CS425: Algorithms for Web Scale Data

Lecture 9: Recommender Systems:
Latent Factor Models & Netflix Challenge

Most of the slides are from the Mining of Massive Datasets book.
These slides have been modified for CS425. The original slides can be accessed at: www.mmds.org
The Netflix Prize

- **Training data**
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005

- **Test data**
  - Last few ratings of each user (2.8 million)
  - **Evaluation criterion**: Root Mean Square Error (RMSE) = 
    \[
    \frac{1}{|R|} \sqrt{\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2}
    \]
  - Netflix’s system RMSE: 0.9514

- **Competition**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

---

The Netflix Utility Matrix $R$

Matrix $R$

480,000 users

17,700 movies

1  3  4
  3  5
  4  5
  3

2  2  2

2  1  1

3  3

1

Utility Matrix $R$: Evaluation

Matrix $R$

$$\text{RMSE} = \frac{1}{|R|} \sqrt{\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2}$$

The winner of the Netflix Challenge!

**Multi-scale modeling of the data:** Combine top level, “regional” modeling of the data, with a refined, local view:

- **Global:**
  - Overall deviations of users/movies

- **Factorization:**
  - Addressing “regional” effects

- **Collaborative filtering:**
  - Extract local patterns
Global:

- Mean movie rating: 3.7 stars
- *The Sixth Sense* is 0.5 stars above avg.
- Joe rates 0.2 stars below avg.

⇒ Baseline estimation:

Joe will rate *The Sixth Sense* 4 stars

Local neighborhood (CF/NN):

- Joe didn’t like related movie *Signs*

⇒ Final estimate:

Joe will rate *The Sixth Sense* 3.8 stars
Recap: Collaborative Filtering (CF)

- Earliest and most popular **collaborative filtering method**
- Derive unknown ratings from those of “**similar**” movies (item-item variant)
- Define **similarity measure** $s_{ij}$ of items $i$ and $j$
- Select $k$-nearest neighbors, compute the rating
  - $N(i; x)$: items **most similar to** $i$ **that were rated by** $x$

$$\hat{r}_{xi} = \frac{\sum_{j \in N(i; x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i; x)} s_{ij}}$$

$s_{ij}$... similarity of items $i$ and $j$
$r_{xj}$... rating of user $x$ on item $j$
$N(i; x)$... set of items similar to item $i$ that were rated by $x$
In practice we get better estimates if we model deviations:

\[ \hat{r}_{xi} = b_{xi} + \frac{\sum_{j \in N(i; x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i; x)} s_{ij}} \]

Baseline estimate for \( r_{xi} \):

\[ b_{xi} = \mu + b_x + b_i \]

- \( \mu \) = overall mean rating
- \( b_x \) = rating deviation of user \( x \)
  \( = (\text{avg. rating of user } x) - \mu \)
- \( b_i \) = (avg. rating of movie \( i \)) – \( \mu \)

Problems/Issues:
1) Similarity measures are “arbitrary”
2) Pairwise similarities neglect interdependencies among users
3) Taking a weighted average can be restricting

Solution: Instead of \( s_{ij} \) use \( w_{ij} \) that we estimate directly from data
Use a **weighted sum** rather than **weighted avg.**:

\[
\widehat{r}_{xi} = b_{xi} + \sum_{j \in N(i; x)} w_{ij}(r_{xj} - b_{xj})
\]

**A few notes:**

- **\(N(i; x)\)** ... set of movies rated by user \(x\) that are similar to movie \(i\)
- **\(w_{ij}\)** is the interpolation weight (some real number)
  - We allow: \(\sum_{j \in N(i, x)} w_{ij} \neq 1\)
- **\(w_{ij}\)** models interaction between pairs of movies (it does not depend on user \(x\))
Idea: Interpolation Weights $w_{ij}$

- $\hat{r}_{xi} = b_{xi} + \sum_{j \in N(i,x)} w_{ij} (r_{xj} - b_{xj})$

- How to set $w_{ij}$?
  - Remember, error metric is: $\frac{1}{|R|} \sqrt{\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2}$ or equivalently $\text{SSE}: \sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2$
  - Find $w_{ij}$ that minimize SSE on training data!
    - Models relationships between item $i$ and its neighbors $j$
    - $w_{ij}$ can be learned/estimated based on $x$ and all other users that rated $i$

Why is this a good idea?
- **Goal:** Make good recommendations
  - Quantify goodness using **RMSE:**
    - Lower RMSE $\Rightarrow$ better recommendations
  - Want to make good recommendations on items that user has not yet seen. Can’t really do this!

