

INFO 251: Applied Machine Learning

Recommender Systems



Outline

- Recommender systems: Motivation
- The Netflix Challenge: Introduction
- Content-Based Recommendations
- Learning Features
- Neighborhood Methods
- The Netflix Challenge: The winners

Motivation

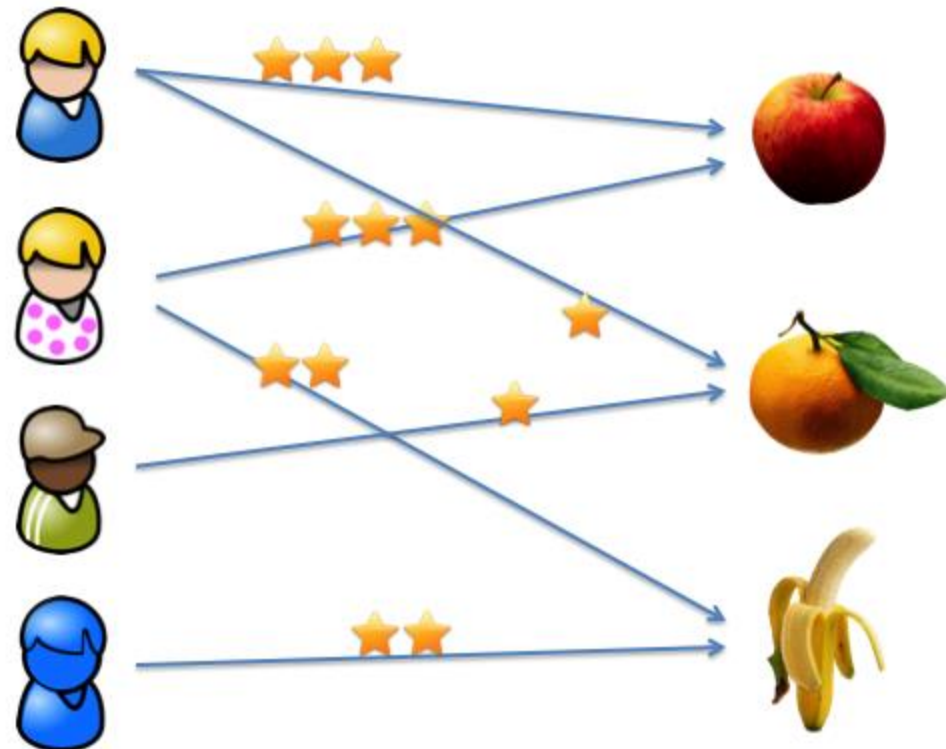
- Problem definition
 - Predict whether (or how much) a given person will like a given product
- Broad range of applications
 - Movie ratings
 - Amazon product recommendations
 - iTunes music suggestions
 - Friend recommendations
 - Supply chain optimization (predict product demand)

How to solve this problem?

- Worth brainstorming on this... (it's a billion dollar problem!)
 - Use features of the product
 - Past purchases of similar products
 - Use features of the person
 - Known demographic characteristics
 - Expressed preferences
 - Intrinsic preferences?
 - *Based on what people like you think*

Collaborative Filtering: Setup

- Bipartite Graph: two types of nodes, edges join nodes of different types



Collaborative Filtering

- Insight: Preferences are correlated across similar people
 - If Annie likes products X and Y and dislikes product Z, and Bob likes product X and dislikes Z, it's likely that Bob will like Y
- Goal: make predictions based on preferences of similar people
 - Find people similar to you, look at what they liked, and use that information to predict what you will like
- “Collaborative Filtering”
 - With Netflix, hundreds of millions of people are “collaborating” to determine what your tastes are and therefore what movies you are likely to enjoy

Outline

- Recommender systems: Motivation
- **The Netflix Challenge: Introduction**
- Content-Based Recommendations
- Learning Features
- Neighborhood Methods
- The Netflix Challenge: The winners

