Treat distinct rating values as classes
- Consider classification for item i
- Main assumption
  - For any items $j \neq k \neq i$, r_j and r_k are conditionally independent given r_i
  - When we know rating r_{ui} all of a user's other ratings are independent
- Parameters to estimate
  - Prior class probabilities: $P(r_i = v)$
  - Likelihood: $P(r_j = w | r_i = v)$

Naive Bayes Classifier

Train classifier with all users who have rated item i

- Use counts to estimate prior and likelihood

$$P(r_i = v) = \frac{\sum_{u=1}^{U} \mathbf{1}(r_{ui} = v)}{\sum_{w=1}^{V} \sum_{i=1}^{U} \mathbf{1}(r_{ui} = w)}$$

$$P(r_j = w | r_i = v) = \frac{\sum_{u=1}^{U} \mathbf{1}\left(r_{ui} = v, r_{uj} = w\right)}{\sum_{z=1}^{V} \sum_{u=1}^{U} \mathbf{1}\left(r_{ui} = v, r_{uj} = z\right)}$$

- Complexity
  - $O(\sum_{u=1}^{U} |R(u)|^2)$ time and $O(M^2 V^2)$ space for all items

Predict rating for (u, i) using posterior

$$P(r_{ui} = v | r_{u1}, \ldots, r_{uM}) = \frac{P(r_{ui} = v) \prod_{j \neq i} P(r_{uj} | r_{ui} = v)}{\sum_{w=1}^{V} P(r_{ui} = w) \prod_{j \neq i} P(r_{uj} | r_{ui} = w)}$$

Naive Bayes Summary

Pros:

- Easy to implement
- Off-the-shelf implementations readily available

Cons:

- Large space requirements when storing parameters for all M predictors
- Makes strong independence assumptions
- Parameter estimates will be noisy for items with few ratings
    - E.g. $P(r_j = w | r_i = v) = 0$ if no user rated both i and j

Addressing cons:

- Tie together parameter learning in each item's predictor
- Shrinkage/smoothing is an example of this

K Nearest Neighbor Methods

Most widely used class of CF methods

- Flavors: Item-based and User-based
- Represent each item as incomplete vector of user ratings: $\vec{r}_{.i} = (3, ?, ?, 4, ?, 5, ?, 1, 3)$
- To predict new rating r_{ui} for query user u and item i:
  1. Compute similarity between i and every other item
  2. Find K items rated by u most similar to i
  3. Predict weighted average of similar items' ratings
- Intuition: Users rate similar items similarly.

KNN: Computing Similarities

How to measure similarity between items?

- Cosine similarity

$$S(\vec{r}_{.i}, \vec{r}_{.j}) = \frac{\langle \vec{r}_{.i}, \vec{r}_{.j} \rangle}{\|\vec{r}_{.i}\| \|\vec{r}_{.j}\|}$$

- Pearson correlation coefficient

$$S(\vec{r}_{.i}, \vec{r}_{.j}) = \frac{\langle \vec{r}_{.i} - \mathrm{mean}(\vec{r}_{.i}), \vec{r}_{.j} - \mathrm{mean}(\vec{r}_{.j}) \rangle}{\|\vec{r}_{.i} - \mathrm{mean}(\vec{r}_{.i})\| \|\vec{r}_{.j} - \mathrm{mean}(\vec{r}_{.j})\|}$$

- Inverse Euclidean distance

$$S(\vec{r}_{.i}, \vec{r}_{.j}) = \frac{1}{\|\vec{r}_{.i} - \vec{r}_{.j}\|}$$

Problem: These measures assume complete vectors

Solution: Compute over subset of users rated by both items

Complexity: $O(\sum_{u=1}^{U} |R(u)|^2)$ time

KNN: Choosing K neighbors

How to choose K nearest neighbors?