- Let’s set build a system such that it works well on known (user, item) ratings
  - And **hope** the system will also predict well the unknown ratings
Recommendations via Optimization

- Idea: Let’s set values $w$ such that they work well on known (user, item) ratings
- How to find such values $w$?
  - Idea: Define an objective function and solve the optimization problem
  - Find $w_{ij}$ that minimize $\text{SSE}$ on training data!

$$J(w) = \sum_{x,i} \left( b_{xi} + \sum_{j \in N(i; x)} w_{ij} (r_{xj} - b_{xj}) \right)^2 - r_{xi}$$

- Think of $w$ as a vector of numbers
A simple way to minimize a function $f(x)$:

- Compute the derivative $\nabla f$
- Start at some point $y$ and evaluate $\nabla f(y)$
- Make a step in the reverse direction of the gradient: $y = y - \nabla f(y)$
- Repeat until converged
Example: Formulation

- Assume we have a dataset with a single user $x$ and items 0, 1, and 2. We are given all ratings, and we want to compute the weights $w_{01}$, $w_{02}$, and $w_{03}$.

- Rating estimate: \[ \hat{r}_{xi} = b_{xi} + \sum_{j \in N(i,x)} w_{ij} (r_{xj} - b_{xj}) \]

  Training dataset already has the correct $r_{xi}$ values. We will use the estimation formula to compute the unknown weights $w_{01}$, $w_{02}$, and $w_{03}$.

- Optimization problem: Compute $w_{ij}$ values to minimize:
  \[ \sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2 \]

- Plug in the formulas:
  \[
  \text{minimize } J(w) = \left[ b_{x0} + w_{01} (r_{x1} - b_{x1}) + w_{02} (r_{x2} - b_{x2}) - r_{x0} \right]^2 \\
  + \left[ b_{x1} + w_{01} (r_{x0} - b_{x0}) + w_{12} (r_{x2} - b_{x2}) - r_{x1} \right]^2 \\
  + \left[ b_{x2} + w_{02} (r_{x0} - b_{x0}) + w_{12} (r_{x1} - b_{x1}) - r_{x2} \right]^2
  \]
Example: Algorithm

Initialize unknown variables:

\[ \mathbf{w}_{\text{new}} = \begin{bmatrix} w_{01}^{\text{new}} \\ w_{02}^{\text{new}} \\ w_{12}^{\text{new}} \end{bmatrix} = \begin{bmatrix} w_{01}^0 \\ w_{02}^0 \\ w_{12}^0 \end{bmatrix} \]

Iterate:

\[
\text{while } |\mathbf{w}_{\text{new}} - \mathbf{w}_{\text{old}}| > \varepsilon \\
\mathbf{w}_{\text{old}} = \mathbf{w}_{\text{new}} \\
\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \cdot \nabla J(\mathbf{w}_{\text{old}})
\]

\( \eta \) is the learning rate (a parameter)

How to compute \( \nabla J(\mathbf{w}_{\text{old}}) \)?
Example: Gradient-Based Update

\[ J(w) = \left[ b_{x0} + w_{01}(r_{x1} - b_{x1}) + w_{02}(r_{x2} - b_{x2}) - r_{x0} \right]^2 \]
\[ + \left[ b_{x1} + w_{01}(r_{x0} - b_{x0}) + w_{12}(r_{x2} - b_{x2}) - r_{x1} \right]^2 \]
\[ + \left[ b_{x2} + w_{02}(r_{x0} - b_{x0}) + w_{12}(r_{x1} - b_{x1}) - r_{x2} \right]^2 \]

\[ \nabla J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_{01}} \\ \frac{\partial J(w)}{\partial w_{02}} \\ \frac{\partial J(w)}{\partial w_{12}} \end{bmatrix} \]

\[
\begin{bmatrix} w_{01}^{\text{new}} \\ w_{02}^{\text{new}} \\ w_{12}^{\text{new}} \end{bmatrix} = \begin{bmatrix} w_{01}^{\text{old}} \\ w_{02}^{\text{old}} \\ w_{12}^{\text{old}} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial J(w)}{\partial w_{01}} \\ \frac{\partial J(w)}{\partial w_{02}} \\ \frac{\partial J(w)}{\partial w_{12}} \end{bmatrix}
\]