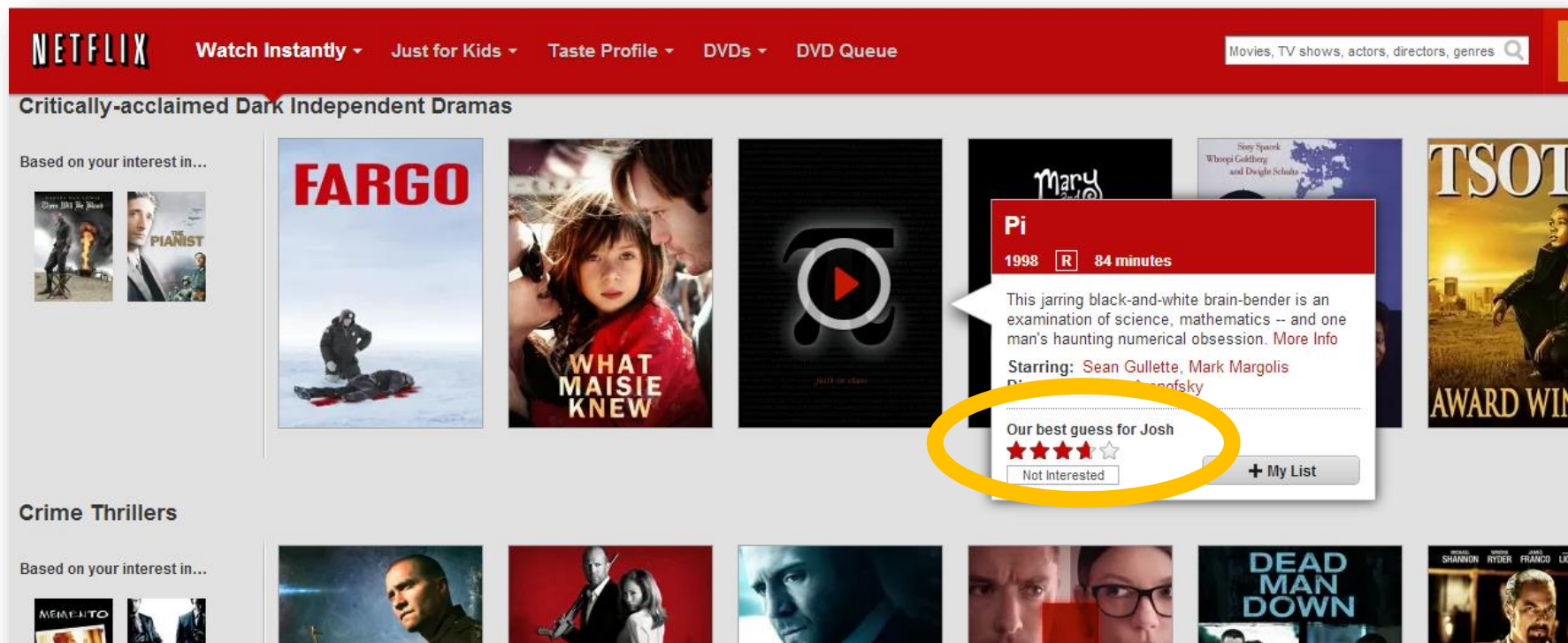
Example: the Netflix Challenge

- Goal: improve rating predictions of Netflix



How Netflix Works

- Netflix makes money when people pay to rent and watch movies
- Their algorithm predicts how much a given user will like a given movie:
- Netflix can make lots of \$\$\$ by improving quality of these ratings



The Netflix Challenge

- Training Data

- 100,480,507 ratings
- 480,189 users
- 17,770 movies
- $\langle \text{user}, \text{movie}, \text{date of grade}, \text{grade} \rangle$
- $\langle \text{integer}, \text{integer}, \text{date}, 1-5 \rangle$

- Test Data

- 2,817,131 ratings
- $\langle \text{integer}, \text{integer}, \text{date} \rangle$

- Objective: Minimize RMSE: $\sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2}$



What happened?

- October 2, 2006: Contest launched - \$1,000,000 first prize
 - Cinematch RMSE=0.9514
 - Naïve “average rating” RMSE: 1.0540
- October 8, 2006: “WXYZ” team beats Cinematch
- June 2007: 20,000 teams in competition (150 countries)
- September 2007: “BellKor:” RMSE=0.8728 (\$50k)
- September 2008: “BellKor:” RMSE=0.8616 (\$50k)
- July 2009: two teams hit 10% margin, no more entries
- September 2009: “BellKor’s Pragmatic Chaos” wins \$1,000,000 with RMSE 0.8567. Same RMSE matched by “The Ensemble”, but submission made 20 minutes later, tiebreaker went to BellKor’s Pragmatic Chaos.