- Select K items with largest similarity score to query item i

Problem: Not all items were rated by query user u

Solution: Choose K most similar items rated by u

Complexity: $O(min(KM, M \log M))$

Herlocker et al., 1999

KNN: Forming Weighted Predictions

Predicted rating for query user u and item i

- $N(i;u)$ is the *neighborhood* of item i for user u
  - i.e. the K most similar items rated by u
- $\hat{r}_{ui} = b_{ui} + \sum_{N(i;u)} w_{ij}(r_{uj} - b_{uj})$

How to choose weights for each neighbor?

- Equal weights: $w_{ij} = \frac{1}{|N(i;u)|}$
- Similarity weights: $w_{ij} = \frac{S(i,j)}{\sum_{j \in N(i;u)} S(i,j)}$ (Herlocker et al., 1999)
- Learn optimal weights for each user (Bell and Koren, 2007)
- Learn optimal global weights (Koren, 2008)

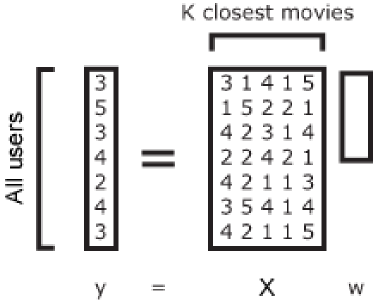
Complexity: $O(K)$

KNN: User Optimized Weights

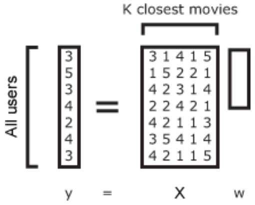
Intuition: For a given query user u and item i, choose weights that best predict other known ratings of item i using only $N(i;u)$:

$$\min_{\mathbf{w}_{i\cdot}} \sum_{s \in R(i), s \neq u} \left(r_{si} - \sum_{j \in N(i;u)} w_{ij} r_{sj} \right)^2$$

With no missing ratings, this is a linear regression problem:



KNN: User Optimized Weights



Bell and Koren, 2007

- Optimal solution: $w = A^{-1}b$ for $A = X^TX, b = X^Ty$
- Problem: X contains missing entries
  - Not all items in $N(i;u)$ were rated by all users
- Solution: Approximate A and b

$$\hat{A}_{jk} = \frac{\sum_{s \in R(j) \cap R(k)} r_{sj} r_{sk}}{|R(j) \cap R(k)|}$$

$$\hat{b}_k = \frac{\sum_{s \in R(i) \cap R(k)} r_{si} r_{sk}}{|R(i) \cap R(k)|}$$

$$\hat{w} = \hat{A}^{-1}\hat{b}$$

- Estimates based on users who rated each pair of items

KNN: User Optimized Weights

Benefits

- Weights optimized for the task of rating prediction
  - Not just borrowed from the neighborhood selection phase
- Weights not constrained to sum to 1
  - Important if all nearest neighbors are dissimilar
- Weights derived simultaneously
  - Accounts for correlations among neighbors
- Outperforms KNN with similarity or equal weights
- Can compute entries of $\hat{A}$ and $\hat{b}$ offline in parallel

Drawbacks

- Must solve additional *KxK* system of linear equations per query

Bell and Koren, 2007

KNN: Globally Optimized Weights

Consider the following KNN prediction rule for query (u, i):

$$\hat{r}_{ui} = b_{ui} + |N(i;u)|^{-\frac{1}{2}} \sum_{j \in N(i;u)} w_{ij}(r_{uj} - b_{uj})$$

Could learn a single set of KNN weights w_{ij}, shared by all users, that minimize regularized MSE:

$$E = \frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} \frac{1}{2}(\hat{r}_{ui} - r_{ui})^2 + \lambda \sum_{i=1}^{M} \sum_{j=1}^{M} \frac{1}{2} w_{ij}^2 = \frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} E_{ui}$$

Optimize objective using **stochastic gradient descent**:

