Algorithms for Collaborative Filtering

or

“How to Get Half Way to Winning $1 million from Netflix”

Todd Lipcon
Advisor: Prof. Philip Klein
The Real-World Problem

- E-commerce sites would like to make personalized product recommendations to customers

- Often, there is no data about the customers, and/or the products are hard to describe explicitly (e.g. movies, music, art, books)
The Simplified Problem

- Regression/prediction problem
- Given a \{user, item\} pair we would like to predict the user’s ordinal rating of that item
- In the case of the Netflix Prize, we are scored based on squared error of our predictions
Collaborative Filtering

- We do have past preference data for repeat users
  - Purchase history (binary)
  - Ratings (ordinal)
- We can infer traits of users and items from this preference data
The Problem Take 2

(this time with equations)

- We form a Rating Matrix \( A \in \mathbb{R}^{m \times n} \) where \( A_{ij} \) is the rating by user \( i \) on item \( j \).
- In Netflix, 99% of the entries in \( A \) are missing.
- We would like to solve:

\[
X = \underset{X \in \mathbb{R}^{m \times n}}{\arg\min} \sum_{ij \in A} (A_{ij} - X_{ij})^2
\]

subject to some constraint or regularization penalty on \( X \) to ensure generalization to new examples \( ij \).
Item-KNN Approach

- We assume that items in the system can be described by their similarity to other items.
- We can then predict that a given user will assign similar ratings to similar items.
- We build a list of neighbors for each movie based on Pearson Correlation between their overlapping rating data.
- We adjust correlation by calculating a confidence interval.
Item-KNN Prediction

<table>
<thead>
<tr>
<th></th>
<th>RMSE</th>
<th>NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>0.9411</td>
<td>0.4274</td>
</tr>
<tr>
<td>MovieLens</td>
<td>0.9126</td>
<td>0.4208</td>
</tr>
</tbody>
</table>

- State of the art ("Ensemble MMMF") achieves NMAE of 0.4054 on MovieLens
- Best Netflix result is RMSE = 0.8831
- CineMatch RMSE = 0.9514
## Item-KNN Data Mining

**Movies Similar to *Finding Nemo (Fullscreen)***

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Movie Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.650</td>
<td><em>Finding Nemo (Widescreen)</em></td>
</tr>
<tr>
<td>0.316</td>
<td><em>Monsters, Inc.</em></td>
</tr>
<tr>
<td>0.281</td>
<td><em>A Bug’s Life</em></td>
</tr>
<tr>
<td>0.259</td>
<td><em>Toy Story</em></td>
</tr>
<tr>
<td>0.256</td>
<td><em>Toy Story 2</em></td>
</tr>
<tr>
<td>0.251</td>
<td><em>The Incredibles</em></td>
</tr>
<tr>
<td>0.243</td>
<td><em>Shrek 2</em></td>
</tr>
<tr>
<td>0.226</td>
<td><em>Ice Age</em></td>
</tr>
<tr>
<td>0.218</td>
<td><em>The Lion King: Special Edition</em></td>
</tr>
<tr>
<td>0.215</td>
<td><em>Harry Potter and the Prisoner of Azkaban</em></td>
</tr>
</tbody>
</table>

- Same movie!
- Pixar
- CG/Kids
- Animation
- Kids
Matrix Factorization

We replace objective

\[ X = \arg\min_{X \in \mathbb{R}^{m \times n}} \sum_{i,j \in A} (A_{ij} - X_{ij})^2 \]

with the factorized objective

\[ X = \arg\min_{U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{n \times r}} \sum_{i,j \in A} \left( A_{ij} - (UV^T)_{ij} \right) \]

which is easier to learn due to smaller number of free parameters \( m + n \) instead of \( m \cdot n \)
Learning $U$ and $V$

- The factorized objective isn’t convex

- We’d rather just learn the matrices one column at a time such that each additional column reduces the error as much as possible

- Learning the columns $u$ and $v$ is still non-convex, but works well in practice anyway

- Learning the columns incrementally is an example of a gradient boosting machine [Friedman]
Gradient Boosting
Matrix Factorization

- Pseudocode:

\[
\forall i,j \in X : X_{ij} = \text{BASELINE}(i, j)
\]

for \( i = 1 \) to \( M \)

\[
\forall i,j \in X : \tilde{X}_{ij} = -\frac{\partial}{\partial X_{ij}} (A_{ij} - X_{ij})^2
\]

\[
\{u, v\} = \text{UVLEARN}(\tilde{X})
\]

\[
U = [U \nu u]; V = [V \nu v]; X = X + \nu uu^T
\]

end for

any convex loss function can be substituted here!

shrinkage parameter (more later)
UVLearn Objective

• Aim to learn best rank-1 approximation of the input matrix $\tilde{X}$

• Given $X \in \mathbb{R}^{m \times n}$, solve:

$$\{u, v\} = \arg\min_{u \in \mathbb{R}^m, v \in \mathbb{R}^n} \sum_{i,j} (X_{ij} - u_i v_j)^2$$

• But this will overfit training data – needs regularization!
UVLearn Objective

(regularized)

• Given \( X \in \mathbb{R}^{m \times n} \) and regularization parameter \( \lambda \), solve:

\[
\{ u, v \} = \arg\min_{u \in \mathbb{R}^m, v \in \mathbb{R}^n} \left[ \sum_{i,j} (X_{i,j} - u_{i}v_{j})^2 + \lambda \sum_{i=1}^{m} |u_i - \bar{u}|^2 + \lambda \sum_{j=1}^{n} |v_j - \bar{v}|^2 \right]
\]
UVLearn Regularization

• Penalization is based around distance from "empirical prior" means $\bar{u}$ and $\bar{v}$

