Advanced Topics

Components of Learning

- Suppose that a bank wants to automate the process of evaluating credit card applications.
  - Input x (customer information that is used to make a credit application).
  - Target function $f: X \to Y$ (ideal formula for credit approval), where X and Y are the input and output space, respectively.
  - Dataset D of input-output examples $(x_1, y_1), \ldots, (x_n, y_n)$.
  - Hypothesis (skill) with hopefully good performance:

$$g: X \to Y \text{ (“learned” formula to be used)}$$

Recall the Perceptron

For $\mathbf{x} = x_1, \ldots, x_d$ ("features of the customer"), compute a weighted score and:

$$\text{Approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold},$$

$$\text{Deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}.$$

Perceptron: A Mathematical Description

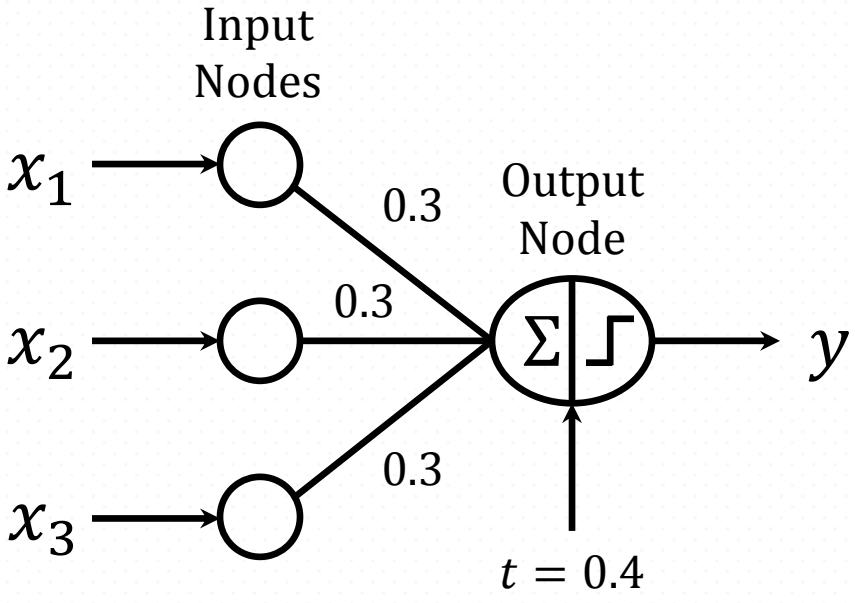
This formula can be written more compactly as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right),$$

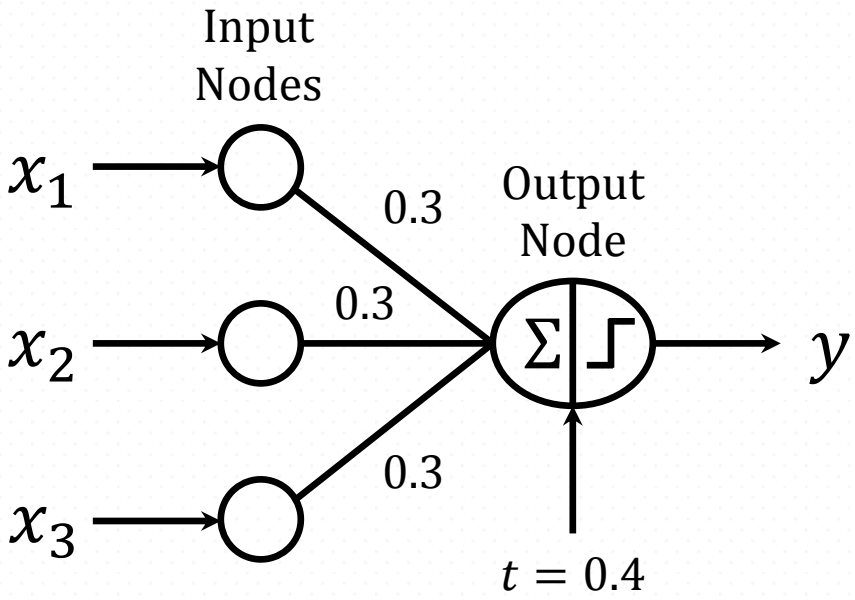
where $h(\mathbf{x}) = +1$ means 'approve credit' and $h(\mathbf{x}) = -1$ means 'deny credit'; $\text{sign}(s) = +1$ if $s > 0$ and $\text{sign}(s) = -1$ if $s < 0$. This model is called a *perceptron*.

Perceptron: A Visual Description

x_1	x_2	x_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Perceptron: A Visual Description



Perceptron Learning Process

The key computation for this algorithm is the weight update formula:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda \left(y_i - \hat{y}_i^{(k)} \right) x_{ij},$$

where $w^{(k)}$ is the weight parameters associated with the ith input linke after the kth iteration, λ is a parameter known as the learning rate, and x_{ij} is the value of the jth feature of the training example x_i.

Perceptron Learning Process

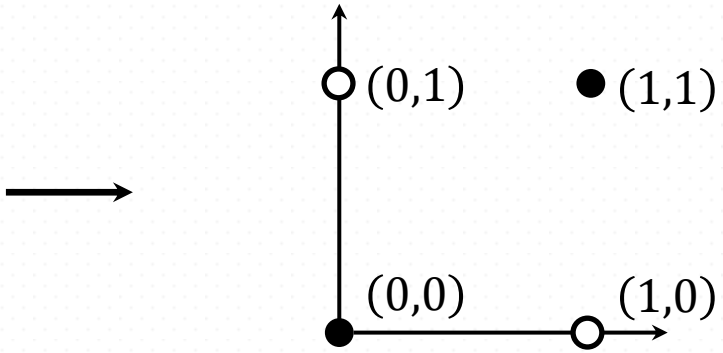
$$w_j^{(k+1)} = w_j^{(k)} + \lambda \left(y_i - \hat{y}_i^{(k)} \right) x_{ij},$$

1. If $y = +1$ and $\hat{y} = -1$, then the prediction error is $(y_i - \hat{y}_i) = 2$. To compensate for the error, we need to increase the value of the predicted output.
2. If $y = -1$ and $\hat{y} = +1$, then $(y_i - \hat{y}_i) = -2$. To compensate for the error, we need to decrease the value of the predicted output.