- For each example $(u, i) \in \mathcal{T}$, update $w_{ij}\ \forall j \in N(i;u)$

$$\begin{aligned} w_{ij}^{t+1} &= w_{ij}^{t} - \gamma \frac{\partial}{\partial w_{ij}} E_{ui} \\ &= w_{ij}^{t} - \gamma (|N(i;u)|^{-\frac{1}{2}} (\hat{r}_{ui} - r_{ui})(r_{uj} - b_{uj}) + \lambda w_{ij}^{t}) \end{aligned}$$

KNN: Globally Optimized Weights

Benefits

- Weights optimized for the task of rating prediction
  - Not just borrowed from the neighborhood selection phase
- Weights not constrained to sum to 1
  - Important if all nearest neighbors are dissimilar
- Weights derived simultaneously
  - Accounts for correlations among neighbors
- Outperforms KNN with similarity or equal weights

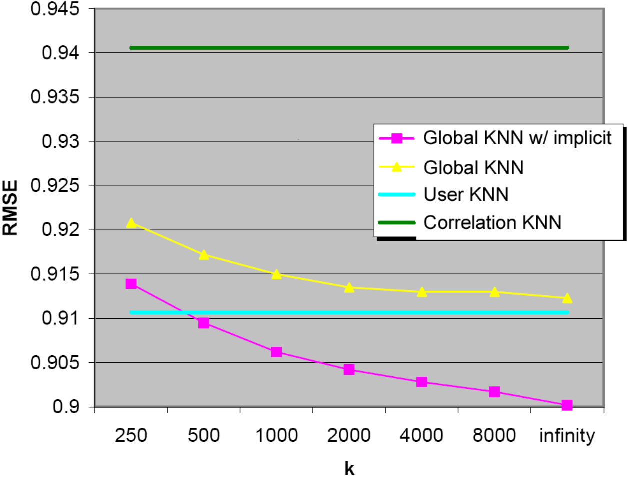
Drawbacks

- Must solve global optimization problem at training time
- Must store $O(M^2)$ weights in memory

Koren, 2008

KNN: Summary

Comparison of KNN weighting schemes on Netflix quiz data



KNN: Summary

Pros

- Intuitive interpretation
- When weights not learned...
  - Easy to implement
  - Zero training time
- Learning prediction weights can greatly improve accuracy for little overhead in space and time

Cons

- When weights not learned...
  - Need to store all item (or user) vectors in memory
  - May redundantly recompute similarity scores at test time
  - Similarity/equal weights not always suitable for prediction
- When weights learned...
  - Need to store $O(M^2)$ or $O(U^2)$ parameters
  - Must update stored parameters when new ratings occur

Low Dimensional Matrix Factorization

Matrix Completion

- Filling in the unknown ratings in a sparse $U \times M$ matrix R

$$\mathbf{R} = \begin{bmatrix} ? & ? & 1 & \dots & 4 \\ 3 & ? & ? & \dots & ? \\ ? & 5 & ? & \dots & 5 \end{bmatrix}$$

Low dimensional matrix factorization

- Model R as a product of two lower dimensional matrices

- A is $U \times K$ “user factor” matrix, $K \ll U, M$
- B is $M \times K$, “item factor” matrix
- Learning A and B allows us to reconstruct all of R

Low Dimensional Matrix Factorization

Interpretation: Rows of A and B are low dimensional feature vectors a_u and b_i for each user u and item i

