Binary Principal Component Analysis in the Netflix Collaborative Filtering Task

László Kozma, Alexander Ilin, Tapani Raiko
first.last@tkk.fi

Helsinki University of Technology
Adaptive Informatics Research Center

Saarbrücken, 19th October 2009

- **Recommender systems**
  - Predict user preference in order to offer more relevant items (Amazon.com, Netflix, iTunes Genius, Pandora)
  - *Content-based approach*: use similarity between items (e.g. book text, genre, title, author, actors in a movie, etc.)
  - *Collaborative filtering*: predict preferences from previous user-item relationships

Collaborative filtering: Netflix prize

- Netflix data set
  - Training set: 100,480,507 ratings from n = 480,189 users on m = 17,770 movies
  - Probe (test) set: 1,408,395 ratings are given for validation
  - Quiz set: 2,817,131 user/movie pairs with ratings withheld
  - Train, Probe, Quiz sets contain ratings from the same users and movies
  - Ratings from 1 to 5, time of voting provided

Collaborative filtering: Netflix prize

- Netflix prize competition
  - Score: RMSE on quiz set ratings, goal: 10% better RMSE than Cinematch (Netflix own algorithm trained on same data: 0.9514)
  - Leading solution: A blend of many methods (>100): k-NN, factorization (SVD), restricted Boltzmann machines and many other
  - `http://www.netflixprize.com/`
  - *Robert M. Bell, Yehuda Koren and Chris Volinsky: The BellKor solution to the Netflix Prize*

Collaborative filtering

..can be formulated as missing value reconstruction:

$$\mathbf{X}_{n\times m} = \begin{bmatrix} & 5 & & & \\ 2 & & & 2 & \\ & & ? & & 5 \\ & 3 & & & \\ & 1 & & ? & \\ ? & & & & 3 \\ & & & & \\ & 5 & & & ? \end{bmatrix}$$

n = 480,189 users, m = 17,770 movies, 100,480,507 ratings
(over 98% missing)

Singular Value Decomposition

- Rank-c approximation of ratings-matrix $\mathbf{X}$:

$$\mathbf{X}_{n\times m} \approx \mathbf{A}_{n\times c}\mathbf{S}_{c\times m}, \qquad \min_{\mathbf{A},\mathbf{S}} \|\mathbf{X} - \mathbf{A}\mathbf{S}\|_F^2$$

- Since $\mathbf{X}$ is incomplete, the cost function is:

$$\sum_{(i,j)\in O} (x_{ij} - \mathbf{a}_i^\top \mathbf{s}_j)^2 + \lambda(\|\mathbf{A}\|_F^2 + \|\mathbf{S}\|_F^2)$$

- The last term is added to avoid overfitting
- O is the set of observed ratings
- Unknown ratings are predicted as $x_{ij} = \mathbf{a}_i^\top \mathbf{s}_j, (i,j) \notin O$
- Code:
  `http://www.cis.hut.fi/alexilin/software`

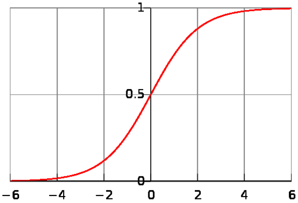
Logistic PCA

- Similar to SVD, but we use non-linearity:

$$P(y_{ij} = 1) = \sigma(\mathbf{a}_i^\top \mathbf{s}_j) \quad (1)$$

- Sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2)$$



- To avoid having a bias term, we fix the last row of $\mathbf{S}$ to all ones. Then, the elements in the last column of $\mathbf{A}$ play the role of the bias term.