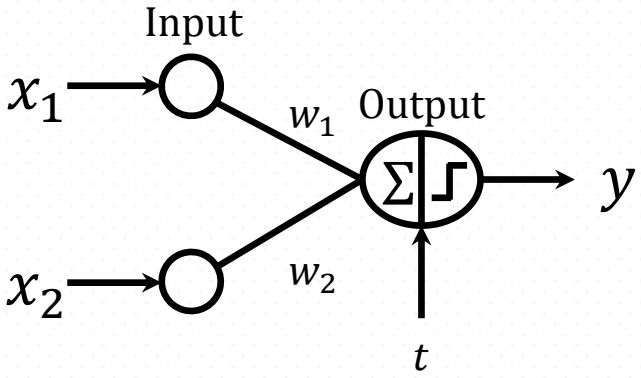
Perceptron Limitations

Data

x_1	x_2	$y = x_1$ XOR x_2
1	1	0
1	0	1
0	1	1
0	0	0



Model



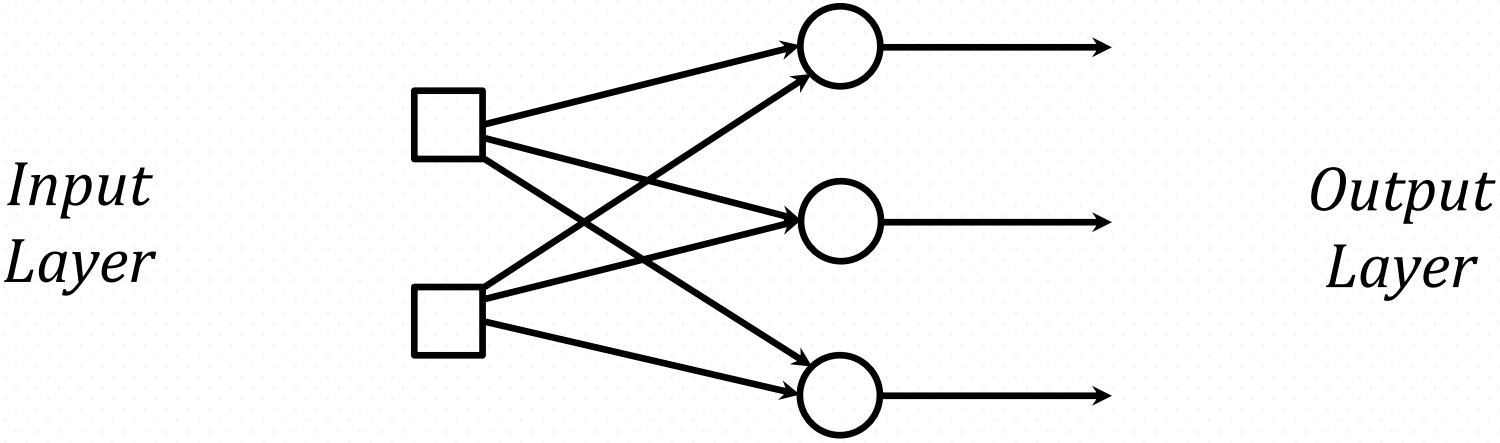
The following cannot all be true:

$$w_1 \times 1 + w_2 \times 1 < t$$
$$w_1 \times 1 + w_2 \times 0 > t$$
$$w_1 \times 0 + w_2 \times 1 > t$$
$$w_1 \times 0 + w_2 \times 0 < t$$

Network Architectures

- Three different network architectures:
  1. Single-layer feed-forward
  2. Multi-layer feed-forward
  3. Recurrent
- The architecture of a neural network is linked with the learning algorithm used to train.

Single-Layer Feed-Forward (FF) NN



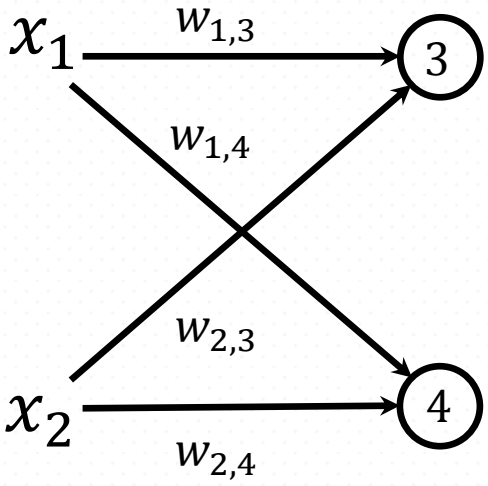
Multilayer Artificial Neural Network

An artificial neural network (ANN) has a more complex structure than that of a perceptron model. The additional complexities may arise in a number of ways:

