Using SVD to Predict Movie Ratings

Serdar Sali

sali@soe.ucsc.edu
Collaborative filtering

- Similar tastes in the past: similar tastes in the future

Customers Who Bought This Item Also Bought

- HP Pavilion TX1420US 12.1" Laptop (AMD Turion 64 X2 Dual-Core) ★★★★★ (11) $1,169.99
- HP Pavilion DV775US 15.4" Entertainment Laptop (Intel Core 2) ★★★★★ (18) $1,099.99
- HP Pavilion DV2740SE 14.1" Entertainment Laptop (AMD Turion 64) ★★★★★ (12) $1,179.83
- HP Pavilion TX1320US 12.1" Entertainment Notebook PC (AMD Tur... ★★★★★ (35) $1,199.99
- HP Artist Edition Messenger Case $49.99
Our domain: Movie Ratings

• We would like to predict how a user would rate a given movie

• A multi-label classification problem
  • Ratings: 1 to 5
  • Only available data: User – movie – rating triplets
Problem Formulation

- $R : u \times m$ matrix of ratings
  - $u : \text{number of users}$
  - $m : \text{number of movies}$

- Ideally, we would have:
  - A set of features for each movie: $f_j$
  - A set of preference multipliers for each user: call $p_i$

- Then rating of user $i$ for movie $j$ becomes $r_{ij} = p_i f_j^T$
Problems

- Feature list for movies are hard to obtain
  - Task is inherently subjective and difficult, classification is hard
  - Dependence on external data resources
  - Tremendous effort required to clean-up data

- Notice that R already contains this data, but it is lumped together in a sum.
  - Can we retrieve it somehow?
Singular Value Decomposition

- SVD states that every $m \times n$ matrix $A$ can be written as $A = USV^T$ where
  - $U$ is an $m \times m$ orthogonal matrix,
  - $S$ is an $m \times n$ diagonal matrix with singular values of $A$ along the diagonal,
  - $V$ is an $m \times n$ orthogonal matrix.
Since $S$ is diagonal, we can obtain a more compact representation:

\[
\begin{bmatrix}
X & \mathbf{0} \\
\mathbf{0} & \mathbf{0}
\end{bmatrix}
\begin{bmatrix}
U & V
\end{bmatrix}
= 
\begin{bmatrix}
U & V
\end{bmatrix}
\begin{bmatrix}
X & \mathbf{0} \\
\mathbf{0} & \mathbf{0}
\end{bmatrix}
\begin{bmatrix}
\mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0}
\end{bmatrix}
SVD for Matrix Approximation

• Instead of using all the singular values of S, use only the most significant r

• Compute a rank-r approximation $A'$ to $A$ such that $A' = U'S'V'^T$ where $U'$ is mxr, $S'$ is rxr, and $V'$ is mxr

• This approximation minimizes the Frobenius form: $\|A-A'\|_F = \sqrt{\sum (a_{ij} - a'_{ij})^2}$
SVD for Movie Rating Prediction

- Given a matrix of ratings $R$, we want to compute an approximate matrix $R_{\text{app}}$ such that RMSE is minimized.

- But $\text{RMSE} = \|R - R_{\text{app}}\|_F$

- So, SVD is a perfect fit to our problem
SVD for Movie Rating Prediction

• Recall $R_{uxm}$: ratings matrix

• Compute an SVD for $R$ and just lump the singular value matrix in the sum:

  $$R = P_{uxf} F_{mxf}^T$$

  • $P$: Preference matrix for $f$ features for $u$ users
  • $F$: $f$-features matrix for $m$ movies
But...