Each partial derivative is evaluated at \( w^{\text{old}} \).
Example: Computing Partial Derivatives

\[ J(w) = [b_{x0} + w_{01}(r_{x1} - b_{x1}) + w_{02}(r_{x2} - b_{x2}) - r_{x0}]^2 \]
\[ + [b_{x1} + w_{01}(r_{x0} - b_{x0}) + w_{12}(r_{x2} - b_{x2}) - r_{x1}]^2 \]
\[ + [b_{x2} + w_{02}(r_{x0} - b_{x0}) + w_{12}(r_{x1} - b_{x1}) - r_{x2}]^2 \]

Reminder: \( \frac{\partial((ax+b)^2)}{\partial x} = 2(ax+b)a \)

\[ \frac{\partial J(w)}{\partial w_{01}} = 2[b_{x0} + w_{01}(r_{x1} - b_{x1}) + w_{02}(r_{x2} - b_{x2}) - r_{x0}] (r_{x1} - b_{x1}) \]
\[ + 2 [b_{x1} + w_{01}(r_{x0} - b_{x0}) + w_{12}(r_{x2} - b_{x2}) - r_{x1}] (r_{x0} - b_{x0}) \]

Evaluate each partial derivative at \( w^{old} \) to compute the gradient direction.
We have the optimization problem, now what?

Gradient descent:

- Iterate until convergence: $w \leftarrow w - \eta \nabla_w J$
- $\eta$ … learning rate

where $\nabla_w J$ is the gradient (derivative evaluated on data):

$$\nabla_w J = \left[ \frac{\partial J(w)}{\partial w_{ij}} \right] = 2 \sum_{x, i} \left( \left[ b_{xi} + \sum_{k \in N(i; x)} w_{ik} (r_{xk} - b_{xk}) \right] - r_{xi} \right) (r_{xj} - b_{xj})$$

for $j \in \{N(i; x), \forall i, \forall x \}$

else $\frac{\partial J(w)}{\partial w_{ij}} = 0$

**Note:** We fix movie $i$, go over all $r_{xi}$, for every movie $j \in N(i; x)$, we compute $\frac{\partial J(w)}{\partial w_{ij}}$ while $|w_{new} - w_{old}| > \varepsilon$:

$w_{old} = w_{new}$

$w_{new} = w_{old} - \eta \cdot \nabla_w J$
Interpolation Weights

So far: \( \hat{r}_{xi} = b_{xi} + \sum_{j \in N(i; \chi)} w_{ij} (r_{xj} - b_{xj}) \)

- Weights \( w_{ij} \) derived based on their role; **no use of an arbitrary similarity measure** \( (w_{ij} \neq s_{ij}) \)
- Explicitly account for interrelationships among the neighboring movies

Next: Latent factor model
- Extract “regional” correlations
Performance of Various Methods

- Global average: 1.1296
- User average: 1.0651
- Movie average: 1.0533
- Netflix: 0.9514

Basic Collaborative filtering: 0.94
CF+Biases+learned weights: 0.91

Grand Prize: 0.8563
Latent Factor Models (e.g., SVD)

- The Color Purple
- Sense and Sensibility
- The Princess Diaries
- The Lion King
- Independence Day
- Funny
- Geared towards females
- Serious Amadeus
- Ocean’s 11
- Geared towards males
- Braveheart
- Lethal Weapon
- Dumb and Dumber
“SVD” on Netflix data: $R \approx Q \cdot P^T$

For now let’s assume we can approximate the rating matrix $R$ as a product of “thin” $Q \cdot P^T$

- $R$ has missing entries but let’s ignore that for now!
  - Basically, we will want the reconstruction error to be small on known ratings and we don’t care about the values on the missing ones
Ratings as Products of Factors

- How to estimate the missing rating of user $x$ for item $i$?

$$\hat{r}_{xi} = q_i \cdot p_x$$

$$= \sum_f q_{if} \cdot p_{xf}$$

$q_i = \text{row } i \text{ of } Q$

$p_x = \text{column } x \text{ of } P^T$
Ratings as Products of Factors

- How to estimate the missing rating of user $x$ for item $i$?