Motivation: Dimensionality reduction

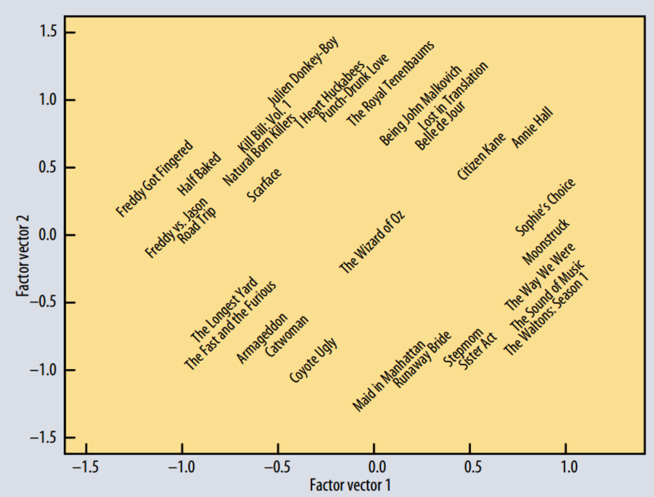
- Compact representation: only need to learn and store $UK + MK$ parameters
- Matrices can often be adequately represented by low rank factorizations

Low Dimensional Matrix Factorization

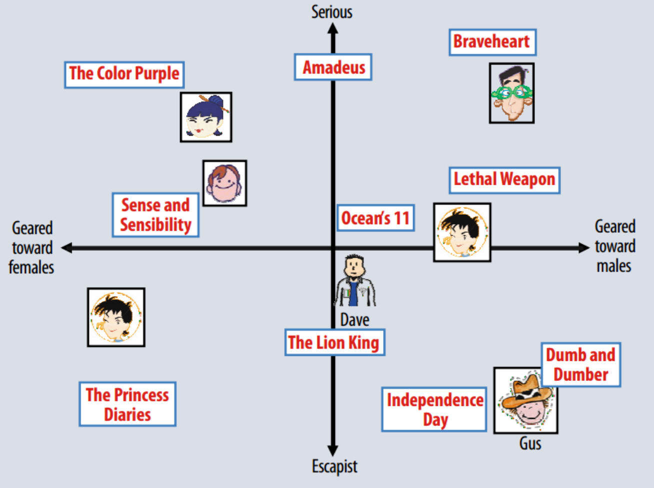
Very general framework that encapsulates many ML methods

- Singular value decomposition
- Clustering
  - A can represent cluster centers
  - B probabilities of belonging to each cluster
- Factor Analysis/Probabilistic PCA

IProjecting movies



Projecting users and movies



Singular Value Decomposition

Squared error objective for MF

$$\underset{A,B}{\operatorname{argmin}} \|R - AB^T\|_2^2 = \underset{A,B}{\operatorname{argmin}} \sum_{u=1}^{U} \sum_{i=1}^{M} (r_{ui} - \langle a_u, b_i \rangle)^2$$

- Reasonable objective since RMSE is our error metric

When all of R is observed, this problem is solved by singular value decomposition (SVD)

- **SVD:** $R = H \Sigma V^T$
  - H is $U \times U$ with $H^T H = I_{U \times U}$
  - V is $M \times M$ with $V^T V = I_{M \times M}$
  - Σ is $U \times M$ and diagonal
- **Solution:** Take first K pairs of singular vectors
  - Let $A = H_{U \times K} \Sigma_{K \times K}$ and $B = V_{M \times K}$

SVD with Missing Values

Weighted SE objective

$$\underset{A,B}{\operatorname{argmin}} \sum_{u=1}^{U} \sum_{i=1}^{M} W_{ui}(r_{ui} - \langle a_u, b_i \rangle)^2$$

Binary weights

- $W_{ui} = 1$ if r_{ui} observed, $W_{ui} = 0$ otherwise
- Only penalize errors on known ratings

How to optimize?