Netflix Prize Winners

- BellKor's Pragmatic Chaos (Bellkor+BigChaos+Pragmatic Theory)



Outline

- Recommender systems: Motivation
- The Netflix Challenge: Introduction
- **Content-Based Recommendations**
- Learning Features
- Neighborhood Methods
- The Netflix Challenge: The winners

Content-based recommendations

- Problem definition
 - Predict whether (or how much) a given person will like a given product
 - AKA item-based recommendations
- “Typical” machine learning approach (still very common):
 - Learn person-feature weights
 - Does Annie like suspense movies?
 - Does Annie like movies with Leonardo di Caprio?
 - Etc.
- This is a reasonable approach!

Content-based recommendations: Example

- Given known features of each movie...
- And given knowledge of how users rate these movies...
- We can develop a model that predicts how a user will rate an unseen movie, given features of the movie

Movie Features	<i>Fargo</i>	<i>Finding Nemo</i>	<i>Flashdance</i>	<i>Footloose</i>	...
"Action"	4	3	1	1	...
"Romance"	1	4	5	5	...
...
Movie Ratings	Y_1	Y_2	Y_3	Y_5	...
Annie	4	5	4	?	...
Bob	1	5	5	?	...
Carol	3	4	1	?	...
Dave	2	1	2	?	...
...

Notation

- Formally, given k features (denoted by x_k) for each movie m :

$$x_m = \{x_{m1}, \dots, x_{mk}\}$$

- Our goal is to develop a model of Y_{im} given x_m

- Note $Y_{im} = f_i(x_m)$ is potentially different for every person

- Notation

- Y_{im} = Rating given by person i to movie m (e.g. "4")
- \hat{Y}_{im} = *Predicted* rating of person i for movie m (e.g. "4.1")
- x_{mp} = Feature p of movie m (e.g. "0.1", value for "Action")

Content-based recommendations: Example

- We can develop a model for each person, where $Y_{im} = f_i(x_m)$, and fit this model

- E.g. using a regression:

$$Y_{im} = \beta_0 + \beta_{i1}x_{m1} + \dots + \beta_{ik}x_{mk}$$

- How do we interpret parameters?
- Of course, any other classifier/algorithm works too!

$x_m = \{x_{m1}, \dots, x_{mk}\}$

{

Movie Features	<i>Fargo</i>	<i>Finding Nemo</i>	<i>Flashdance</i>
"Action"	4	3	1
"Romance"	1	4	5
"Budget"	7	94	7
...
Annie's Rating	4	5	4

Y_{im}

→

What's wrong with this approach?

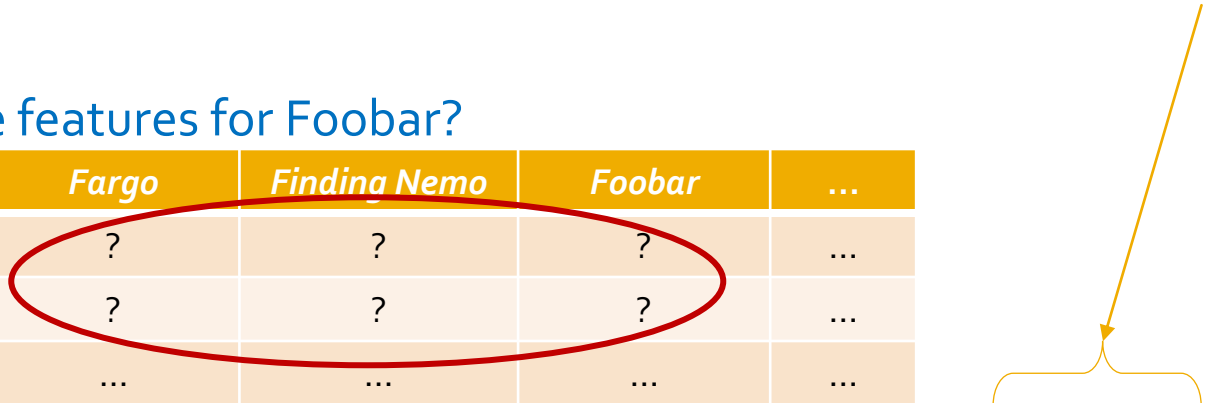
- Some important limitations
 - People's tastes are generally not that straightforward
 - Annie generally doesn't like suspense movies, but she likes suspense movies where Leonardo di Caprio almost dies but survives in the end
 - Sparsity of data
 - All users, initially, have no ratings. Even after rating hundreds of movies, have still only rated a very small percentage of universe of movies
 - How do we get features of the movies (the x_m)?
 - Difficult, time-consuming, and expensive to collect
 - Some features may be intangible, subjective, complex

Outline

- Recommender systems: Motivation
- The Netflix Challenge: Introduction
- Content-Based Recommendations
- **Learning Features**
- Neighborhood Methods
- The Netflix Challenge: The winners

Collaborative Filtering: Learning features

- What if we don't know the movie features x_m ?
 - For instance, whether *Finding Nemo* is "Action"
- Assume we do know preference parameters (i.e., the β_{ik}), we just don't know the movie features
 - What are reasonable features for Foobar?