The estimated rating is given by:

$$\hat{r}_{xi} = q_i \cdot p_x$$

where $q_i$ is row $i$ of matrix $Q$ and $p_x$ is column $x$ of matrix $P^T$.
How to estimate the missing rating of user $x$ for item $i$?

$$\hat{r}_{xi} = q_i \cdot p_x = \sum_{f} q_{if} \cdot p_{xf}$$

$q_i = \text{row } i \text{ of } Q$
$p_x = \text{column } x \text{ of } P^T$
Latent Factor Models

The Color Purple

Sense and Sensibility

The Princess Diaries

Serious

Amadeus

Ocean’s 11

The Lion King

Factor 1

Factor 2

Geared towards males

Geared towards females

Lethal Weapon

Dumb and Dumber

Funny

Independence Day
Latent Factor Models

The Color Purple
Sense and Sensibility
The Princess Diaries

Serious
Amadeus

Geared towards females

Geared towards males

Ocean’s 11

Factor 1

Factor 2

The Lion King
Independence Day

Funny

Sense

Serious

Geared towards females

The Color Purple
Sense and Sensibility
The Princess Diaries

Braveheart
Lethal Weapon
Dumb and Dumber

Geared towards males

Sense

Dumb and Dumber

The Color Purple
Sense and Sensibility
The Princess Diaries

Sense

The Color Purple
Sense and Sensibility
The Princess Diaries

Dumb and Dumber

The Color Purple
Sense and Sensibility
The Princess Diaries

Dumb and Dumber

The Color Purple
Sense and Sensibility
The Princess Diaries

Dumb and Dumber

The Color Purple
Sense and Sensibility
The Princess Diaries

Dumb and Dumber
FYI, SVD:
- **A**: Input data matrix
- **U**: Left singular vecs
- **V**: Right singular vecs
- **Σ**: Singular values

So in our case:
“SVD” on Netflix data: \( R \approx Q \cdot P^T \)
\[ A = R, \quad Q = U, \quad P^T = \Sigma \cdot V^T \]
\[ \hat{r}_{xi} = q_i \cdot p_x \]
SVD: More good stuff

- We already know that SVD gives minimum reconstruction error (Sum of Squared Errors):

  \[
  \min_{U,V,\Sigma} \sum_{ij \in A} (A_{ij} - [U\Sigma V^T]_{ij})^2
  \]

- Note two things:
  - **SSE** and **RMSE** are monotonically related:
    - \[RMSE = \frac{1}{c} \sqrt{SSE}\]  
      Great news: SVD is minimizing RMSE
  - **Complication:** The sum in SVD error term is over all entries (no-rating in interpreted as zero-rating). But our \(R\) has missing entries!
Latent Factor Models

- SVD isn’t defined when entries are missing!
- Use specialized methods to find $P$, $Q$

$$
\min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - q_i \cdot p_x)^2
$$

\[ \hat{r}_{xi} = q_i \cdot p_x \]

- Note:
  - We don’t require cols of $P$, $Q$ to be orthogonal/unit length
  - $P$, $Q$ map users/movies to a latent space
  - The most popular model among Netflix contestants
Finding the Latent Factors
General Concept: Overfitting

Almost-linear data is fit to a linear function and a polynomial function.

Polynomial model fits perfectly to data.

Linear model has some error in the training set.

Linear model is expected to perform better on test data, because it filters out noise.

Our goal is to find $P$ and $Q$ such that:

$$\min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - q_i \cdot p_x)^2$$
Want to minimize SSE for unseen test data

Idea: Minimize SSE on training data
- Want large $k$ (# of factors) to capture all the signals
- But, SSE on test data begins to rise for $k > 2$

This is a classical example of overfitting:
- With too much freedom (too many free parameters) the model starts fitting noise
  - That is it fits too well the training data and thus not generalizing well to unseen test data
To solve overfitting we introduce regularization:

- Allow rich model where there are sufficient data
- Shrink aggressively where data are scarce

\[
\min_{P,Q} \sum_{\text{training}} (r_{xi} - q_i p_x)^2 + \left[ \lambda_1 \sum_x \| p_x \|^2 + \lambda_2 \sum_i \| q_i \|^2 \right]
\]

\( \lambda_1, \lambda_2 \ldots \) user set regularization parameters

**Note:** We do not care about the “raw” value of the objective function, but we care in \( P,Q \) that achieve the minimum of the objective
Regularization

\[
\min_{P,Q} \sum_{t \in \text{training}} (r_{xt} - q_ip_x)^2 + \left[ \lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]
\]

- \( \lambda_1, \lambda_2 \ldots \) user set regularization parameters
- "error"
- "length"

- What happens if the user \( x \) has rated hundreds of movies?
  The error term will dominate, and we’ll get a rich model
  Noise is less of an issue because we have lots of data

- What happens if the user \( x \) has rated only a few movies?
  Length term for \( p_x \) will have more effect, and we’ll get a simple model

