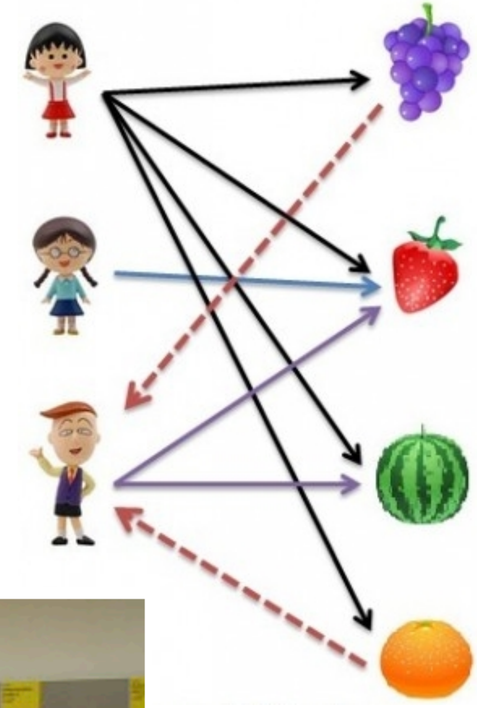
Lecture 5: Recommendation

Example



- Diapers and beer

diapers and beer. http://www.forbes.com/forbes/1998/0406/6107128a.html

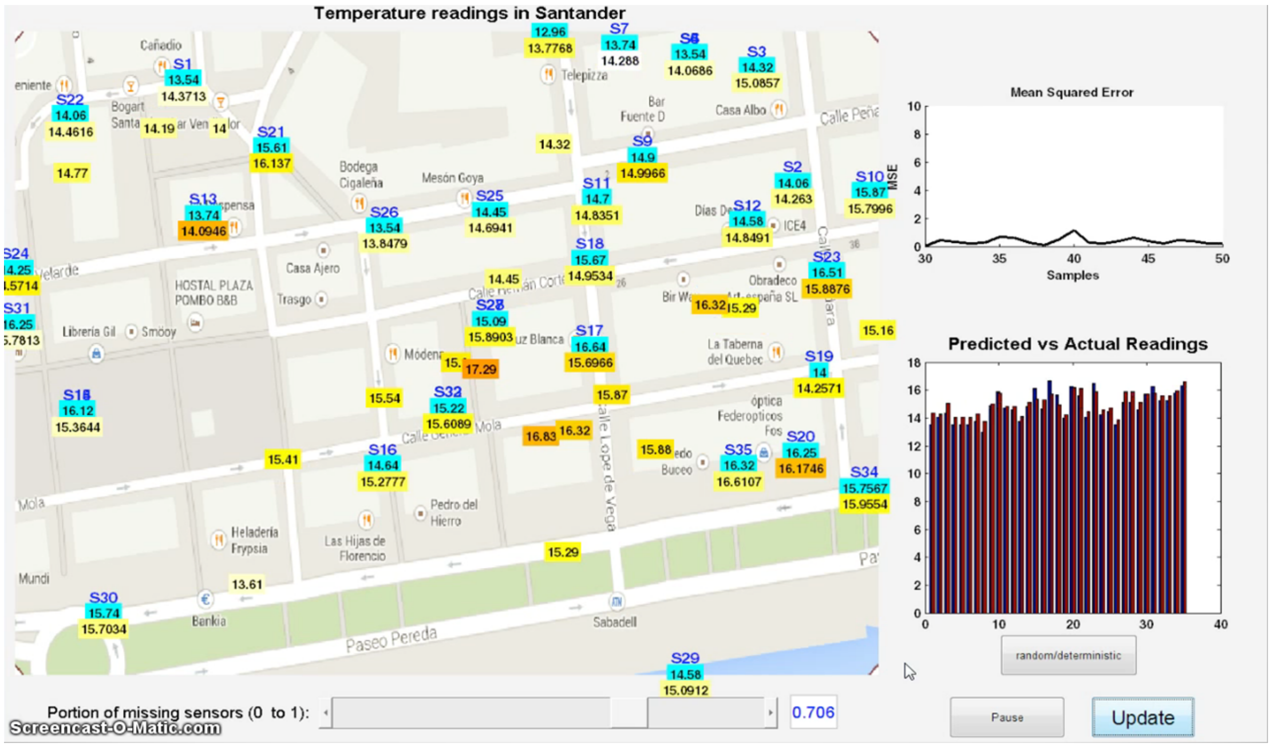
Examples

- Netflix Prize 2009

- Amazon Recommendation Engine

Examples

- Missing sensor data



Collaborative Filtering

- 400,000 users
  17,000 movies
  But only a few ratings (1%)
- User a
  Movie i
  Rating $Y_{ai} \in \mathbb{R}$ (e.g. 1-5)
- Training data is the incomplete matrix Y
- Goal is to predict unobserved ratings

m movies

n users

5	5						5		
		3	5	1	3	4	4		4
	4	2			2				
		5							5
4	5							4	
4							4		
5		4	5	1		4			
	4								
5				4					
5						4			
		5				5		3	