- SVD is not defined for sparse matrices
  - Netflix data: 8.5B possible entries, 8.4B empty
- Fill in with averages, some clever combinations
  - Perturbs the data too much
  - And even if we fill in the missing values...
- Computing SVD for large matrices is computationally very expensive
Incremental SVD Method

- Devised by Simon Funk
- Only consider existing values
- Do a gradient descent to minimize the error:
  - \( E = (R - R_{\text{app}})_{ij}^2 \)
  - Take the derivative wrt \( p_{ij} \) and \( f_{jk} \), and the updates become
    - \( p_{ik}^{(t+1)} = p_{ik}^{(t)} + \text{learning\_rate} \times (R - R_{\text{app}})_{ij} f_{jk}^{(t)} \)
    - \( f_{jk}^{(t+1)} = f_{jk}^{(t)} + \text{learning\_rate} \times (R - R_{\text{app}})_{ij} p_{ik}^{(t)} \)
Implementation

• Online update
• Set each feature & each multiplier to 0.1
• Train the most significant feature first, and then the second, etc.

• Parameters:
  • Number of features
  • Learning rate
  • Regularization : very simple : -K*(update target)
  • Different starting values
Experiments

- Smaller dataset
  - 2000 movies, 480189 users
  - 10314269 ratings
  - Just blind downsampling
- Test: predict 3000 ratings
- Does not perform as well as it does on whole dataset
Experiments

- Number of Features

<table>
<thead>
<tr>
<th>#feats</th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.7623</td>
<td>0.9858</td>
</tr>
<tr>
<td>20</td>
<td>0.7128</td>
<td>0.9869</td>
</tr>
<tr>
<td>30</td>
<td>0.6820</td>
<td>0.9882</td>
</tr>
<tr>
<td>40</td>
<td>0.6619</td>
<td>1.0028</td>
</tr>
<tr>
<td>50</td>
<td>0.6468</td>
<td>1.0438</td>
</tr>
</tbody>
</table>

Combined: 0.9895
Experiments

- Regularization

![Effect of Regularization graph](image)

<table>
<thead>
<tr>
<th>Reg Rate</th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.6118</td>
<td>0.9973</td>
</tr>
<tr>
<td>0.005</td>
<td>0.6418</td>
<td>0.9899</td>
</tr>
<tr>
<td>0.010</td>
<td>0.6705</td>
<td>0.9823</td>
</tr>
<tr>
<td>0.015</td>
<td>0.7027</td>
<td>0.9787</td>
</tr>
<tr>
<td>0.020</td>
<td>0.7333</td>
<td>0.9686</td>
</tr>
</tbody>
</table>

Combined: 0.9741
Experiments

- Different learning rates

<table>
<thead>
<tr>
<th>Lrate</th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0005</td>
<td>0.8180</td>
<td>0.9755</td>
</tr>
<tr>
<td>0.0010</td>
<td>0.7666</td>
<td>0.9765</td>
</tr>
<tr>
<td>0.0015</td>
<td>0.7464</td>
<td>0.9843</td>
</tr>
<tr>
<td>0.0020</td>
<td>0.7365</td>
<td>0.9904</td>
</tr>
<tr>
<td>0.0025</td>
<td>0.7314</td>
<td>0.9955</td>
</tr>
</tbody>
</table>

Combined: 0.9756
Experiments

• Different starting values

<table>
<thead>
<tr>
<th></th>
<th>Base = 1</th>
<th>Base = Average</th>
<th>Base = Average + Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>0.7651</td>
<td>0.7638</td>
<td>0.7638</td>
</tr>
<tr>
<td>Test data</td>
<td>1.0144</td>
<td>1.0156</td>
<td>1.0158</td>
</tr>
</tbody>
</table>

• Combined : 0.9741
Conclusion

• Overfitting is a problem
  • Especially with my smaller dataset, models tend to overfit the training data very easily
• Even a blind combination of results give surprisingly good results
  • This implies that different models work good for different cases
  • A combination of different models is the way to go
Future Work

- Many parameters to adjust
- More clever downsampling of the dataset
- Use the computed features as input to another algorithm?
  - May help fine-tune the results at least
- Different regularization methods
Questions?