- Straightforward singular value decomposition no longer applies
- Local minima exist $\Rightarrow$ algorithm initialization is important

SVD with Missing Values

Insight: Chicken and egg problem

- If we knew the missing values in R, could apply SVD
- If we could apply SVD, we could find the missing values in R
- Idea: Fill in unknown entries with best guess; apply SVD; repeat

Expectation-Maximization (EM) algorithm

- Alternate until convergence:
  1. E step: $X = W * R + (1 - W) * \hat{R}$
     (* represents entrywise product)
  2. M step: $[H, \Sigma, V] = SVD(X)$, $\hat{R} = H_{U \times K} \Sigma_{K \times K} V^T_{M \times K}$

Complexity: $O(UM)$ space and $O(UMK)$ time per EM iteration

- What if UM or UMK is very large?
  - $UM = 8.5$ billion for Netflix Prize dataset
- Complete ratings matrix may not even fit into memory!

SVD with Missing Values

Regularized weighted SE objective

$$\underset{A,B}{\operatorname{argmin}} \sum_{u=1}^{U} \sum_{i=1}^{M} W_{ui}(r_{ui} - \langle a_u, b_i \rangle)^2 + \lambda\left(\sum_{u=1}^{U} \|a_u\|^2 + \sum_{i=1}^{M} \|b_i\|^2\right)$$

Equivalent form

$$\underset{A,B}{\operatorname{argmin}} \sum_{(u,i)\in\mathcal{T}} (r_{ui} - \langle a_u, b_i \rangle)^2 + \lambda\left(\sum_{u=1}^{U} \|a_u\|^2 + \sum_{i=1}^{M} \|b_i\|^2\right)$$

Motivation

- Counters *overfitting* by implicitly restricting optimization space
  - Shrinks entries of A and B toward 0
- Can improve *generalization error*, performance on unseen test data

SVD with Missing Values

Insight: If we knew B, could solve for each row of A via ridge regression and vice-versa

- Alternate between optimizing A and optimizing B with the other matrix held fixed

Alternating least squares (ALS) algorithm

- Alternate until convergence:
  1. For each user u, update
     $a_u \leftarrow (\sum_{i \in R(u)} b_i b_i^T + \lambda I)^{-1} \sum_{i \in R(u)} r_{ui} b_i$
  2. For each item i, update
     $b_i \leftarrow (\sum_{u \in R(i)} a_u a_u^T + \lambda I)^{-1} \sum_{u \in R(i)} r_{ui} a_u$

Complexity: $O(UK + MK)$ space, $O(UK^3 + MK^3)$ time per iteration

- Note: updates for vectors a_u can all be performed in parallel (same for b_i)
- No need to store completed ratings matrix

Zhou et al., 2008

SVD with Missing Values

Insight: Use standard gradient descent

- $\nabla_{a_u} E = \lambda a_u + \sum_{i \in R(u)} b_i(\langle a_u, b_i \rangle - r_{ui})$
- $\nabla_{b_i} E = \lambda b_i + \sum_{u \in R(i)} a_u(\langle a_u, b_i \rangle - r_{ui})$

Gradient descent algorithm

- Repeat until convergence:
  1. For each user u, update
     $a_u \leftarrow a_u - \gamma(\lambda a_u + \sum_{i \in R(u)} b_i(\langle a_u, b_i \rangle - r_{ui}))$
  2. For each item i, update
     $b_i \leftarrow b_i - \gamma(\lambda b_i + \sum_{u \in R(i)} a_u(\langle a_u, b_i \rangle - r_{ui}))$
- Can update all a_u in parallel (same for b_i)

Complexity: $O(UK + MK)$ space, $O(NK)$ time per iteration

- No need to store completed ratings matrix
- No K^3 overhead from solving linear regressions

SVD with Missing Values

Insight: Update parameter after each observed rating

- $\nabla_{a_u} E_{ui} = \lambda a_u + b_i(\langle a_u, b_i \rangle - r_{ui})$
- $\nabla_{b_i} E_{ui} = \lambda b_i + a_u(\langle a_u, b_i \rangle - r_{ui})$