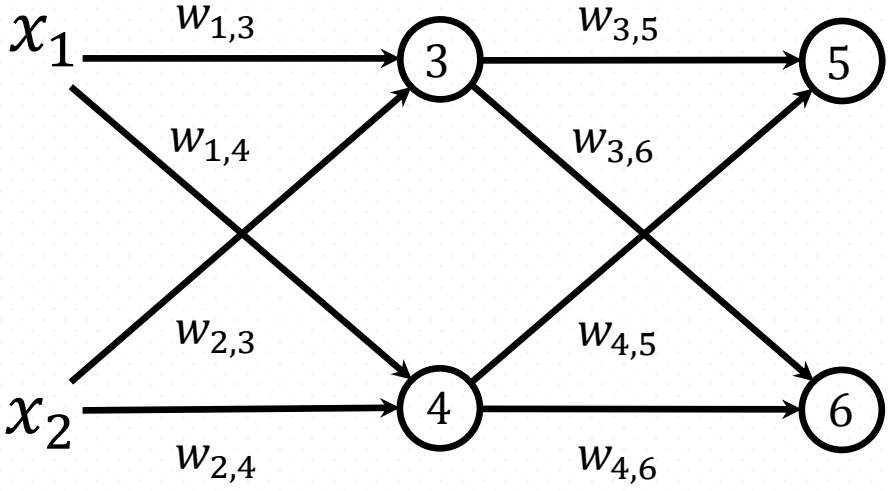
1. The network may contain several intermediary layers between its input and output layers.
2. The network may use types of activation functions other than the sign function.

Multilayer Perceptrons

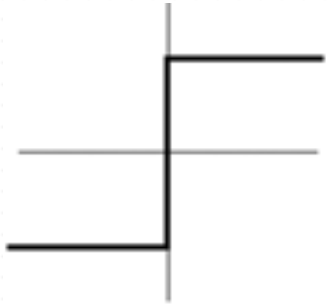
Single Layer



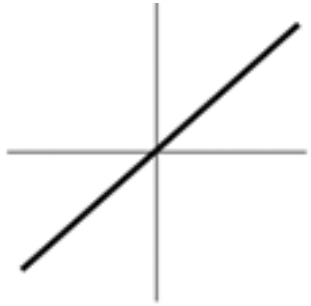
Multiple Layers



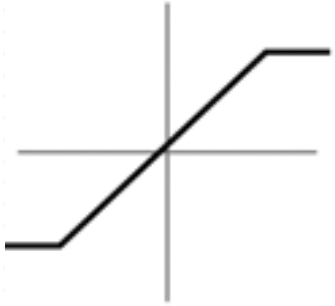
Activation Functions



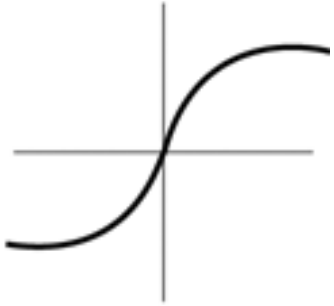
Step Function



Linear Function



Threshold Logic



Sigmoid Function

ANN Learning Process

To learn the weights of an ANN model, we need an efficient algorithm that converges to the right solution when a sufficient amount of training data is provided.

ANN Learning Process

One approach is to treat each hidden node or output node in the network as an independent perceptron unit and to apply the perceptron weight update formula.

Problem: We lack a priori knowledge about the true outputs of the hidden nodes.

Gradient Descent

The goal of the ANN learning algorithm is to determine a set of weights **w** that minimize the total sum of squared errors:

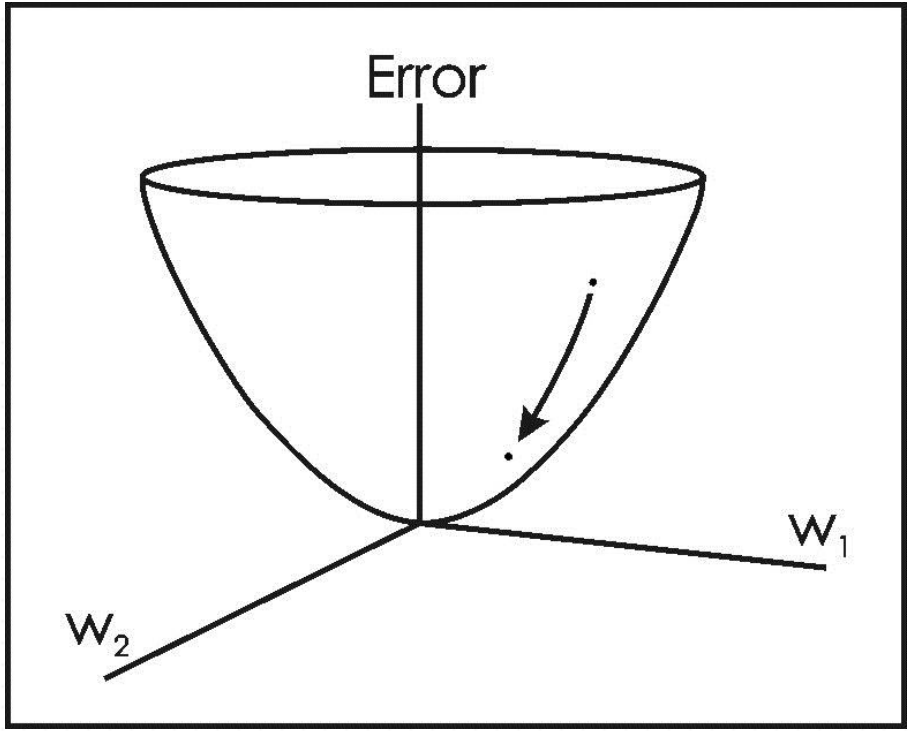
$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

The weight update formula used by gradient descent is:

$$w_j \leftarrow w_j - \lambda \frac{\partial E(\mathbf{w})}{\partial w_j}, \text{where } \lambda \text{ is the learning rate.}$$