Movie Features	<i>Fargo</i>	<i>Finding Nemo</i>	<i>Foobar</i>	...
"Action"	?	?	?	...
"Romance"	?	?	?	...
...

Movie Ratings	Y_1	Y_2	Y_3	...	$\beta_{iAction}$	$\beta_{iRomance}$
Annie	4	5	0	...	5	0
Bob	4	1	0	...	5	0
Carol	3	4	5	...	0	5
Dave	2	1	5	...	0	5
...

Sanity Check

- What would be a reasonable value of “Action” for the movie Freedom?

Movie Features	<i>Fargo</i>	<i>Finding Nemo</i>	<i>Flashdance</i>	Freedom
“Action”	?	?	?	?
“Romance”	?	?	?	?
...

Movie Ratings	Y_1	Y_2	Y_3	Y_4	$\beta_{iAction}$	$B_{iRomance}$
Annie	4	5	0	5	5	0
Bob	4	1	0	4	5	0
Carol	3	4	5	3	0	5
Dave	2	1	5	3	0	5
...

Learning Features

- Intuitively: given knowledge of users' ratings and preferences, we should be able to figure out the movie features
- Learning movie features is akin to learning the β_{ik}
 - Rather than trying to infer individual preferences (given ratings of movies with known features),
 - We are given β_{ik} and want to infer movie features x_m
- Before, our task was:

$$Y_{im} = \beta_0 + \beta_{i1}x_{m1} + \dots + \beta_{ik}x_{mk}$$

- Before, the beta values were unknown
- Now, the x values are unknown:

$$Y_{im} = \gamma_0 + x_{m1}\hat{\beta}_{i1} + \dots + x_{mk}\hat{\beta}_{k1}$$

Putting it together

- Collaborative filtering
 - Given features, we can learn user preferences (item-based)
 - Given user preferences, we can learn features (user-based)
- In practice, we can “guess” (randomly initialize) preference parameters and use that to estimate features; then use estimated features to estimate parameters, etc.
 - Guess $\beta \Rightarrow x \Rightarrow \beta \Rightarrow x \Rightarrow \dots$
 - This is the basic “collaborative filtering”, iterative algorithm
- Note: similar to Low Dimensional Rank Factorization
 - LDMF combines the two steps into a single optimization
 - Minimizes two sources of error simultaneously

Outline

- Recommender systems: Motivation
- The Netflix Challenge: Introduction
- Content-Based Recommendations
- Learning Features
- **Neighborhood Methods**
- The Netflix Challenge: The winners

Nearest-Neighbor C.F.

- One of most popular and common methods for CF
 - Also very intuitive to understand
 - very similar to kNN and kNN regression
- Two related intuitions:
 - **Similar** people rate same item similarly (user-based CF)
 - Same person rates **similar** items similarly (item-based CF)

Toy example

- Given these data, infer missing values
 - (Annie, Y_4)
 - Probably, you used user-based CF
 - With item-based CF, not as obvious
 - (Carol, Y_1)
 - Here, item-based is more natural

Movie	Y_1	Y_2	Y_3	Y_4	Y_5	...
Annie	4	5	5	-	4	...
Bob	4	-	5	2	4	...
Carol	-	4	1	-	1	...
Dave	3	-	-	5	3	...
...

How it works: The Basics

- Bare-bones version (good for intuition, not in practice)

$$\hat{Y}_{im} = \alpha \sum_{x_j \in N_i} w_{ij}(Y_{jm})$$

- Predicted rating \hat{Y}_{im} of user i for movie m is weighted sum of all neighbor j 's ratings of m
- Interpretation
 - If i and j are similar, and j rated m highly, then predict i will rate m highly
 - Very similar to distance-weighted k-NN regression

How it works: basics

- Bare-bones version (good for intuition, not in practice)

$$\hat{Y}_{im} = \alpha \sum_{x_j \in N_i} w_{ij} (Y_{jm})$$

- How to determine w_{ij} (similarity weights)?