Y_{ai}

Collaborative Filtering

- **Matrix or tensor completion problems**
- Collaborative: cross-users Filtering: prediction
- Dimensionality reduction

m movies

n users

5	5						5		
		3	5	1	3	4	4		4
	4	2			2				
		5							5
4	5							4	
4							4		
5		4	5	1		4			
	4								
5				4					
5						4			
		5				5		3	

Y_{ai}

A tensor is a multi-dimensional array.
e.g. $n \times m$ 2-tensor
e.g. $p \times q \times r$ 3-tensor
(2-tensor = matrix)

Users who share similar interests on an item in the past are more likely to hold similar opinions on other items compared with a randomly chosen user.

* not in syllabus

Types

Model based (Treat Y_{ai} as responses)

- Given training data of the form $\big((a,i),Y_{ai}\big)$,
  find a function $f\colon\{1,\dots,n\}\times\{1,\dots,m\}\to\mathbb{R}$.

Memory based (Treat Y_{ai} as features)

- Given (incomplete) user ratings $Y_a\in\mathbb{R}^m$,
  find structure in the data to predict missing values.

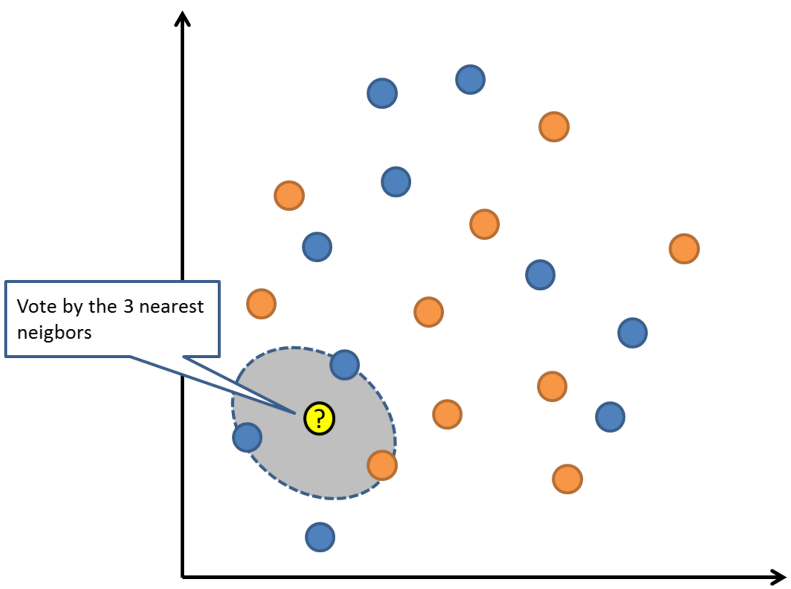
Recommendation is a *task* for which you can use either model based or memory based learning algorithms.

K-Nearest Neighbors

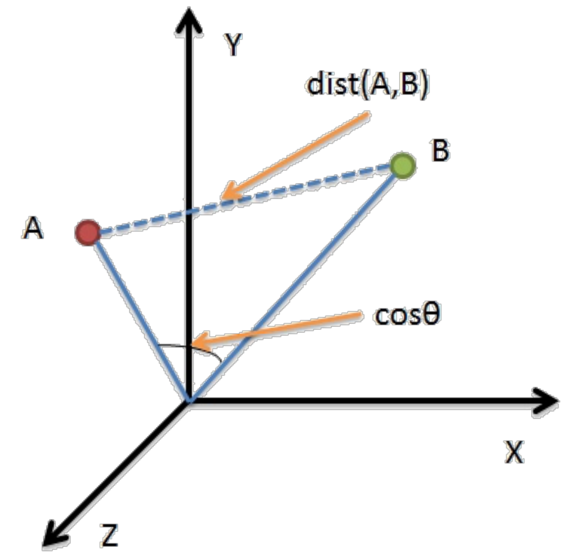
k-Nearest Neighbors

Main Idea.