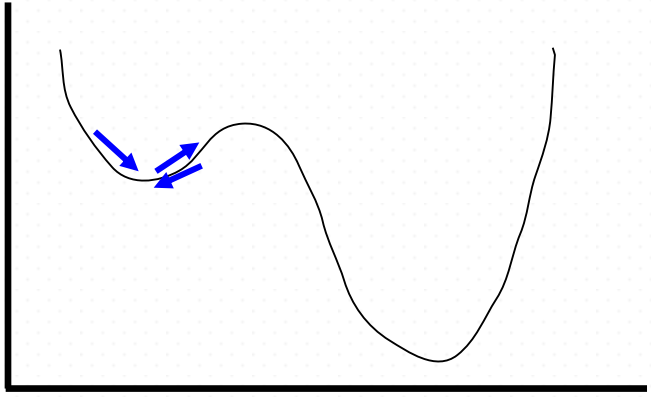
Gradient Descent

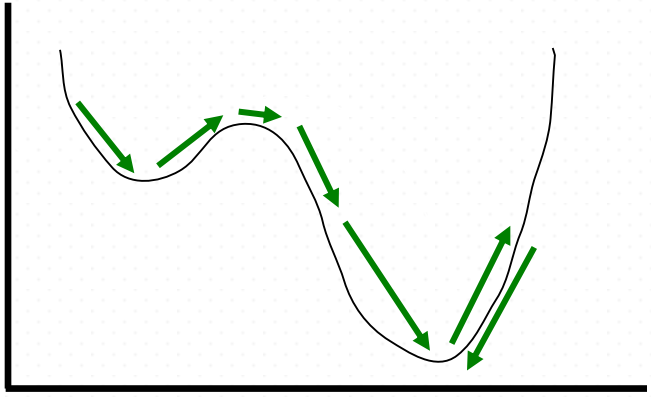
Error Surface for a Two-Parameter Model



Enhancements to Gradient Descent

Momentum: Adds a percentage of the last movement to the current movement.





Gradient Descent

Key Idea:

Intuitively, we increase the weight in a direction that reduces the overall error term. The greater the decrease in error, the greater the increase in weight.

Gradient Descent

Key Idea:

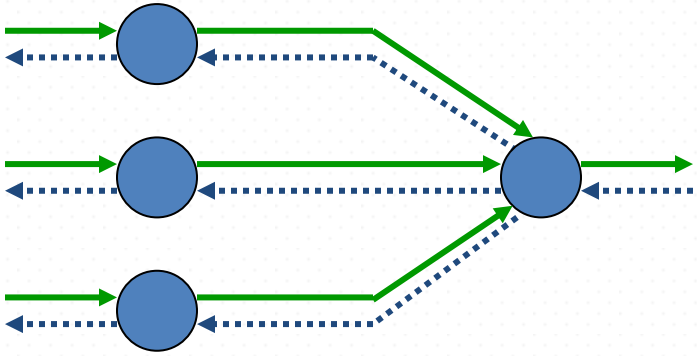
Intuitively, we increase the weight in a direction that reduces the overall error term. The greater the decrease in error, the greater the increase in weight.

Problem:

This only applies to learning the weights of the output and hidden nodes of a neural network.

Backpropagation: An Overview

- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.



Network activation
Forward Step

Error propagation
Backward Step

Backpropagation Defined

- Developed to address the problem of computing the error for the hidden nodes, since their output values are not known *a priori*.
- After the weights have been computed for each layer (activation), the weight update formula is applied in the reverse direction.
- Backpropagation allows us to use the errors for neurons at layer $k + 1$ to estimate the errors for neurons at layer k.

Training with Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron (minimizes the total error).
- Backpropagation consists of the repeated application of the following two passes.
  - **Forward pass**: The network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: The network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