$$w_{ij} = r = \frac{\sum_m (Y_{im} - \bar{Y}_i)(Y_{jm} - \bar{Y}_j)}{\sqrt{\sum_m (Y_{im} - \bar{Y}_i)^2 (Y_{jm} - \bar{Y}_j)^2}}$$

- Pearson CC: Covariance divided by product of standard deviations
- Measure of whether two variables go up and down in sync or not
- Numerator: larger if i rates above i 's mean when j rates above j 's mean
- Denominator normalizes by overall variance: some people are more reactive (1's and 5's), some people are more measured (3's and 4's). If these differences are highly correlated, they are still highly informative

How it works: Details

■ Details and nuances

- Basic version doesn't account for baseline preferences
- Normalization makes a huge difference:

$$\hat{Y}_{im} = \bar{Y}_i + \alpha \sum_{X_j \in N_i} W_{ij} (Y_{jm} - \bar{Y}_j)$$

- Average rating given to all movies by i
- Plus **normalized**, weighted sum of neighbors' ratings of j
- Have to define which neighbors are included
 - Need overlap in movies. Can't compute similarity if rated different movies
 - Art+Science. Use odd numbers (to avoid ties). Tens of neighbors?

Other Issues

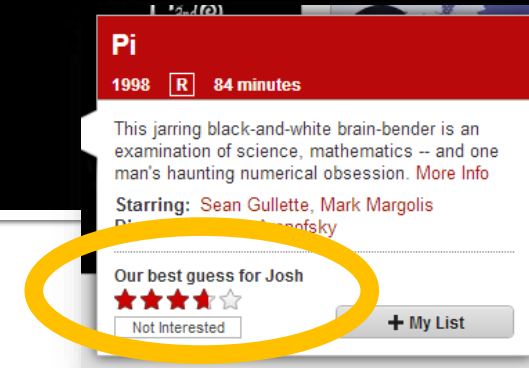
- “Cold Start” problem – what to do with a new user / new item?
 - Use a group average (e.g. \bar{Y}_j)
 - Collect booster data (e.g., 20 questions)
- Top recommendations may be very redundant, “too obvious”
 - People who liked Rocky 1 will see Rocky 2, Rocky 3, Rocky 4, Rocky N
 - They may have seen these and not rated them
 - Often a goal of recommendations is content discovery
- Diverse recommendations are useful
 - Users are multi-faceted, we may want to hedge our bets
 - If we just bought a laptop, don’t want to recommend another laptop!

How it works: Details

- Instance-based method using Pearson correlation is effective, common
- Lots of possible extensions:
 - Remove bias term from each rating before applying CF
 - **User's mean rating**, or global mean rating, or item's mean rating
 - Item's mean rating + user's mean deviation from item mean
 - Measuring similarity between people (or items)
 - Pearson correlation coefficient
 - Cosine similarity: [measure of angle between two vectors](#)
 - Inverse Euclidean (or other) distance
 - Note: These metrics assume complete data. In practice, can only compute over set of items rated by both users

How it works: Details

- Implicit vs. explicit ratings
 - Strongest signal is in explicit ratings made by people
 - Other sources of information
 - Whether a person rented the movie (or bought the product)
 - Whether the person viewed the page, watched the trailer
 - Etc.



Outline

- Recommender systems: Motivation
- The Netflix Challenge: Introduction
- Content-Based Recommendations
- Learning Features
- Neighborhood Methods
- **The Netflix Challenge: The winners**

Lecture so far

- Introduced CF
- Netflix Prize
- Content-based recommendations (regression)
- Learning features and CF
- Neighborhood methods
- Now: back to the Netflix prize