- Find a few users $b_1, \ldots, b_k$ (neighbors) that are similar to user a
- Use information from users $b_1, \ldots, b_k$ to predict ratings of user a



Correlation Coefficient



Cosine Similarity

$$\cos(x, y) = \frac{x^{\top} y}{\|x\| \, \|y\|}$$

Correlation Coefficient

$$\text{corr}(x, y) = \cos(x - \bar{x}, y - \bar{y}) = \frac{(x - \bar{x})^{\top}(y - \bar{y})}{\|x - \bar{x}\| \, \|y - \bar{y}\|}$$

where $\bar{x}, \bar{y}$ are the averages of the entries of x, y.

User Similarity

To compute the similarity between users a and b,

1. Find $\mathrm{CR}(a, b)$, the set of movies rated by both a and b.
2. Let Z_a , Z_b be the vector of ratings in $\mathrm{CR}(a, b)$ for each user.
3. Compute the correlation coefficient between Z_a and Z_b.

$$\mathrm{sim}(a, b) = \mathrm{corr}(Z_a, Z_b) \in [-1, 1]$$

CR = "Common Ratings"

Weighted Prediction

Let $\bar{Y}_b$ denote the average of movie ratings by user b.

To predict the rating Y_{ai} of user a for movie i,

1. Rank users b who rated movie i according to value of $\mathrm{sim}(a, b)$.
2. Let $\mathrm{kNN}(a, i)$ be the set of highest k users.
3. Let $(Y_{ai} - \bar{Y}_a)$ be weighted average of $(Y_{bi} - \bar{Y}_b), b \in \mathrm{kNN}(a, i)$.
4. Let weight for $(Y_{bi} - \bar{Y}_b)$ be proportional to $\mathrm{sim}(a, b)$.

Anti-Correlated Users

If $(Y_{bi} - \bar{Y}_b)$ is strongly anti-correlated with $(Y_{ai} - \bar{Y}_a)$, i.e.

$$\text{sim}(a, b) \ll 0,$$

then $-(Y_{bi} - \bar{Y}_b)$ is strongly correlated with $(Y_{ai} - \bar{Y}_a)$.

We should exploit this information because ratings are rare.

Weighted Prediction

Let $\bar{Y}_b$ denote the average of movie ratings by user b.

To predict the rating Y_{ai} of a user a for a movie i,

1. Rank users b who rated movie i according to value of $|\mathrm{sim}(a, b)|$.
2. Let $\mathrm{kNN}(a, i)$ be the set of highest k users.
3. Let $(Y_{ai} - \bar{Y}_a)$ be weighted average of $\pm(Y_{bi} - \bar{Y}_b), b \in \mathrm{kNN}(a, i)$.
4. Let weight for $\pm(Y_{bi} - \bar{Y}_b)$ be proportional to $|\mathrm{sim}(a, b)|$.

$$\hat{Y}_{ai} - \bar{Y}_a = \frac{\sum_{b \in \mathrm{kNN}(a,i)} \mathrm{sim}(a,b)(Y_{bi} - \bar{Y}_b)}{\sum_{b \in \mathrm{kNN}(a,i)} |\mathrm{sim}(a,b)|}$$

Discussion

$$\hat{Y}_{ai} - \bar{Y}_a = \frac{\sum_{b \in \mathrm{kNN}(a,i)} \mathrm{sim}(a,b)(Y_{bi} - \bar{Y}_b)}{\sum_{b \in \mathrm{kNN}(a,i)} |\mathrm{sim}(a,b)|}$$

- Formula is not sensitive to the bias (mean) of each user, but it is sensitive to the spread (variance) of each user.
- There is no training loss, no training algorithm for kNN.

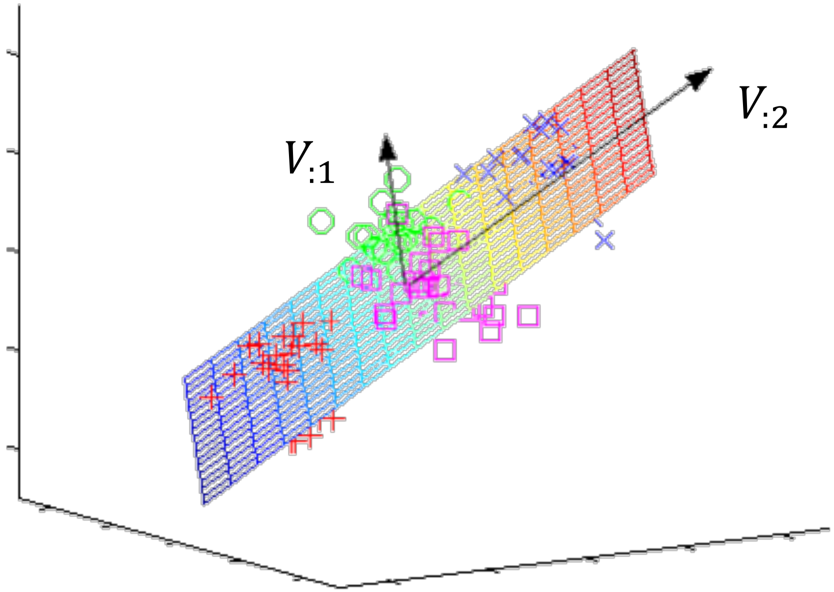
Matrix Factorization

Subspace Learning

$k \ll m$ means k is much smaller than m

Main idea.