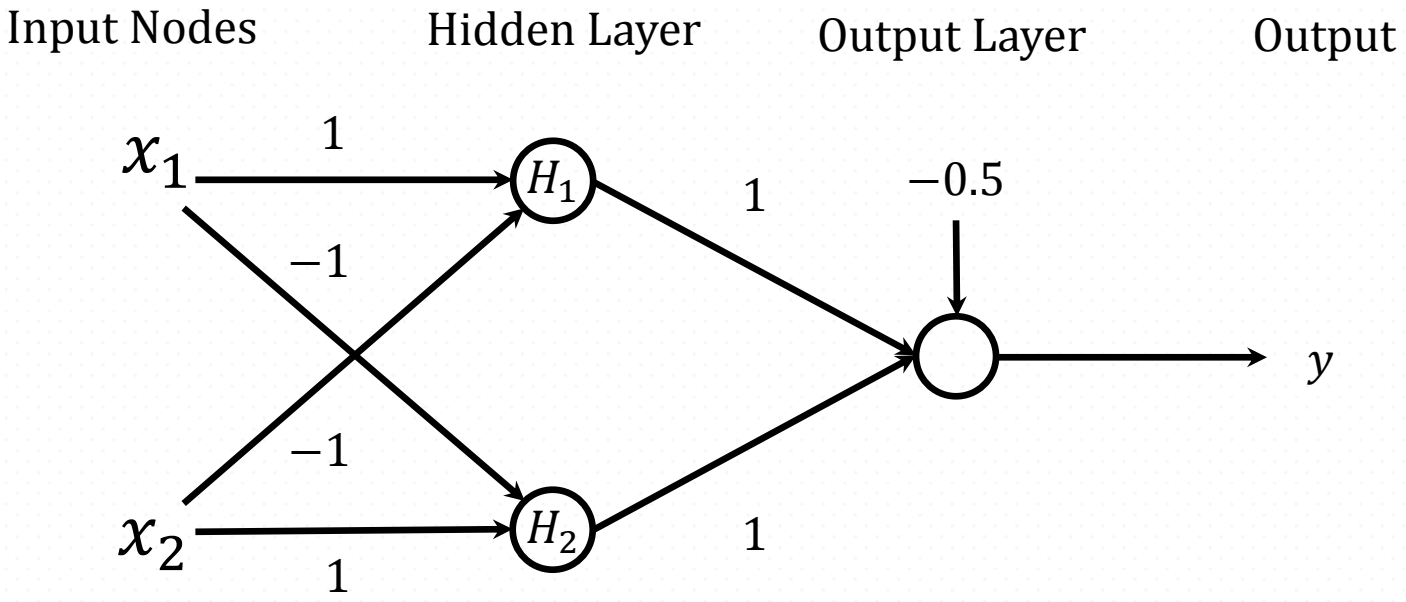
Stopping Criteria

- Total mean squared error change:
  - Backpropagation is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for testing the network generalization will not used for updating the weights.

Feed-Forward Neural Network for XOR

- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors (1,-1) and (-1,1).
- The output node is used to combine the outputs of the two hidden nodes.

Feed-Forward Neural Network for XOR



Feed-Forward Neural Network for XOR

Inputs		**Output of Hidden Nodes**		**Output Node**	**X_1 XOR X_2**
X_1	X_2	H_1	H_2		
0	0	0	0	–0.5 → 0	0
0	1	–1 → 0	1	0.5 → 1	1
1	0	1	–1 → 0	0.5 → 1	1
1	1	0	0	–0.5 → 0	0

Since we are representing two states by 0 (false) and 1 (true), we can map negative outputs (–1, –0.5) of hidden and output layers to 0 and positive output (0.5) to 1.

Neural Network Topology

- The number of layers and neurons depend on the specific task.
- In practice this issue is solved by trial and error.
- Two types of adaptive algorithms can be used:
  - Begin with a large network and successively remove some neurons and links until network performance degrades.
  - Begin with a small network and introduce new neurons until performance is satisfactory.

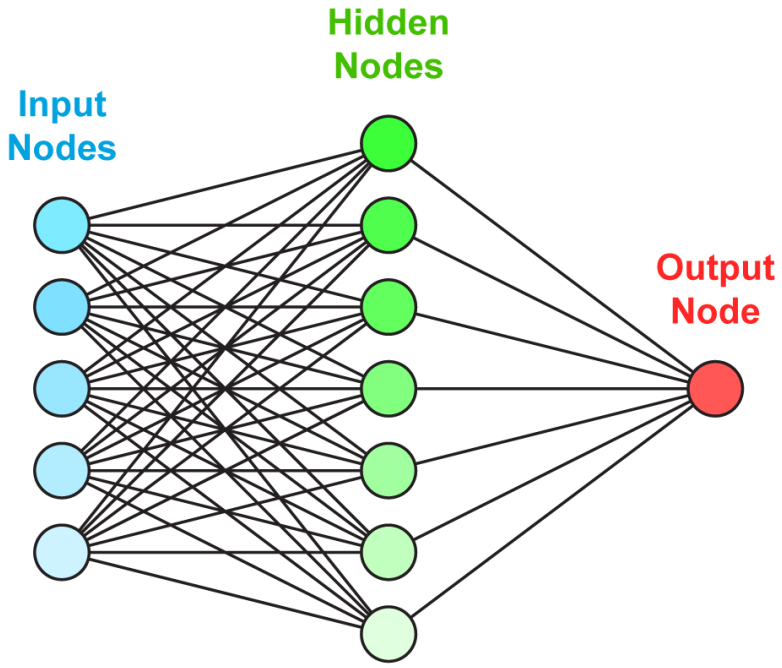
Neural Network Design Issues

- Data representation
- Network Topology
- Network Parameters
- Training
- Validation

Advanced Topics

Last class

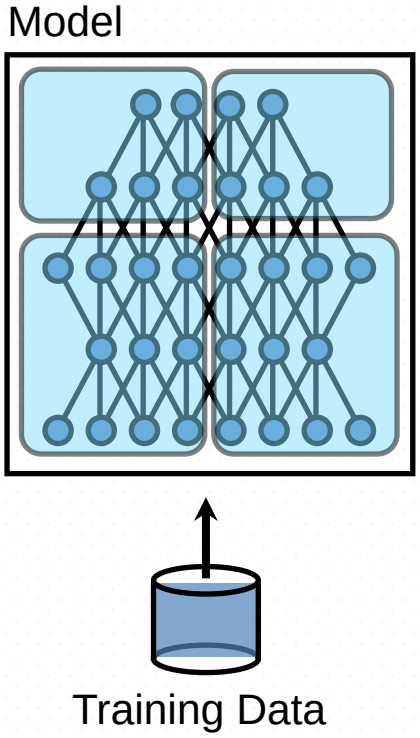
- Neural Networks



Popularity and Applications

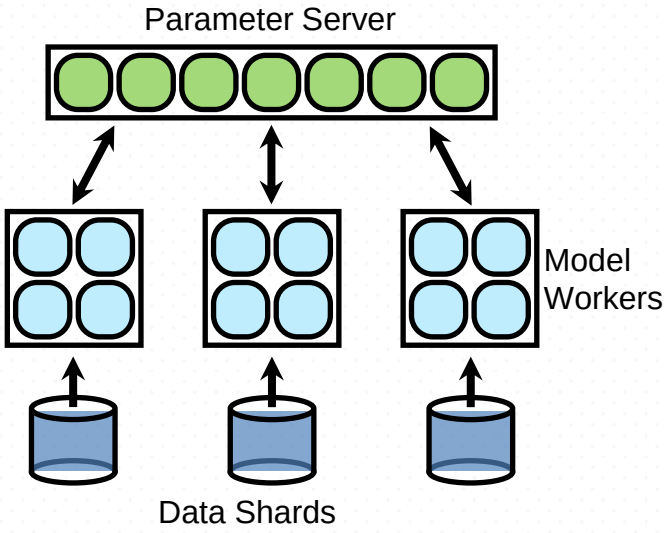
- Major increase in popularity of Neural Networks
- Google developed a couple of efficient methods that allow for the training of huge deep NN
  - Asynchronous Distributed Gradient Descent
  - L-BFGS
- These have recently been made available to the public