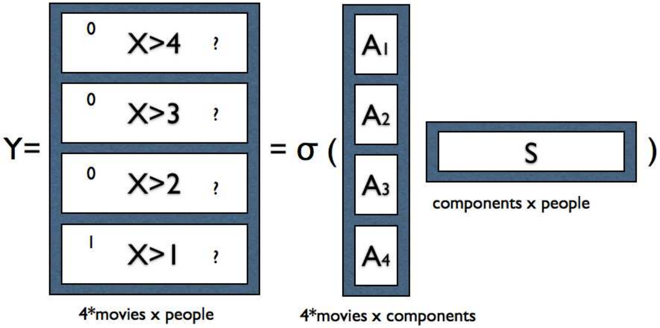
Binarizing Data

- Each rating value (1-5) encoded on 4 bits:

$$\begin{aligned} 1 &\rightarrow 0000 \\ 2 &\rightarrow 0001 \\ 3 &\rightarrow 0011 \\ 4 &\rightarrow 0111 \\ 5 &\rightarrow 1111 \end{aligned}$$

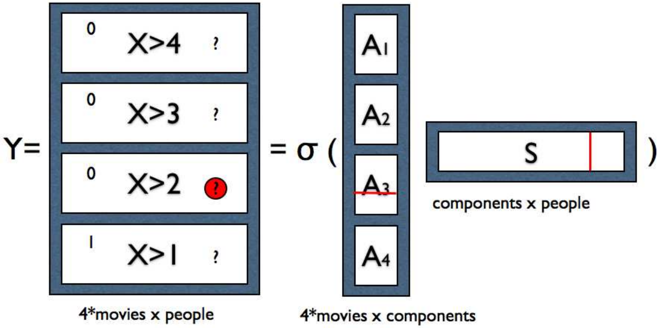
- We create 4 binary matrices from the original matrix of ratings
- Each element tells whether a rating is greater or smaller than a threshold
- Binarization scheme could be used also for continuous data

Method



Y=
0
X>4
?
0
X>3
?
0
X>2
?
1
X>1
?
4*movies x people
= σ (
A1
A2
A3
A4
4*movies x components
S
)
components x people

Method



Y=
0
X>4
?
0
X>3
?
0
X>2
?
1
X>1
?
4*movies x people
= σ (
A1
A2
A3
A4
4*movies x components
S
)
components x people

Logistic PCA

$$P(y_{ij} = 1) = \sigma(\mathbf{a}_i^\top \mathbf{s}_j) \tag{3}$$

- Both $\mathbf{A}$ and $\mathbf{S}$ are unknown and have to be estimated to fit the available ratings.
- We express the likelihood as a product of Bernoulli distributions based on $\mathbf{Y}$

$$L = \prod_{ij \in O} \sigma(\mathbf{a}_i^T \mathbf{s}_j)^{y_{ij}} (1 - \sigma(\mathbf{a}_i^\top \mathbf{s}_j))^{(1 - y_{ij})} \tag{4}$$

- O are the observed ratings in $\mathbf{Y}$

Regularization

- Maximum likelihood estimate prone to overfitting
- We use Gaussian priors for $\mathbf{A}$ and $\mathbf{S}$

$$P(s_{kj}) = \mathcal{N}(0, 1)$$

$$P(a_{ik}) = \mathcal{N}(0, v_k)$$

$$i = 1, \ldots, m\,; \quad k = 1, \ldots, c\,; \quad j = 1, \ldots, n$$

Learning

We use MAP-estimation for the model parameters:

$$
\begin{aligned}
F = & \sum_{ij \in O} y_{ij} \log \sigma(\mathbf{a}_i^{\mathsf{T}} \mathbf{s}_j) \\
& + \sum_{ij \in O} (1 - y_{ij}) \log(1 - \sigma(\mathbf{a}_i^{\mathsf{T}} \mathbf{s}_j)) \\
& - \sum_{i=1}^{m} \sum_{k=1}^{c} \left[\frac{1}{2} \log 2\pi v_k + \frac{1}{2 v_k} a_{ik}^2 \right] \\
& - \sum_{k=1}^{c} \sum_{j=1}^{n} \left[\frac{1}{2} \log 2\pi + \frac{1}{2} s_{kj}^2 \right]
\end{aligned}
\tag{5}
$$