- *Completed* rating vectors $\hat{Y}_1, \ldots, \hat{Y}_n \in \mathbb{R}^m$ lie in some k-dimensional subspace, $k \ll m$.
- i.e. there exists vectors $V_{:1}, \ldots, V_{:k} \in \mathbb{R}^m$ such that for all users a,



$$\hat{Y}_a = U_{a1} V_{:1} + U_{a2} V_{:2} + \ldots + U_{ak} V_{:k}$$

for some coefficients $U_{a1}, \ldots, U_{ak} \in \mathbb{R}$.

Matrix Factorization

$$
\begin{aligned}
\hat{Y}_1^\top &= U_{11}\, V_{:1}^\top + \cdots + U_{1k}\, V_{:k}^\top \\
\hat{Y}_2^\top &= U_{21}\, V_{:1}^\top + \ldots + U_{2k}\, V_{:k}^\top \quad \Rightarrow \\
&\vdots \\
\hat{Y}_n^\top &= U_{n1}\, V_{:1}^\top + \cdots + U_{nk}\, V_{:k}^\top
\end{aligned}
$$

$$
\begin{aligned}
\hat{Y} &= U_{:1}\, V_{:1}^\top + \cdots + U_{:k}\, V_{:k}^\top \\
&= (U_{:1}, \ldots, U_{:k}) \begin{pmatrix} V_{:1}^\top \\ \vdots \\ V_{:k}^\top \end{pmatrix} \\
&= (U_{:1}, \ldots, U_{:k})(V_{:1}, \ldots, V_{:k})^\top \\
&= UV^\top
\end{aligned}
$$

where $U_{:1}, \ldots, U_{:k}$ are the columns of $U \in \mathbb{R}^{n\times k}$,
and $V_{:1}, \ldots, V_{:k}$ are the columns of $V \in \mathbb{R}^{m\times k}$.

Matrix Rank

outer product!

Definition. A matrix $Y \in \mathbb{R}^{n \times m}$ is of rank-one if $Y = ab^\top$ for some non-zero vectors $a \in \mathbb{R}^n, b \in \mathbb{R}^m$.

Definition. The rank of a matrix Y is the smallest number k of rank-one matrices required in the sum

$$Y = Y^{(1)} + \cdots + Y^{(k)}.$$

If $Y = UV^\top$ with $U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}$, i.e.

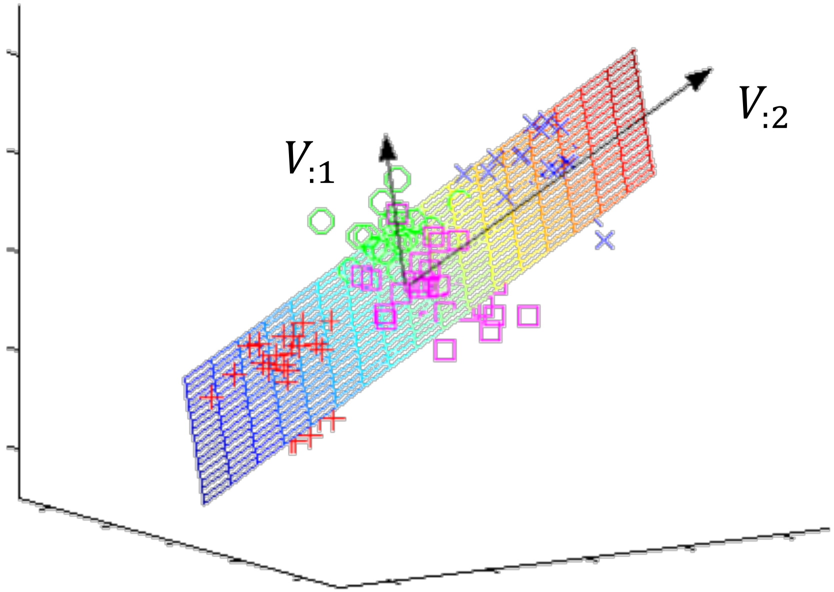
$$Y = UV^\top = U_{:1} V_{:1}^\top + U_{:2} V_{:2}^\top + \ \ldots + \ U_{:k} V_{:k}^\top \ ,$$

then the rank Y is at most k.