Stochastic gradient descent algorithm

- Repeat until convergence:
  1. For each $(u, i) \in \mathcal{T}$
     1. Calculate error: $e_{ui} \leftarrow (\langle a_u, b_i \rangle - r_{ui})$
     2. Update $a_u \leftarrow a_u - \gamma(\lambda a_u + b_i e_{ui})$
     3. Update $b_i \leftarrow b_i - \gamma(\lambda b_i + a_u e_{ui})$

Complexity: $O(UK + MK)$ space, $O(NK)$ time per pass through training set

- No need to store completed ratings matrix
- No K^3 overhead from solving linear regressions

Takacs et al., 2008, Funk, 2006

Constrained MF as Clustering

Insight: Soft clustering of items is MF

- Row b_i represents item i's fractional belonging to each cluster
- Columns of A are cluster centers
- Yields greater interpretability

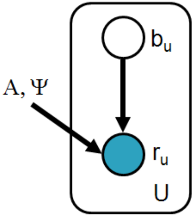
Constrained weighted SE objective

$$\underset{A,B}{\operatorname{argmin}} \sum_{u=1}^{U} \sum_{i=1}^{M} W_{ui}(r_{ui} - \langle a_u, b_i \rangle)^2 \text{ s.t. } \forall i \;\; b_i \geq 0, \sum_{k=1}^{K} b_{ik} = 1$$

- Wu and Li (2008) penalize constraints in the objective and optimize via stochastic gradient descent

Takeaway: Can add your favorite constraints and optimize with standard techniques

Factor Analysis



Motivation

- Explain data variability in terms of latent *factors*
- Provide model for how data is generated

The Model

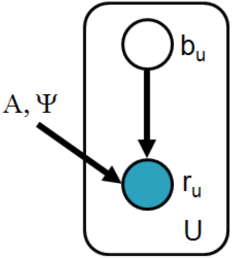
- For each user, r_u = partially observed ratings vector in $\mathbb{R}^M$
- For each user, b_u = latent factor vector in $\mathbb{R}^K$
- A is an $M \times K$ matrix of parameters (*factor loading matrix*)
- Ψ is an $M \times M$ covariance matrix
  - Probabilistic PCA: Special case when $\Psi = \sigma^2 I$
- To generate ratings for user u:
  1. Draw $b_u \sim \mathcal{N}(0, I_{K \times K})$
  2. Draw $r_u \sim \mathcal{N}(Ab_u, \Psi)$

Canny, 2002

Factor Analysis

Parameter Learning

- Only need to learn A and Ψ
- b_u are variables to be integrated out
- Typically use EM algorithm (Canny, 2002)
  - Can be very slow for large datasets
- Alternative: Stochastic gradient descent on negative log likelihood (Lawrence and Urtasun, 2009)



Low Dimensional MF: Summary

Pros

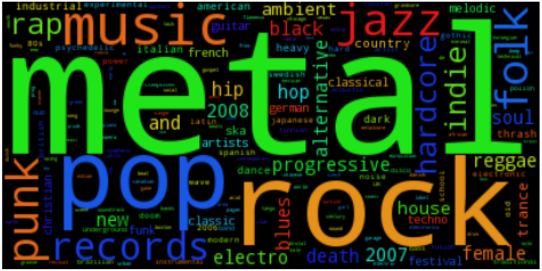
- Data reduction: only need to store $UK + MK$ parameters at test time
  - $MK + M^2$ needed for Factor Analysis
- Gradient descent and ALS procedures are easy to implement and scale well to large datasets
- Empirically yields high accuracy in CF tasks
- Matrix factors could be used as inputs into other learning algorithms (e.g. classifiers)

Cons

- Missing data MF objectives plagued by many local minima
- Initialization is important
- EM approaches tend to be slow for large datasets

This Spark afternoon

- create a wordcount



- recommend music

User: 0 Listens to:
elizabeth mitchell, the frames, ween, the decemberists, the delgados, tlc, clem snide, johnny cash, real mccoy, the beatles, camera obscura, belle and sebastian