Large scale ANN

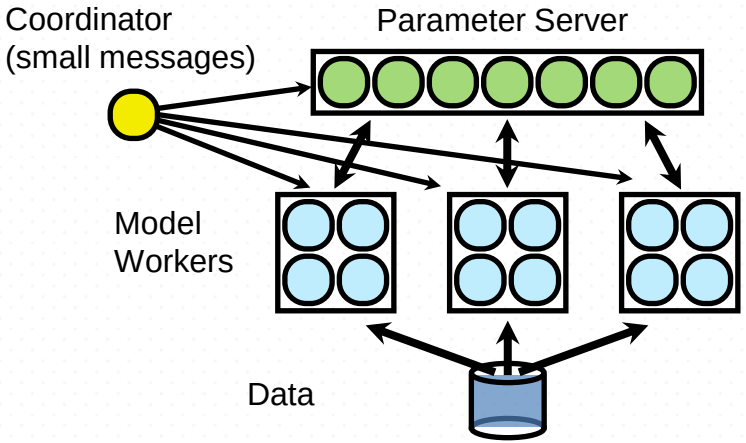


Popularity and Applications

Asynchronous Distributed Gradient Descent



L-BFGS



- 20,000 cores in a single cluster
- up to 1 billion data items / mega-batch (in ~1 hour)

Neural Networks

Images

CIFAR Object classification	Accuracy
Prior art (Ciresan et al., 2011)	80.5%
Stanford Feature learning	**82.0%**

NORB Object classification	Accuracy
Prior art (Scherer et al., 2010)	94.4%
Stanford Feature learning	**95.0%**

Video

Hollywood2 Classification	Accuracy
Prior art (Laptev et al., 2004)	48%
Stanford Feature learning	**53%**

KTH	Accuracy
Prior art (Wang et al., 2010)	92.1%
Stanford Feature learning	**93.9%**

YouTube	Accuracy
Prior art (Liu et al., 2009)	71.2%
Stanford Feature learning	**75.8%**

UCF	Accuracy
Prior art (Wang et al., 2010)	85.6%
Stanford Feature learning	**86.5%**

Text/NLP

Paraphrase detection	Accuracy
Prior art (Das & Smith, 2009)	76.1%
Stanford Feature learning	**76.4%**

Sentiment (MR/MPQA data)	Accuracy
Prior art (Nakagawa et al., 2010)	77.3%
Stanford Feature learning	**77.7%**

Multimodal (audio/video)

AVLetters Lip reading	Accuracy
Prior art (Zhao et al., 2009)	58.9%
Stanford Feature learning	**65.8%**

Popularity and Applications

AUG 6

Speech Recognition and Deep Learning

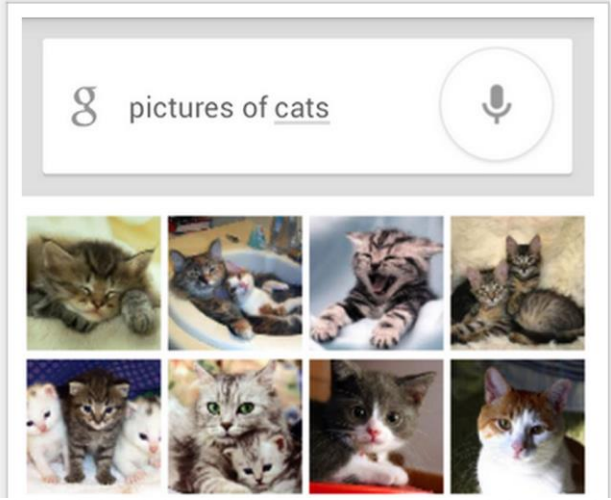
Posted by Vincent Vanhoucke, Research Scientist, Speech Team

The New York Times recently published an article about Google's large scale deep learning project, which learns to discover patterns in large datasets, including... cats on YouTube!

What's the point of building a gigantic cat detector you might ask? When you combine large amounts of data, large-scale distributed computing and powerful machine learning algorithms, you can apply the technology to address a large variety of practical problems.