Learning

The derivatives of the log-posterior:

$$\frac{\partial F}{\partial \mathbf{a}_i} = \sum_{j|ij \in O} \mathbf{s}_j^T (y_{ij} - \sigma(\mathbf{a}_i^\mathsf{T} \mathbf{s}_j)) - \mathrm{diag}(1/v_k)\mathbf{a}_i \tag{6}$$

$$\frac{\partial F}{\partial \mathbf{s}_j} = \sum_{i|ij \in O} \mathbf{a}_i^T (y_{ij} - \sigma(\mathbf{a}_i^\mathsf{T} \mathbf{s}_j)) - \mathbf{s}_j \,, \tag{7}$$

Gradient-ascent simultaneous update rules:

$$\mathbf{a}_i \leftarrow \mathbf{a}_i + \alpha \frac{\partial F}{\partial \mathbf{a}_i} \tag{8}$$

$$\mathbf{s}_k \leftarrow \mathbf{s}_k + \alpha \sqrt{\frac{m}{n}} \frac{\partial F}{\partial \mathbf{s}_j} \tag{9}$$

Learning

Gradient-ascent update rules:

$$\mathbf{a}_i \leftarrow \mathbf{a}_i + \alpha \frac{\partial F}{\partial \mathbf{a}_i} \tag{10}$$

$$\mathbf{s}_k \leftarrow \mathbf{s}_k + \alpha \sqrt{\frac{m}{n}} \frac{\partial F}{\partial \mathbf{s}_j} \tag{11}$$

- Scaling
- Update of the learning rate α: decrease by half if step unsuccessful, increase by 20% if successful
- Variance-parameters point-estimated to maximize F:

$$v_k = \frac{1}{d} \sum_{i=1}^{d} a_{ik}^2 \tag{12}$$

- **Prediction**
  - We split $\sigma(\mathbf{a}_i^T \mathbf{s}_j)$ into $X1$, $X2$, $X3$, $X4$. The probability of each rating value is then:

$$P_{x=1} = (1 - x_1)(1 - x_2)(1 - x_3)(1 - x_4) \tag{13}$$
$$P_{x=2} = (1 - x_1)(1 - x_2)(1 - x_3)x_4 \tag{14}$$
$$P_{x=3} = (1 - x_1)(1 - x_2)x_3 x_4 \tag{15}$$
$$P_{x=4} = (1 - x_1)x_2 x_3 x_4 \tag{16}$$
$$P_{x=5} = x_1 x_2 x_3 x_4 \tag{17}$$

  - We estimate the rating as expectation:

$$\hat{x} = \frac{1P_{x=1} + 2P_{x=2} + 3P_{x=3} + 4P_{x=4} + 5P_{x=5}}{P_{x=1} + P_{x=2} + P_{x=3} + P_{x=4} + P_{x=5}}. \tag{18}$$

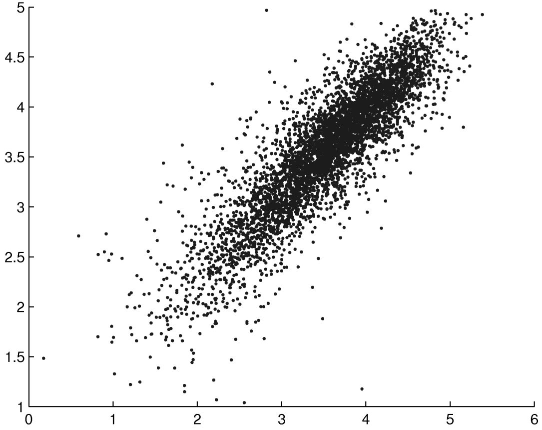
Experiments: MovieLens

# components c	Train rms	Test rms
binary PCA, movies$\times$users		
10	0.8956	0.9248
binary PCA, users$\times$movies		
10	0.8449	**0.9028**
20	0.8413	0.9053
30	0.8577	0.9146
VBPCA, users$\times$movies		
10	0.7674	0.8905
20	0.7706	0.8892
30	0.7696	**0.8880**

