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
Introduction

- **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:

\[
X_{n \times m} = \begin{bmatrix}
5 & 2 \\
2 & 2 & ? \\
3 & ? & 5 \\
1 & ? \\
? & 3 \\
5 & ?
\end{bmatrix}
\]

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

• Rank-$c$ approximation of ratings-matrix $X$:

$$X_{n \times m} \approx A_{n \times c} S_{c \times m}, \quad \min_{A,S} \|X - AS\|_F^2$$

• Since $X$ is incomplete, the cost function is:

$$\sum_{(i,j) \in O} (x_{ij} - a_i^T s_j)^2 + \lambda (\|A\|_F^2 + \|S\|_F^2)$$

• The last term is added to avoid overfitting
• $O$ is the set of observed ratings
• Unknown ratings are predicted as $x_{ij} = a_i^T s_j, (i,j) \notin O$
• Code:
  
  http://www.cis.hut.fi/alexilin/software
Logistic PCA

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

\[ P(y_{ij} = 1) = \sigma(a_i^T s_j) \]  \hspace{1cm} (1)

- Sigmoid function:

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]  \hspace{1cm} (2)

- To avoid having a bias term, we fix the last row of \( S \) to all ones. Then, the elements in the last column of \( A \) play the role of the bias term.
Method

Binarizing Data

- Each rating value (1-5) encoded on 4 bits:
  
  1 → 0000  
  2 → 0001  
  3 → 0011  
  4 → 0111  
  5 → 1111  

- 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

\[
Y = \sigma(\text{components x people})
\]

4*movies x people

4*movies x components
Method

Logistic PCA

\[ P(y_{ij} = 1) = \sigma(a_i^T s_j) \quad (3) \]

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

\[ L = \prod_{ij \in O} \sigma(a_i^T s_j)^{y_{ij}} (1 - \sigma(a_i^T s_j))^{(1-y_{ij})} \quad (4) \]

- \( O \) are the observed ratings in \( Y \)
Method

Regularization

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

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

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

\[i = 1, \ldots, m; \quad k = 1, \ldots, c; \quad j = 1, \ldots, n\]
Learning

We use MAP-estimation for the model parameters:

\[
F = \sum_{ij \in O} y_{ij} \log \sigma(a_i^T s_j) \\
+ \sum_{ij \in O} (1 - y_{ij}) \log(1 - \sigma(a_i^T s_j)) \\
- \sum_{i=1}^{m} \sum_{k=1}^{c} \left[ \frac{1}{2} \log 2\pi v_k + \frac{1}{2v_k} a_{ik}^2 \right] \\
- \sum_{k=1}^{c} \sum_{j=1}^{n} \left[ \frac{1}{2} \log 2\pi + \frac{1}{2} s_{kj}^2 \right]
\]

(5)
Method

Learning
The derivatives of the log-posterior:

\[
\frac{\partial F}{\partial a_i} = \sum_{j|ij \in O} s_j^T (y_{ij} - \sigma(a_i^s_j)) - \text{diag}(1/v_k)a_i \quad (6)
\]

\[
\frac{\partial F}{\partial s_j} = \sum_{i|i,j \in O} a_i^T (y_{ij} - \sigma(a_i^s_j)) - s_j , \quad (7)
\]

Gradient-ascent simultaneous update rules:

\[
a_i \leftarrow a_i + \alpha \frac{\partial F}{\partial a_i} \quad (8)
\]

\[
s_k \leftarrow s_k + \alpha \sqrt{\frac{m}{n}} \frac{\partial F}{\partial s_j} \quad (9)
\]
Method

Learning
Gradient-ascent update rules:

\[
\mathbf{a}_i \leftarrow \mathbf{a}_i + \alpha \frac{\partial F}{\partial \mathbf{a}_i} \\
\mathbf{s}_k \leftarrow \mathbf{s}_k + \alpha \sqrt{\frac{m}{n}} \frac{\partial F}{\partial \mathbf{s}_j}
\]

(10) (11)

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

\[
v_k = \frac{1}{d} \sum_{i=1}^{d} a_{ik}^2
\]

(12)
Method

- **Prediction**
  - We split $\sigma(a_j^T s_j)$ into $X_1$, $X_2$, $X_3$, $X_4$. The probability of each rating value is then:

    $P_{x=1} = (1 - x_1)(1 - x_2)(1 - x_3)(1 - x_4)$  \hspace{1cm} (13)
    
    $P_{x=2} = (1 - x_1)(1 - x_2)(1 - x_3)x_4$  \hspace{1cm} (14)
    
    $P_{x=3} = (1 - x_1)(1 - x_2)x_3x_4$  \hspace{1cm} (15)
    
    $P_{x=4} = (1 - x_1)x_2x_3x_4$  \hspace{1cm} (16)
    
    $P_{x=5} = x_1x_2x_3x_4$  \hspace{1cm} (17)
    
  - We estimate the rating as expectation:

    $$\hat{x} = \frac{1P_{x=1} + 2P_{x=2} + 3P_{x=3} + 4P_{x=4} + 5P_{x=5}}{P_{x=1} + P_{x=2} + P_{x=3} + P_{x=4} + P_{x=5}}.$$  \hspace{1cm} (18)
## Experiments: MovieLens

**Table:** Performance obtained for MovieLens 100K data

<table>
<thead>
<tr>
<th># components $c$</th>
<th>Train rms</th>
<th>Test rms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>binary PCA, movies $\times$ users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.8956</td>
<td>0.9248</td>
</tr>
<tr>
<td><strong>binary PCA, users $\times$ movies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.8449</td>
<td><strong>0.9028</strong></td>
</tr>
<tr>
<td>20</td>
<td>0.8413</td>
<td>0.9053</td>
</tr>
<tr>
<td>30</td>
<td>0.8577</td>
<td>0.9146</td>
</tr>
<tr>
<td><strong>VBPCA, users $\times$ movies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.7674</td>
<td>0.8905</td>
</tr>
<tr>
<td>20</td>
<td>0.7706</td>
<td>0.8892</td>
</tr>
<tr>
<td>30</td>
<td>0.7696</td>
<td><strong>0.8880</strong></td>
</tr>
</tbody>
</table>
Experiments: MovieLens

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Probe rms</th>
<th>Quiz subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBPCA, 50 components</td>
<td>0.9055</td>
<td>0.9070</td>
</tr>
<tr>
<td>BinPCA, 20 components</td>
<td>0.9381</td>
<td>0.9392</td>
</tr>
<tr>
<td>Blend</td>
<td>0.9046</td>
<td>0.9062</td>
</tr>
</tbody>
</table>

**Table:** Performance obtained for Netflix data

### Training and test error:

![Graph showing training and test error for different methods](image-url)
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
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 in Matlab

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