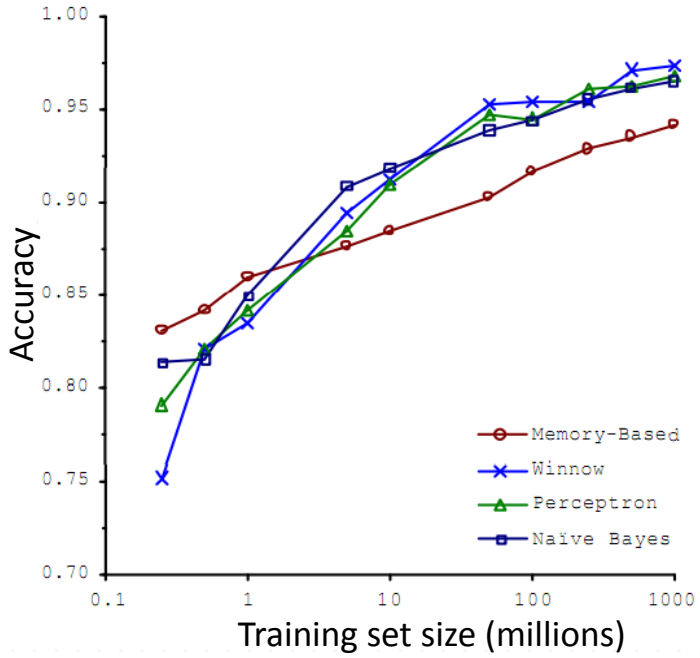
With the launch of the latest Android platform release, Jelly Bean, we've taken a significant step towards making that technology useful: when you speak to your Android phone, chances are, you are talking to a neural network trained to recognize your speech.

Using neural networks for speech recognition is nothing new: the first proofs of concept were developed in the late



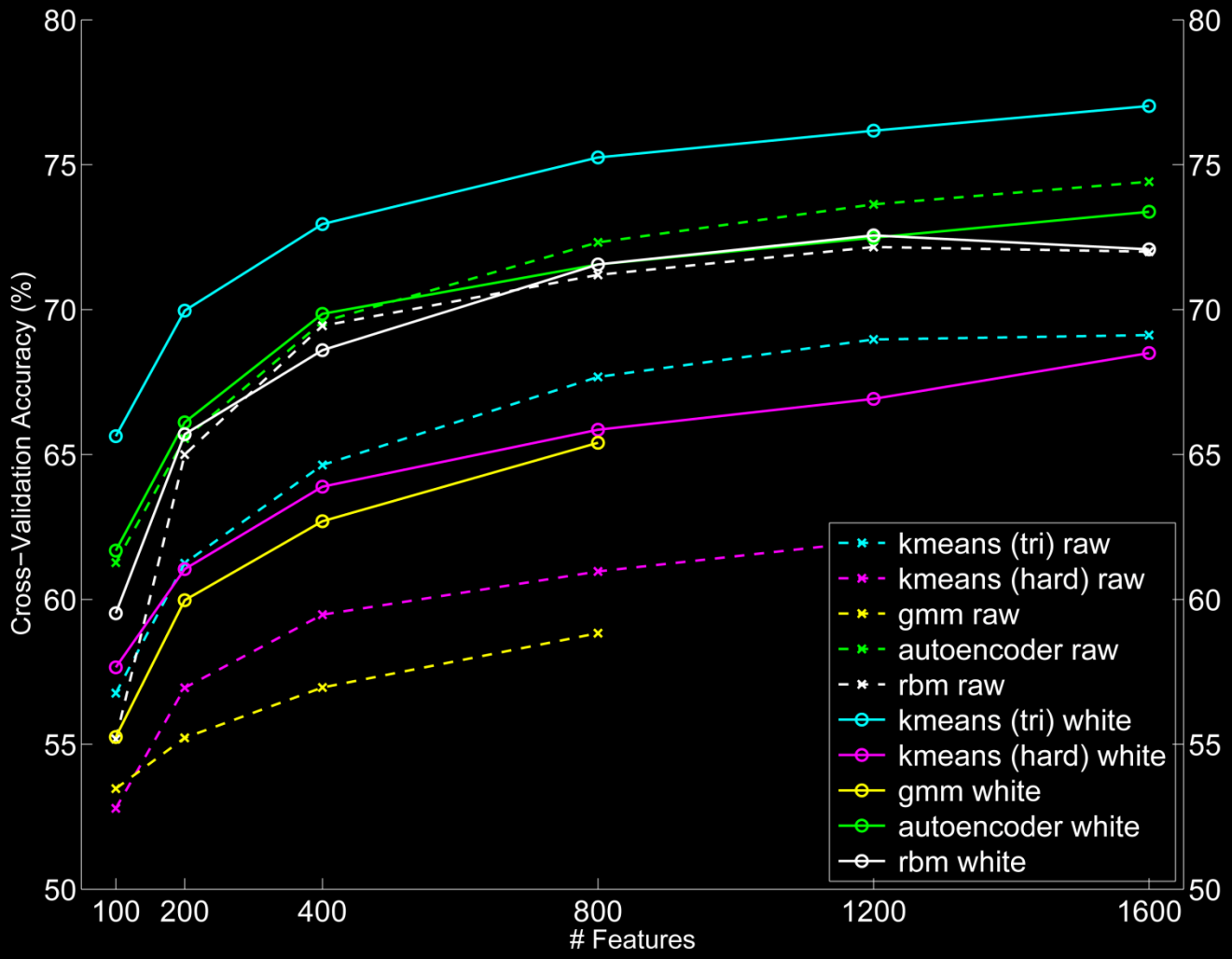
Popularity and Applications

Learning from Unlabeled Data



[Banko & Brill, 2001]

"It's not who has the best algorithm that wins. It's who has the most data."