- Same argument applies for items
The Effect of Regularization

The Color Purple  The Princess Diaries
Sense and Sensibility  The Lion King
Geared towards females  Geared towards males

braveheart  Lethal Weapon

Sense and Sensibility  Ocean’s 11

Geared towards females  Geared towards males

The Princess Diaries  The Lion King

min \[ \sum_{i} \left( r_{xi} - q_{pi} \right)^2 + \lambda \left( \sum_i \left\| p_i \right\|^2 + \sum_i \left\| q_i \right\|^2 \right) \]

min \[ \text{factors} \quad \text{“error” + \lambda “length”} \]
The Effect of Regularization

Geared towards females

The Color Purple
Sense and Sensibility

serious
Amadeus

Ocean’s 11

Lethal Weapon

Geared towards males

The Lion King

Dumb and Dumber

min \sum_{x \in \text{training}} (r_{xi} - q_i p_i)^2 + \lambda \left[ \sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]

min_{factors} “error” + \lambda “length”
The Effect of Regularization

Geared towards females

Geared towards males

Serious

Funny

Factor 1

Factor 2

The Color Purple

Sense and Sensibility

The Lion King

Independence Day

Lethal Weapon

Amadeus

The Princess Diaries

Dumb and Dumber

The Lion King

Ocean’s 11

Braveheart

\[
\min_{P,Q} \sum_{\text{training}} (r_{xi} - q_i p_x)^2 + \lambda \left[ \sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]
\]

\[
\min_{\text{factors}} \text{“error”} + \lambda \text{“length”}
\]
The Effect of Regularization

The Color Purple
Sense and Sensibility
Ocean’s 11
Geared towards females

The Princess Diaries
The Lion King
Independence Day
Geared towards males

serious
Braveheart
Lethal Weapon

funny
Dumb and Dumber
Geared towards males

min \sum_{p,q} \sum_{\text{training}} (r_{xi} - q_i p_x)^2 + \lambda \left[ \sum_i \|p_x\|^2 + \sum_i \|q_i\|^2 \right]

min_{\text{factors}} “error” + \lambda “length”
Want to find matrices $P$ and $Q$:

$$\min_{P,Q} \sum_{\text{training}} (r_{xi} - q_i p_x)^2 + \left[ \lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

Gradient descent:

- Initialize $P$ and $Q$ (using SVD, pretend missing ratings are 0)
- Do gradient descent:
  - $P \leftarrow P - \eta \cdot \nabla P$
  - $Q \leftarrow Q - \eta \cdot \nabla Q$
  - where $\nabla Q$ is gradient/derivative of matrix $Q$: $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x)p_{xf} + 2\lambda_2 q_{if}$
  - Here $q_{if}$ is entry $f$ of row $q_i$ of matrix $Q$

Observation: Computing gradients is slow!
Example

\[
\min_{P,Q} \sum_{x,i} (r_{xi} - q_i p_x)^2 + \left[ \lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right] \\
\]

Assume we want 3 factors per user and item: 
\[
p_x = \begin{bmatrix} p_{x0} \\ p_{x1} \\ p_{x2} \end{bmatrix} \quad q_i = \begin{bmatrix} q_{i0} \\ q_{i1} \\ q_{i2} \end{bmatrix}
\]

Rewrite objective as:

\[
\min \sum_{x,i} \left[ r_{xi} - (q_{i0} p_{x0} + q_{i1} p_{x1} + q_{i2} p_{x2}) \right]^2 \\
+ \lambda_1 \sum_x (p_{x0}^2 + p_{x1}^2 + p_{x2}^2) \\
+ \lambda_2 \sum_i (q_{i0}^2 + q_{i1}^2 + q_{i2}^2)
\]
Example

\[
\min \sum_{x,i} [r_{xi} - (q_{i0}p_{x0} + q_{i1}p_{x1} + q_{i2}p_{x2})]^2 \\
+ \lambda_1 \sum_x (p_{x0}^2 + p_{x1}^2 + p_{x2}^2) \\
+ \lambda_2 \sum_i (q_{i0}^2 + q_{i1}^2 + q_{i2}^2)
\]

\[
p_x = \begin{bmatrix} p_{x0} \\ p_{x1} \\ p_{x2} \end{bmatrix} \quad q_i = \begin{bmatrix} q_{i0} \\ q_{i1} \\ q_{i2} \end{bmatrix}
\]

Compute gradient for variable \(q_{i0}\):

\[
\nabla q_{i0} = \sum_{x,i} -2(r_{xi} - (q_{i0}p_{x0} + q_{i1}p_{x1} + q_{i2}p_{x2}))p_{x0} + 2\lambda_2 q_{i0}
\]