• We simultaneously optimize the parameters $u$ and $v$ along with their changing means
UVLearn Optimization

- Stochastic Gradient Descent – partial derivatives:

\[
\begin{align*}
\frac{\partial E_{ij}}{\partial u_i} &= 2v_j(u_i v_j - R_{ij}) + 2\lambda(u_i - \bar{u}) \\
\frac{\partial E_{ij}}{\partial v_j} &= 2u_i(u_i v_j - R_{ij}) + 2\lambda(v_j - \bar{v}) \\
\frac{\partial E_{ij}}{\partial \bar{u}} &= -2\lambda(u_i - \bar{u}) \\
\frac{\partial E_{ij}}{\partial \bar{v}} &= -2\lambda(v_j - \bar{v})
\end{align*}
\]
UVLearn Optimization

• Stochastic GD update equations:

\[ u'_i = u_i - \eta \left( 2v_j(u_iv_j - X_{ij}) + 2\lambda(u_i - \bar{u}) \right) \]

\[ v'_j = v_j - \eta \left( 2u_i(u_iv_j - X_{ij}) + 2\lambda(v_j - \bar{v}) \right) \]

\[ \bar{u}' = \bar{u} - \eta \left( -2\lambda(u_i - \bar{u}) \right) \]

\[ \bar{v}' = \bar{v} - \eta \left( -2\lambda(v_j - \bar{v}) \right) \]

\[ H \left( E(u_i, v_j, \bar{u}, \bar{v}) \right) = \begin{bmatrix}
2v^2_j + 2\lambda & 4u_i v_j - 2R_{ij} & -2\lambda & 0 \\
4u_i v_j - 2R_{ij} & 2u^2_i + 2\lambda & 0 & -2\lambda \\
-2\lambda & 0 & 2\lambda & 0 \\
0 & -2\lambda & 0 & 2\lambda
\end{bmatrix} \]
GBMF Regularization

- UVLearn regularization parameter $\lambda$
- UVLearn GD stopping threshold $\tau$
- Gradient boosting shrinkage parameter $\nu$
- Number of gradient boosting iterations $M$
Different shrinkage, same UVLearn regularization

Different shrinkage, best UVLearn regularization
<table>
<thead>
<tr>
<th>Dataset</th>
<th>RMSE</th>
<th>NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>0.9134</td>
<td>0.4099</td>
</tr>
<tr>
<td>MovieLens</td>
<td>0.8859</td>
<td>0.4051</td>
</tr>
</tbody>
</table>

- State of the art ("Ensemble MMMF") achieves NMAE of 0.4054 on MovieLens
- Best Netflix result is RMSE = 0.8831
- CineMatch RMSE = 0.9514
Bagged GBMF

- Bootstrap aggregation (*bagging*) - general variance-reduction framework that does not increase bias [Breiman]

- We construct ensembles of 60 GBMF models trained on bootstrap samples of the training set

- Ensemble predictions are the mean of member predictions
### Bagged GBMF Prediction

<table>
<thead>
<tr>
<th></th>
<th>RMSE</th>
<th>NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>MovieLens</td>
<td>0.8850</td>
<td>0.4045</td>
</tr>
</tbody>
</table>

Netflix jobs are still running and getting better

Weren’t able to do good grid search yet even for MovieLens because of resource limitations on SGE cluster

- State of the art (“Ensemble MMMF”) achieves NMAE of 0.4054 on MovieLens
- Best Netflix result is RMSE = 0.8831
- CineMatch RMSE = 0.9514
Combining Predictors

• A single data set has many different types of component data (varying sparsity, etc)

• A single algorithm (or choice of tuning parameters for an algorithm) may not give the best results for every type of component data

• We can do better by smartly blending between different predictors
Learning Combinations

• Train individual predictors on most of the data

• Train combination algorithm on remaining data using $k$-fold cross-validation for model selection

• Then retrain individual predictors on the full data and use the combination predictor learned above
Learning Combinations

- For each entry in the validation set, we create a vector with movie mean, user mean, movie count, user count, and the predictions by all component predictors.

- We then can use standard regression techniques to learn the combination function.
Linear Combination

• Use simple least squares regression:

<table>
<thead>
<tr>
<th></th>
<th>RMSE</th>
<th>NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>0.9092</td>
<td>0.4061</td>
</tr>
<tr>
<td>MovieLens</td>
<td>0.8817</td>
<td>0.4062</td>
</tr>
</tbody>
</table>

• Add 1st-order interaction terms:

<table>
<thead>
<tr>
<th></th>
<th>RMSE</th>
<th>NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>0.9062</td>
<td>0.4049</td>
</tr>
<tr>
<td>MovieLens</td>
<td>0.8758</td>
<td>0.4043</td>
</tr>
</tbody>
</table>

• Adding 2nd-order interaction terms overfits
Support Vector Machine Combination

• Use nu-SVR support vector regression to minimize squared loss
• Use support vector ordinal regression to minimize absolute loss
• Work still in progress - initial results for NMAE significantly improve on Ensemble MMMF (MovieLens NMAE ~ 0.4000)
Other Strong Netflix Approaches

• Restricted Boltzmann Machines (ML @ UToronto team) [Salakhutdinov, Mnih, and Hinton 2007]

• Probabilistic Latent Semantic Analysis ("PLSA" team) [Hofmann 2003]

• Gibbs Sampling of Latent Variables (Paul Harrison)

• Most of the top teams are doing combination of some kind
Future Work

- Better grid search for bagging experiments
- Stochastic Meta-Descent optimization
- Continued work on SVR combination
- “Conditional” rating prediction
- Alternative loss functions (GBMF optimizes any convex loss)
Questions?