Low-Rank Approximation

Main Idea.

- Find low-rank *complete* matrix $\hat{Y}$ that is closest to *incomplete* matrix Y.
- $\hat{Y}$ is called a *low-rank approximation* of Y.



Overall idea – movie recommendation

- $\hat{Y}_{ai} = U_a^\top V_i$
- When watching movies, user a has preferences on
  - Comedy, sci-fi, action, romance (k factors)
  - $U_a \in \mathbb{R}^k$ characterizes user a's preference on these k factors
- Movie i can also be described by
  - Comedy, sci-fi, action, romance (k factors)
  - $V_i \in \mathbb{R}^k$ describes how the k factors distribute in movie i
- When we do the multiplication $U_a^\top V_i$, we obtain how much user a like movie i

Prediction

For an unknown rating Y_{ai} of user a for movie i, we predict

$$\hat{Y}_{ai} = (UV^\top)_{ai} = U_a V_i^\top$$

where U_a is the a-th row of U and V_i is the i-th row of V.

U_a: factor vector of user a.

V_i: factor vector of movie i.

Compare this with

$$\hat{Y} = U_{:1}\, V_{:1}^\top + U_{:2}\, V_{:2}^\top + \ \dots + \ U_{:k}\, V_{:k}^\top$$

where $U_{:j}$ is the j-th column of U and $V_{:j}$ is the j-th column of V.

Training Loss

We regularize because there are infinitely many solutions $U' = cU, V' = c^{-1}V$

Apply squared loss to each observed rating Y_{ai} .

$$\mathcal{L}_{n,k,\lambda}(U,V;Y) = \sum_{(a,i)\in D} \frac{1}{2}(Y_{ai} - (UV^\top)_{ai})^2 + \frac{\lambda}{2}\|U\|^2 + \frac{\lambda}{2}\|V\|^2$$
$$= \sum_{(a,i)\in D} \frac{1}{2}(Y_{ai} - U_a^\top V_i)^2 + \frac{\lambda}{2}\sum_a \|U_a\|^2 + \frac{\lambda}{2}\sum_i \|V_i\|^2$$

where D is the set of (a,i) where Y_{ai} is observed.

Definition. The *Frobenius* norm $\|U\|$ of a matrix $U \in \mathbb{R}^{n\times k}$ is

$$\|U\| = \sqrt{\sum_{a=1}^{n} \sum_{b=1}^{k} U_{ab}^2} .$$

Alternating Least Squares

Coordinate Descent (optimization).

Repeat until convergence:

1. Fix V and minimize $\mathcal{L}_{n,k,\lambda}(U, V; Y)$ over U.
2. Fix U and minimize $\mathcal{L}_{n,k,\lambda}(U, V; Y)$ over V.

Alternating Least Squares

1. Initialize $V_1, V_2, \ldots, V_m \in \mathbb{R}^k$ randomly.
2. Repeat until convergence:
    a. For each user a, find U_a that minimizes
    $$\sum_{i:\ (a,i)\in D} \frac{1}{2}(Y_{ai} - U_a^\top V_i)^2 + \frac{\lambda}{2}\|U_a\|^2$$
    b. For each movie i, find V_i that minimizes
    $$\sum_{a:\ (a,i)\in D} \frac{1}{2}(Y_{ai} - U_a^\top V_i)^2 + \frac{\lambda}{2}\|V_i\|^2$$

These are standard linear regression problems.

Discussion

User Bias

- Do we subtract the average ratings of each user?

Optimization.

- Like k-means, the algorithm converges to a local minimum.
- Perform multiple initializations, and pick best result.

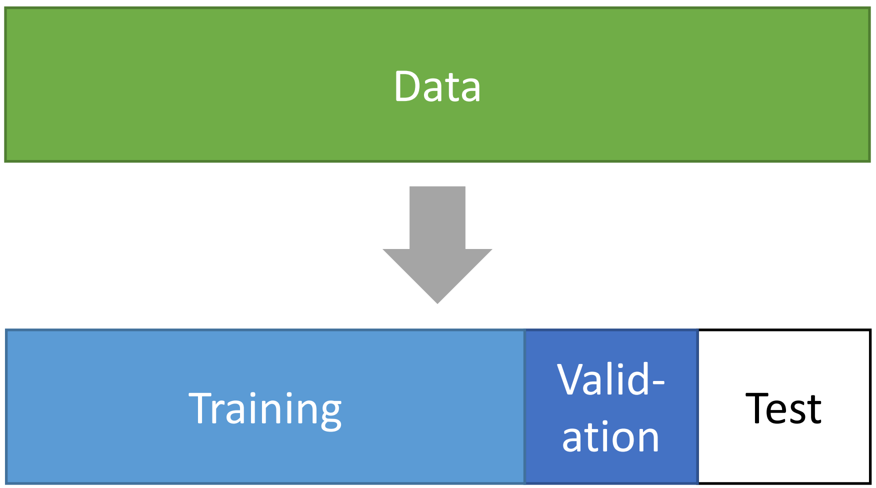
Generalization.