Do the same for every free variable
Gradient Descent - Computation Cost

\[ \nabla Q = [\nabla q_{if}] \quad \text{and} \quad \nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x) p_{xf} + 2\lambda_2 q_{if} \]

- How many free variables do we have?
  \((\text{# of users} + \text{# of items}) \cdot (\text{# of factors})\)
- Which ratings do we process to compute \(\nabla q_{if}\)?
  All ratings for item \(i\)
- Which ratings do we process to compute \(\nabla p_{xf}\)?
  All ratings for user \(x\)
- What is the complexity of one iteration?
  \(O(\text{# of ratings} \cdot \text{# of factors})\)
Stochastic Gradient Descent

- **Gradient Descent (GD):** Update all free variables in one step. Need to process all ratings.
- **Stochastic Gradient Descent (SGD):** Update the free variables associated with a single rating in one step.
  - Need many more steps to converge
  - Each step is much faster
  - In practice: SGD much faster than GD

- **GD:** $Q \leftarrow Q - \eta \left[ \sum r_{xi} \nabla Q(r_{xi}) \right]$
- **SGD:** $Q \leftarrow Q - \mu \nabla Q(r_{xi})$
Stochastic Gradient Descent

\[ \nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_ip_x)p_{xf} + 2\lambda_2 q_{if} \]

\[ \nabla p_{xf} = \sum_{x,i} -2(r_{xi} - q_ip_x)q_{if} + 2\lambda_1 p_{xf} \]

Which free variables are associated with rating \( r_{xi} \)?

\[
\begin{bmatrix}
p_{x0} \\
p_{x1} \\
\vdots \\
p_{xk}
\end{bmatrix}
\quad
\begin{bmatrix}
q_{i0} \\
q_{i1} \\
\vdots \\
q_{ik}
\end{bmatrix}
\]
Stochastic Gradient Descent

\[ \nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x) p_{xf} + 2\lambda_2 q_{if} \]

\[ \nabla p_{xf} = \sum_{x,i} -2(r_{xi} - q_i p_x) q_{if} + 2\lambda_1 p_{xf} \]

For each \( r_{xi} \):

\[ \varepsilon_{xi} = (r_{xi} - q_i \cdot p_x) \]  
\( \) (derivative of the “error”)

\[ q_i \leftarrow q_i + \mu_1 (\varepsilon_{xi} p_x - \lambda_2 q_i) \]  
\( \) (update equation)

\[ p_x \leftarrow p_x + \mu_2 (\varepsilon_{xi} q_i - \lambda_1 p_x) \]  
\( \) (update equation)

\( \mu \ldots \) learning rate

Note: The operations above are vector operations
Stochastic Gradient Descent

- **Stochastic gradient decent:**
  - Initialize \( P \) and \( Q \) (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

  For each \( r_{xi} \):
  - \( \varepsilon_{xi} = (r_{xi} - q_i \cdot p_x) \) (derivative of the “error”)
  - \( q_i \leftarrow q_i + \mu_1 (\varepsilon_{xi} p_x - \lambda_2 q_i) \) (update equation)
  - \( p_x \leftarrow p_x + \mu_2 (\varepsilon_{xi} q_i - \lambda_1 p_x) \) (update equation)

- **2 for loops:**
  - For until convergence:
    - For each \( r_{xi} \)
      - Compute gradient, do a “step”
Convergence of GD vs. SGD

- GD improves the value of the objective function at every step.
- SGD improves the value but in a “noisy” way.
- GD takes fewer steps to converge but each step takes much longer to compute.

In practice, SGD is much faster!
Extending Latent Factor Model to Include Biases
Modeling Biases and Interactions

- \( \mu \) = overall mean rating
- \( b_x \) = bias of user \( x \)
- \( b_i \) = bias of movie \( i \)

Baseline predictor
- Separates users and movies
- Benefits from insights into user’s behavior
- Among the main practical contributions of the competition

User-Movie interaction
- Characterizes the matching between users and movies
- Attracts most research in the field
- Benefits from algorithmic and mathematical innovations
Baseline Predictor

- We have expectations on the rating by user $x$ of movie $i$, even without estimating $x$’s attitude towards movies like $i$

- Rating scale of user $x$
- Values of other ratings user gave recently (day-specific mood, anchoring, multi-user accounts)
- (Recent) popularity of movie $i$
- Selection bias; related to number of ratings user gave on the same day (“frequency”)
Putting It All Together

\[ r_{xi} = \mu + b_x + b_i + q_i \cdot p_x \]