Table: Performance obtained for MovieLens 100K data

Experiments: MovieLens

Predictions on the test set with PCA (x-axis) and logistic PCA (y-axis):

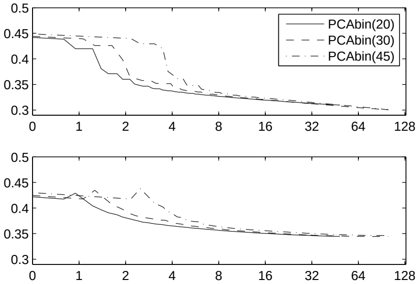


Experiments: Netflix

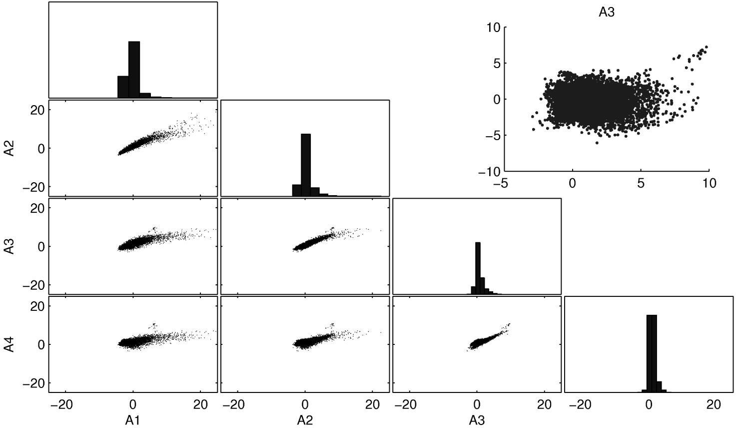
Method	Probe rms	Quiz subset
VBPCA, 50 components	0.9055	0.9070
BinPCA, 20 components	0.9381	0.9392
Blend	0.9046	0.9062

Table: Performance obtained for Netflix data

Training and test error:



Experiments: Netflix

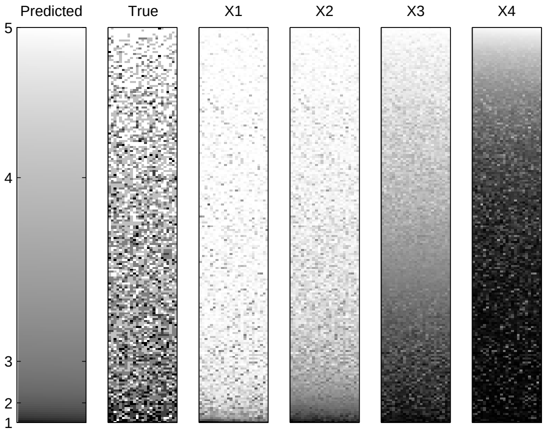


Conclusions

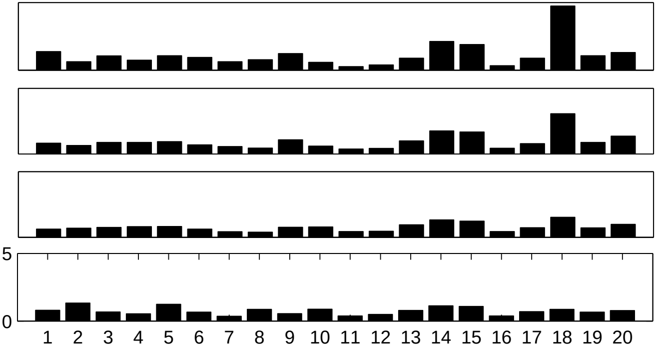
- We introduced an algorithm for binary logistic PCA that scales well to very high dimensional and very sparse data
- We experimented on a large scale collaborative filtering task
- The method captures different aspects of the data than traditional PCA
- We list some possible improvements in the paper

- Questions?