- Use validation to pick right hyperparameters k and λ.

Validation Set

Split the data into

- **Test set** $\mathcal{S}_*$
  For evaluating, reporting performance at the end
- **Training set** $\mathcal{S}_n$
  For training optimal parameters in a model
- **Validation set** $\mathcal{S}_{\text{val}}$
  For model selection,
  e.g. picking k in k-means,
  picking λ in ridge regression.
  Acts as a proxy for test set.



Validation Loss

The *validation error* is the test loss applied to the validation set.

The *training error* is the test loss applied to the training set, and it may be different from the training loss used for optimization.

Example. Ridge Regression

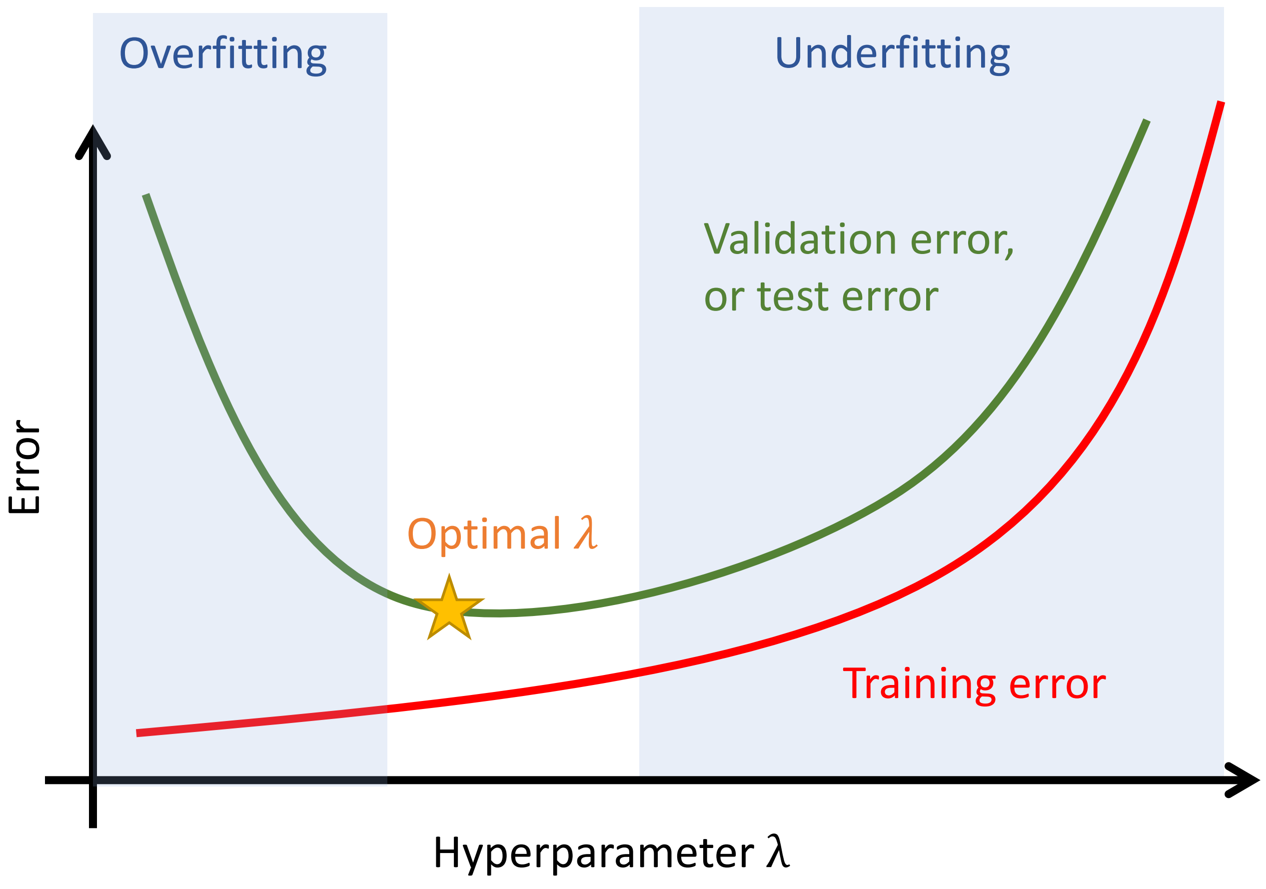
Test loss/error $\mathcal{R}\left(\hat{\theta}; \mathcal{S}_*\right) = \frac{1}{n}\sum_{(x,y)\in\mathcal{S}_*} \frac{1}{2}\left(y - \hat{\theta}^\top x\right)^2$