- **Example:**
  - Mean rating: \( \mu = 3.7 \)
  - You are a critical reviewer: your ratings are 1 star lower than the mean: \( b_x = -1 \)
  - Star Wars gets a mean rating of 0.5 higher than average movie: \( b_i = +0.5 \)
  - Predicted rating for you on Star Wars: 
    \[ = 3.7 - 1 + 0.5 = 3.2 \]
Fitting the New Model

- **Solve:**
  
  $$\min_{Q,P} \sum_{(x,i) \in R} \left( r_{xi} - (\mu + b_x + b_i + q_i p_x) \right)^2$$

  **goodness of fit**

  $$+ \left( \lambda_1 \sum_i \| q_i \|^2 + \lambda_2 \sum_x \| p_x \|^2 + \lambda_3 \sum_x \| b_x \|^2 + \lambda_4 \sum_i \| b_i \|^2 \right)$$

  $\lambda$ is selected via grid-search on a validation set

- **Stochastic gradient decent to find parameters**
  
  - **Note:** Both biases $b_x, b_i$ as well as interactions $q_i, p_x$ are treated as parameters (we estimate them)
Performance of Various Methods

Performance of Various Methods

Global average: 1.1296
User average: 1.0651
Movie average: 1.0533
Netflix: 0.9514
Basic Collaborative filtering: 0.94
Collaborative filtering++: 0.91
Latent factors: 0.90
Latent factors+Biases: 0.89
Grand Prize: 0.8563
The Netflix Challenge: 2006-09
Temporal Biases Of Users

- Sudden rise in the average movie rating (early 2004)
  - Improvements in Netflix
  - GUI improvements
  - Meaning of rating changed
- Movie age
  - Users prefer new movies without any reasons
  - Older movies are just inherently better than newer ones

Y. Koren, Collaborative filtering with temporal dynamics, KDD '09
Temporal Biases & Factors

- **Original model:**
  \[ r_{xi} = \mu + b_x + b_i + q_i \cdot p_x \]

- **Add time dependence to biases:**
  \[ r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x \]
  - Make parameters \( b_x \) and \( b_i \) to depend on time
  - (1) Parameterize time-dependence by linear trends
  - (2) Each bin corresponds to 10 consecutive weeks
    \[ b_i(t) = b_i + b_{i,\text{Bin}(t)} \]

- **Add temporal dependence to factors**
  - \( p_x(t) \)... user preference vector on day \( t \)
Adding Temporal Effects

![Graph showing the relationship between Millions of parameters and RMSE for different models: CF (no time bias), Basic Latent Factors, CF (time bias), Latent Factors w/ Biases, Linear time factors, Per-day user biases, and + CF. The graph illustrates how adding temporal effects generally improves the RMSE.]
Performance of Various Methods

- Global average: 1.1296
- User average: 1.0651
- Movie average: 1.0533
- Netflix: 0.9514

Basic Collaborative filtering: 0.94
Collaborative filtering++: 0.91
Latent factors: 0.90
Latent factors+Biases: 0.89
Latent factors+Biases+Time: 0.876

Grand Prize: 0.8563

Still no prize! 😞
Getting desperate.
Try a “kitchen sink” approach!
The big picture
Solution of BellKor's Pragmatic Chaos

All developed CF models
- BRISMF
- MF1
- NSVDD
- SVD-Time
- Split RBM
- BM3
- KNN
- Baseline
- MB
- Integrated M.
- RBM
- MF2
- SVD-AUF
- Movie KNN
- CTD/MTD
- SVDDNN
- User KNN
- Classifier
- ModeKNN 1
- Asym. 1/2/3

Latent User and Movie Features

Probe Blending

Approx. 500 predictors

Linear Blend 10.09% improvement

Probe Blending

200 blends

30 blends
Standing on June 26th 2009

June 26th submission triggers 30-day “last call”
The Last 30 Days

- **Ensemble team formed**
  - Group of other teams on leaderboard forms a new team
  - Relies on combining their models
  - Quickly also get a qualifying score over 10%

- **BellKor**
  - Continue to get small improvements in their scores
  - Realize that they are in direct competition with Ensemble

- **Strategy**
  - Both teams carefully monitoring the leaderboard
  - Only sure way to check for improvement is to submit a set of predictions
    - This alerts the other team of your latest score
Submissions limited to 1 a day
- Only 1 final submission could be made in the last 24h

24 hours before deadline...
- BellKor team member in Austria notices (by chance) that Ensemble posts a score that is slightly better than BellKor’s