Recommended: radiohead, coldplay, sigur ros, the beatles, sufjan stevens, the white stripes, bob dylan, animal collective, the smashing pumpkins, kings of leon

References

- K. Ali and W. van Stam, “TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture,” Proc. 10th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining, pp. 394-401, 2004.
- J. Basilico, T. Hofmann. 2004. Unifying collaborative and content-based filtering. In Proceedings of the ICML.
- R. Bell and Y. Koren, “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights,” IEEE International Conference on Data Mining (ICDM07), pp. 43-52, 2007.
- J. Bennet and S. Lanning, “The Netflix Prize,” KDD Cup and Workshop, 2007. www.netflixprize.com.
- L. Breiman, (1996). Stacked Regressions. Machine Learning, Vol. 24, pp. 49-64.
- J. Canny, “Collaborative Filtering with Privacy via Factor Analysis,” Proc. 25th ACM SIGIR Conf.on Research and Development in Information Retrieval (SIGIR02), pp. 238-245, 2002.
- A. Das, M. Datar, A. Garg and S. Rajaram, “Google News Personalization: Scalable Online Collaborative Filtering,” WWW07, pp.

References

- S. Funk, “Netflix Update: Try This At Home,” http://sifter.org/simon/journal/20061211.html, 2006.
- J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, “An Algorithmic Framework for Performing Collaborative Filtering,” in Proceedings of the Conference on Research and Development in Information Retrieval, 1999.
- Y. Koren. Collaborative filtering with temporal dynamics KDD, pp. 447-456, ACM, 2009.
- Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD08), pp. 426-434, 2008.
- N. Lawrence and R. Urtasun. Non-linear matrix factorization with Gaussian processes. ICML, ACM International Conference Proceeding Series, Vol. 382, p. 76, ACM, 2009.
- G. Linden, B. Smith and J. York, “Amazon.com Recommendations: Item-to-item Collaborative Filtering,” IEEE Internet Computing 7 (2003), 76-80.

References

- B. Marlin, R. Zemel, S. Roweis, and M. Slaney, “Collaborative filtering and the Missing at Random Assumption,” Proc. 23rd Conference on Uncertainty in Artificial Intelligence, 2007.
- A. Paterek, “Improving Regularized Singular Value Decomposition for Collaborative Filtering,” Proc. KDD Cup and Workshop, 2007.
- M. Piotte and M. Chabbert, “Extending the toolbox,” Netflix Grand Prize technical presentation, http://pragmatictheory.blogspot.com/, 2009.
- R. Salakhutdinov, A. Mnih and G. Hinton. Restricted Boltzmann Machines for collaborative filtering. Proc. 24th Annual International Conference on Machine Learning, pp. 791-798, 2007.
- N. Srebro and T. Jaakkola. Weighted low-rank approximations. In 20th International Conference on Machine Learning, pages 720-727. AAAI Press, 2003.
- Gabor Takacs, Istvan Pilaszy, Bottyan Nemeth, and Domonkos Tikk. Scalable collaborative ltering approaches for large recommender systems. Journal of Machine Learning Research, 10:623-656, 2009.
- C. Thompson. If you liked this, youre sure to love that. The New York Times, Nov 21, 2008.

References

- J. Wu and T. Li. A Modified Fuzzy C-Means Algorithm For Collaborative Filtering. Proc. Netflix-KDD Workshop, 2008.
- K. Yu, J. Lafferty, S. Zhu, and Y. Gong. Large-scale collaborative prediction using a nonparametric random effects model. In The 25th International Conference on Machine Learning (ICML), 2009.
- Y. Zhou, D. Wilkinson, R. Schreiber, R. Pan. “Large-Scale Parallel Collaborative Filtering for the Netix Prize,” AAIM 2008: 337-348.
- Y. Koren, R. Bell, C. Volinsky "Matrix Factorization techniques for recommender systems" IEEE/ACM Computer Journal 30-37, 2009