Validation loss/error $\mathcal{R}\left(\hat{\theta}; \mathcal{S}_{\text{val}}\right) = \frac{1}{n}\sum_{(x,y)\in\mathcal{S}_{\text{val}}} \frac{1}{2}\left(y - \hat{\theta}^\top x\right)^2$

Training error $\mathcal{R}\left(\hat{\theta}; \mathcal{S}_n\right) = \frac{1}{n}\sum_{(x,y)\in\mathcal{S}_n} \frac{1}{2}\left(y - \hat{\theta}^\top x\right)^2$

Training loss $\mathcal{L}_{n,\lambda}(\theta; \mathcal{S}_n) = \frac{1}{n}\sum_{\text{data }(x,y)} \frac{1}{2}(y - \theta^\top x)^2 + \frac{\lambda}{2}\|\theta\|^2$

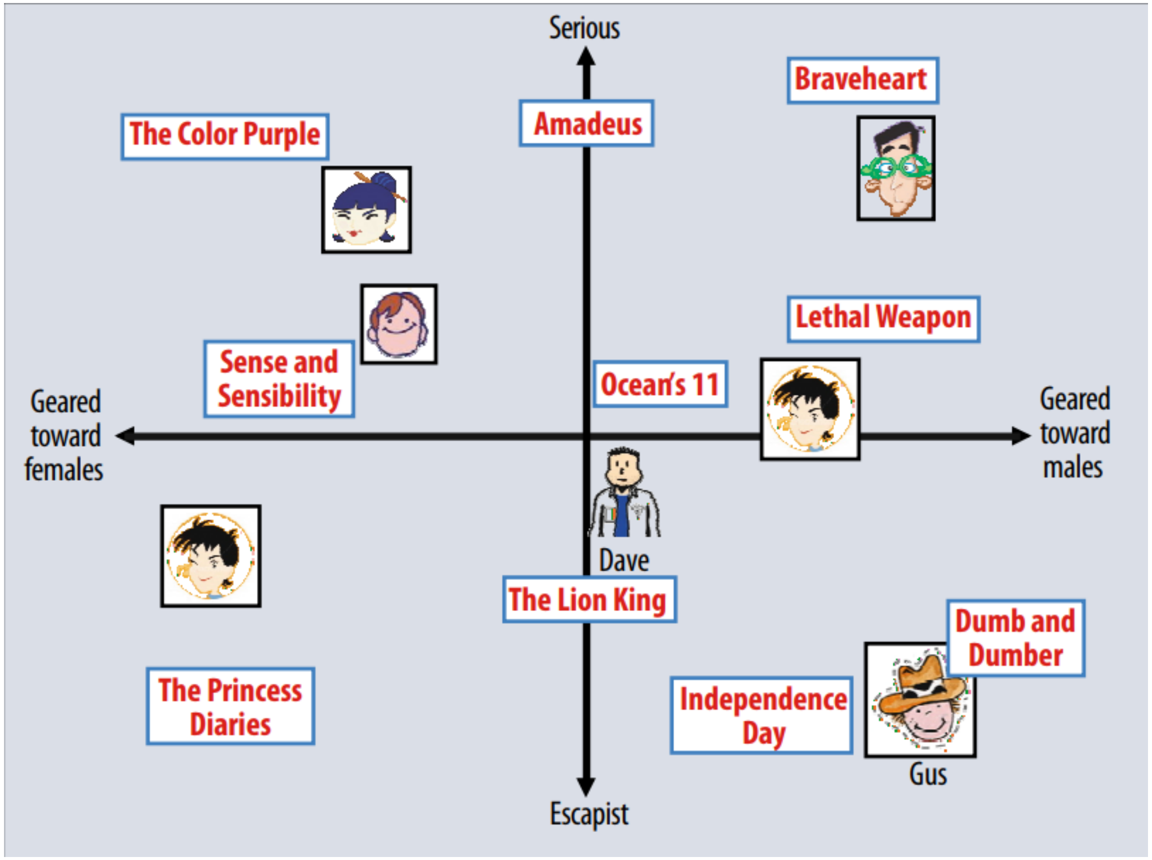
Model Selection



* not in syllabus

Netflix Results

Plot each movie i according to first two coordinates of $V_i \in \mathbb{R}^k$, i.e. V_{i1} , V_{i2}.