Frantic last 24 hours for both teams
- Much computer time on final optimization
- Carefully calibrated to end about an hour before deadline

Final submissions
- BellKor submits a little early (on purpose), 40 mins before deadline
- Ensemble submits their final entry 20 mins later
- ....and everyone waits....
<table>
<thead>
<tr>
<th>Rank</th>
<th>Team Name</th>
<th>Best Test Score</th>
<th>% Improvement</th>
<th>Best Submit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BellKor's Pragmatic Chaos</td>
<td>0.8567</td>
<td>10.06</td>
<td>2009-07-26 18:18:28</td>
</tr>
<tr>
<td>2</td>
<td>The Ensemble</td>
<td>0.8567</td>
<td>10.06</td>
<td>2009-07-26 18:38:22</td>
</tr>
<tr>
<td>3</td>
<td>Grand Prize Team</td>
<td>0.8562</td>
<td>9.93</td>
<td>2009-07-10 10:42:44</td>
</tr>
<tr>
<td>4</td>
<td>Opera Solutions and Vandelay United</td>
<td>0.8588</td>
<td>9.84</td>
<td>2009-07-10 01:12:31</td>
</tr>
<tr>
<td>5</td>
<td>Vandelay Industries I</td>
<td>0.8591</td>
<td>9.81</td>
<td>2009-07-10 00:32:20</td>
</tr>
<tr>
<td>6</td>
<td>PragmaticTheory</td>
<td>0.8594</td>
<td>9.77</td>
<td>2009-06-24 12:06:56</td>
</tr>
<tr>
<td>7</td>
<td>BellKor in BigChaos</td>
<td>0.8601</td>
<td>9.70</td>
<td>2009-05-13 08:14:09</td>
</tr>
<tr>
<td>8</td>
<td>Dace</td>
<td>0.8612</td>
<td>9.59</td>
<td>2009-07-24 17:18:43</td>
</tr>
<tr>
<td>9</td>
<td>Feeds2</td>
<td>0.8622</td>
<td>9.48</td>
<td>2009-07-12 13:11:51</td>
</tr>
<tr>
<td>10</td>
<td>BigChaos</td>
<td>0.8623</td>
<td>9.47</td>
<td>2009-04-07 12:33:59</td>
</tr>
<tr>
<td>11</td>
<td>Opera Solutions</td>
<td>0.8624</td>
<td>9.46</td>
<td>2009-07-26 17:19:11</td>
</tr>
<tr>
<td>12</td>
<td>BellKor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Team Name</th>
<th>Best Test Score</th>
<th>% Improvement</th>
<th>Best Submit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>xianglang</td>
<td>0.8642</td>
<td>9.27</td>
<td>2009-07-15 14:53:22</td>
</tr>
<tr>
<td>14</td>
<td>Gravity</td>
<td>0.8643</td>
<td>9.26</td>
<td>2009-04-22 18:31:32</td>
</tr>
<tr>
<td>15</td>
<td>Ces</td>
<td>0.8651</td>
<td>9.18</td>
<td>2009-06-21 19:24:53</td>
</tr>
<tr>
<td>16</td>
<td>Invisible Ideas</td>
<td>0.8653</td>
<td>9.15</td>
<td>2009-07-15 15:53:04</td>
</tr>
<tr>
<td>17</td>
<td>Just a guy in a garage</td>
<td>0.8662</td>
<td>9.06</td>
<td>2009-05-24 10:02:54</td>
</tr>
<tr>
<td>18</td>
<td>J Dennis Su</td>
<td>0.8666</td>
<td>9.02</td>
<td>2009-03-07 17:16:17</td>
</tr>
<tr>
<td>19</td>
<td>Craig Carmichael</td>
<td>0.8666</td>
<td>9.02</td>
<td>2009-07-25 16:00:54</td>
</tr>
<tr>
<td>20</td>
<td>acmehill</td>
<td>0.8668</td>
<td>9.00</td>
<td>2009-03-21 16:20:50</td>
</tr>
</tbody>
</table>
Million $ Awarded Sept 21st 2009
Acknowledgments

- Some slides and plots borrowed from Yehuda Koren, Robert Bell and Padhraic Smyth
- **Further reading:**
  - Y. Koren, Collaborative filtering with temporal dynamics, KDD ’09
  - [http://www2.research.att.com/~volinsky/netflix/bpc.html](http://www2.research.att.com/~volinsky/netflix/bpc.html)
  - [http://www.the-ensemble.com/](http://www.the-ensemble.com/)