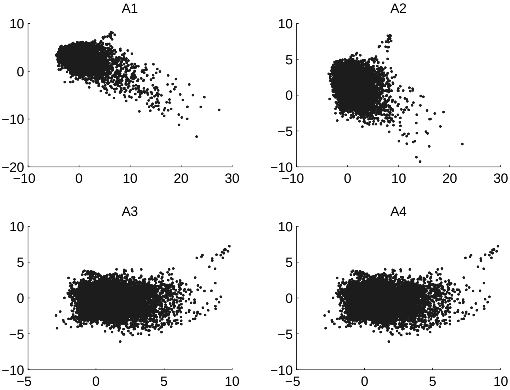
Experiments: Netflix



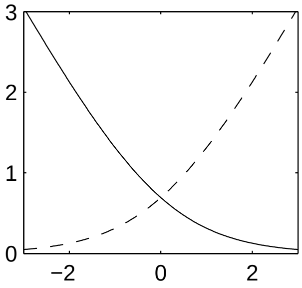
Experiments: Netflix

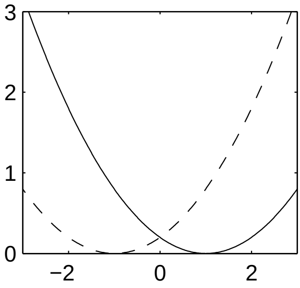


Experiments: Netflix

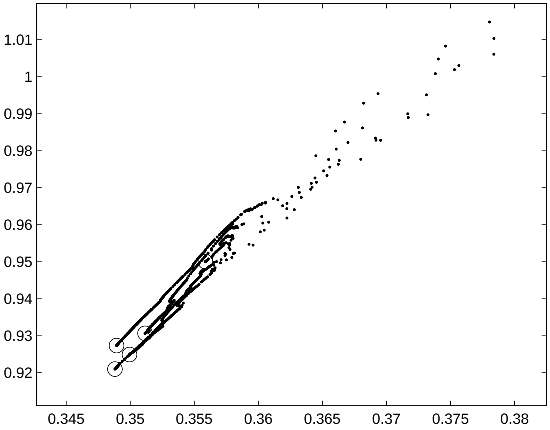


Experiments: Netflix





Experiments: Netflix



Blending

Linear regression using least squares error on known ratings from probe set

$$\begin{bmatrix} \text{predictions of method 1} \\ \text{predictions of method 2} \end{bmatrix}^\top \times \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} \text{probe set ratings} \end{bmatrix}^\top$$

Collaborative filtering

- A real recommender system
    - Low RMSE
    - Accuracy on top few picks
    - Easy to interpret
    - Speed of response
    - Update state (learn online)
    - Discover novel items

Implementation

Implementation in Matlab

- Extension of SVD code from `http://www.cis.hut.fi/alexilin/software`
- Most critical parts (`sigmoid()`, $\mathbf{A} \times \mathbf{S}$) implemented in C++, linked with mex library
- Use only sparse matrices, make sure they are never internally transformed into full matrices
  Example: `Y = spfun(@(x) 1./x, Y);`
- Three different values: $missing \rightarrow 0$, $0 \rightarrow \epsilon$, $1 \rightarrow 1$
- Minibatch: no need to keep whole $\mathbf{Y}$ in memory at all times, either load from disk in every iteration or recompute

Implementation

- We need to compute
  - $L_1 = log(\sigma(x))$
  - $L_2 = log(1 - \sigma(x))$
  - where $\sigma(x) = \frac{1}{1+e^{-x}}$
- For numerical stability we compute
  - $if(x >= 0)$
    - $L_1 = -log(1 + e^{-x})$
    - $L_2 = -x + L_1$
  - $if(x < 0)$
    - $L_2 = -log(1 + e^{x})$
    - $L_1 = x + L_2$

Collaborative filtering

- Other ideas
  - Trouble with data: spam accounts, more people with same account
  - Use information about quiz data set: data is not missing (entirely) at random
  - Use external data: identify movies/users
  - Rating depending on time: influenced by previously seen movies
  - If we both hate *Titanic* which everyone liked, that tells more about our similar tastes than if we both liked it
  - etc.