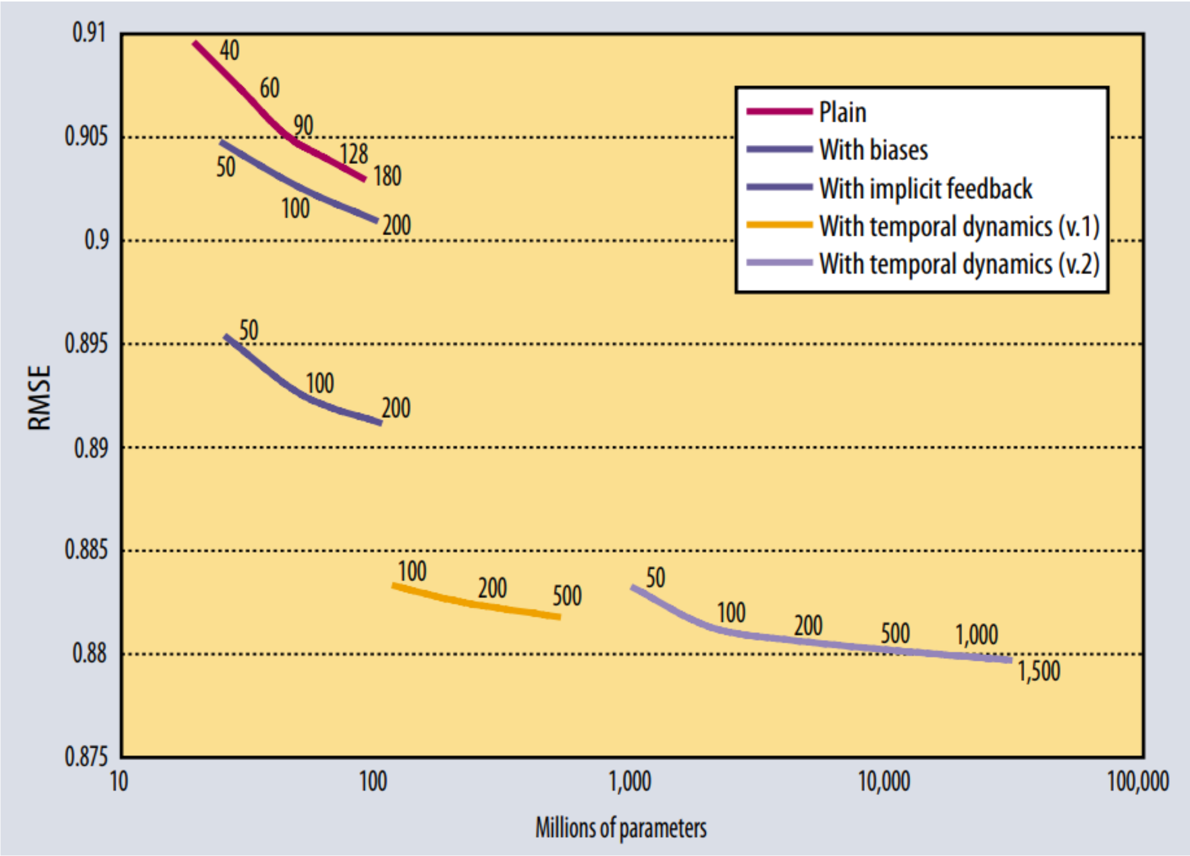


https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

* not in syllabus

Netflix Results

Wining team *Bellkor's Pragmatic Chaos* essentially used a matrix factorization model with additional parameters for biases, user attributes and time dependencies.



https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

Summary

- Matrix Completion
  - Recommender Systems
  - Collaborative Filtering
- k-Nearest Neighbors
  - Correlation Coefficient
  - Weighted Prediction
- Matrix Factorization
  - Subspace Learning
  - Low-rank Approximation
  - Regularized Training Loss
  - Alternating Least Squares

Intended Learning Outcomes

Matrix Completion

- Recognize that the key problem in collaborative filtering is matrix completion. Give examples of collaborative filtering.
- Explain how dimensionality reduction helps matrix completion.

k-Nearest Neighbors

- Compute the user similarity using the correlation coefficient.
- Compute the k-nearest neighbors ranked by user similarity.
- Predict an unknown rating using weighted prediction.

Intended Learning Outcomes

Matrix Factorization

- Describe the relationship between subspace learning, low-rank approximation and matrix factorization.
- Write down the regularized training loss. Explain why regularization is necessary for matrix factorization.
- Describe the alternating least squares algorithm, and explain why it minimizes the training loss. Apply the algorithm in a recommendation problem.
- Describe how validation can be used to select suitable hyperparameters k and λ for good generalization.