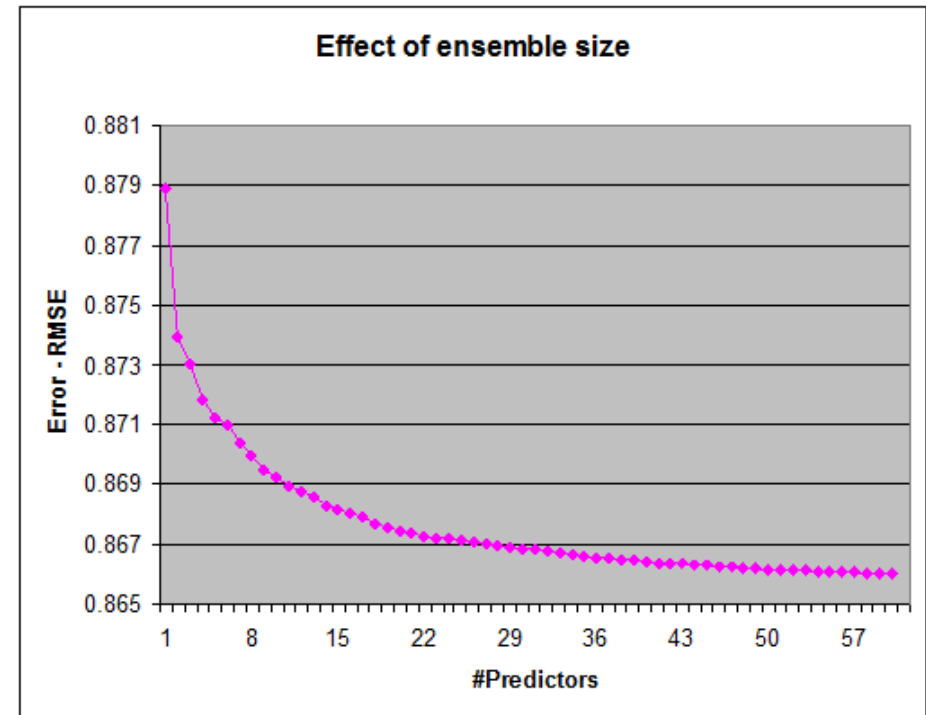


Netflix: The winner

- Core was instance-based, neighborhood methods
 - Main component of top two programs
- With some important tweaks
 - Using implicit data (e.g., which movies got ratings)
 - Correcting for neighborhood definition
 - E.g.: to reduce bias by the fact that people's neighbors also tend to be neighbors, and there can be herd behavior
- And Matrix Factorization
 - Identifies "latent factors" of movies in data, takes a more global approach to the problem than the neighborhood approach
 - A modified version of Singular Value Decomposition that allows for missing elements of the rating matrix
 - Same SVD that we discussed in relation to PCA

Netflix: The winner

- Final product used ensemble methods, which combined results from several algorithms
 - Neighborhood methods
 - Matrix decomposition methods
 - Regression
 - Gradient boosted trees
 - Etc



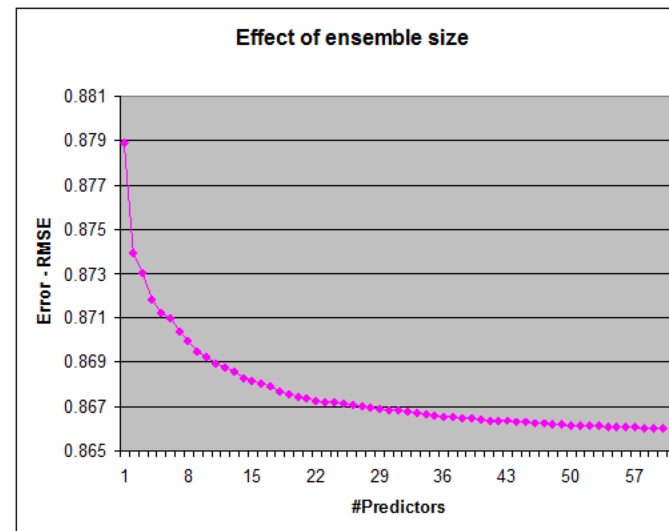
Feature engineering!

- What other features matter?
 - The number of other people who have rated the movie
 - The (log of the) number of days since user's first rating
 - The number of days since the movie's first rating
 - Implicit vs. explicit ratings
 - Whether a person watched (the full) movie
 - Whether the person clicked on the link
- Other things matter too
 - Money and resources!
 - Ensembles

Symphony or quartet?

- However, we would like to stress that it is not necessary to have such a large number of models to do well. The plot below shows RMSE as a function of the number of methods used. One can achieve our winning score (RMSE=0.8712) with less than 50 methods, using the best 3 methods can yield $RMSE < 0.8800$, which would land in the top 10. Even just using our single best method puts us on the leaderboard with an RMSE of 0.8890. The lesson here is that having lots of models is useful for the incremental results needed to win competitions, but practically, excellent systems can be built with just a few well-selected models.

- Yehuda Koren (2009)



- Graph is from 2007

Collaborative Filtering: Summary

- Generic technique for recommending items of one class to items of another class
 - Leverages preferences embedded in graph
 - Broad range of (lucrative) applications