Cross−Validation Accuracy (%)
Features
80
75
70
65
60
55
50
100
200
400
800
1200
1600
kmeans (tri) raw
kmeans (hard) raw
gmm raw
autoencoder raw
rbm raw
kmeans (tri) white
kmeans (hard) white
gmm white
autoencoder white
rbm white

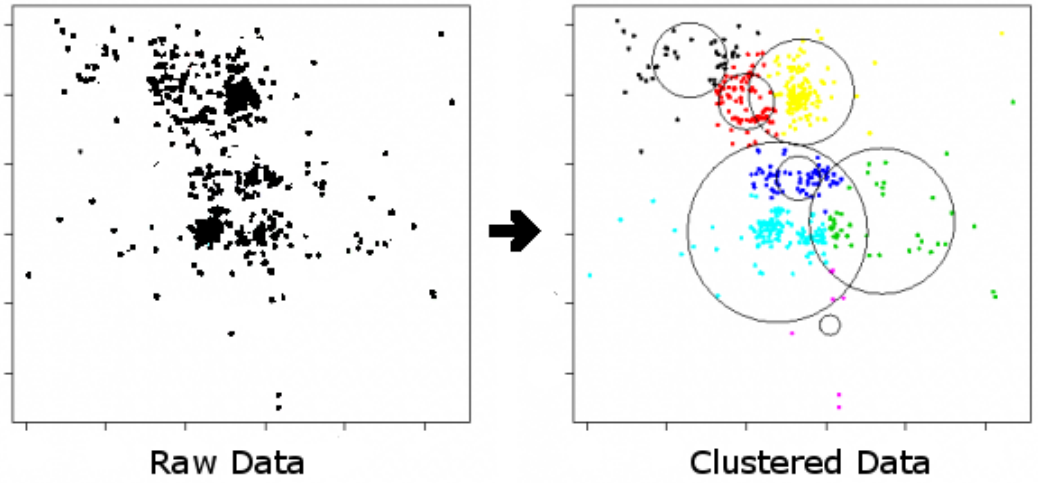
Feature Learning

A set of techniques in machine learning that learn a transformation of "raw" inputs to a representation that can be effectively exploited in a supervised learning task such as classification

Feature Learning

- Can be supervised or unsupervised
- Example of algorithms:
  - Autoencoders
  - Matrix Factorization
  - Restricted Boltzmann machine
  - Clustering
  - Neural Networks

Feature Learning



- Each k-centroid can be used to represent a feature for supervised learning
- Each feature j has value 1 iff the j^{th} centroid learned by k-means is the closest to the instance under consideration

Deep Learning

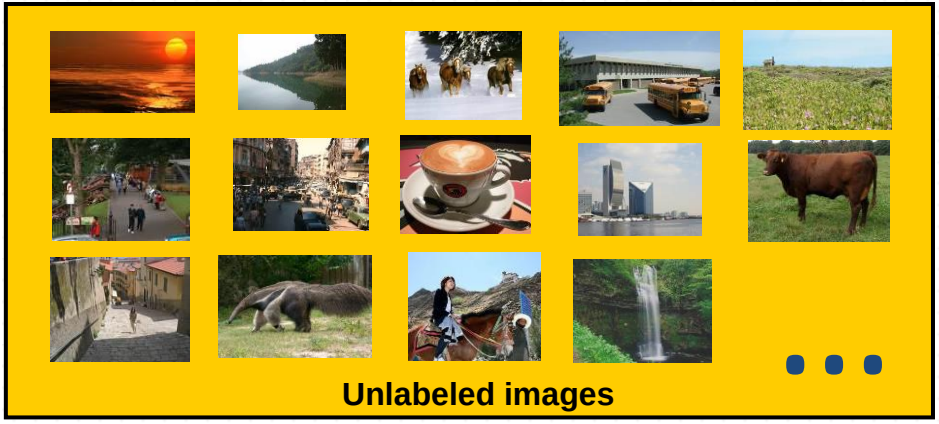
Andrew Ng

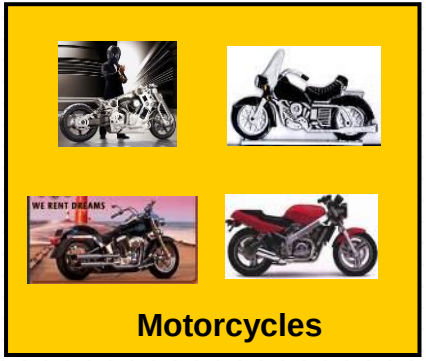
- Machine learning algorithms to model high-level data abstractions using a series of transformations
- Based on feature learning
- The motivation: Some data representations make it easier to learn particular tasks (e.g., image classification)

- The motivation: Some data representations make it easier to learn particular tasks (e.g., image classification)

- Ex.: Our assignment 2:
  - It's hard to give your computer an image and ask "what season does it represent?" but if we can apply transformations that describe similar images with a simpler representation, we might accomplish that task

Self-Taught Learning (Unsupervised Feature Learning)



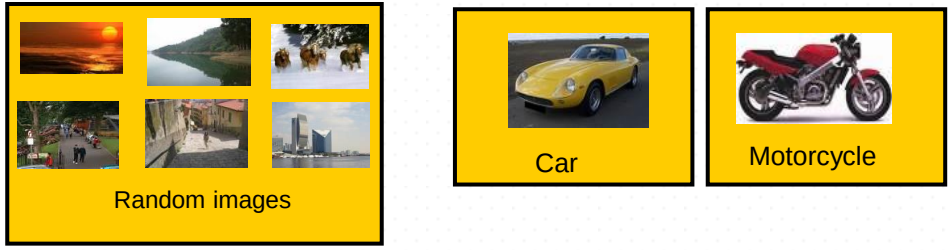


Testing:
What is this?

- Supervised learning

- Semi-supervised learning

- Self-taught learning (unsupervised feature learning)



Self-Taught Learning

- One can always try to get more labeled data, but this can be expensive
  - Amazon Mechanical Turk
  - Expert feature engineering
- The promise of **self-taught learning** and **unsupervised feature learning:**
  - If we can get our algorithms to learn from *unlabeled* data, then we can easily obtain and learn from massive amounts of it
- 1 instance of unlabeled data < 1 instance of labeled data
- Billions of instances of unlabeled data >> some labeled data

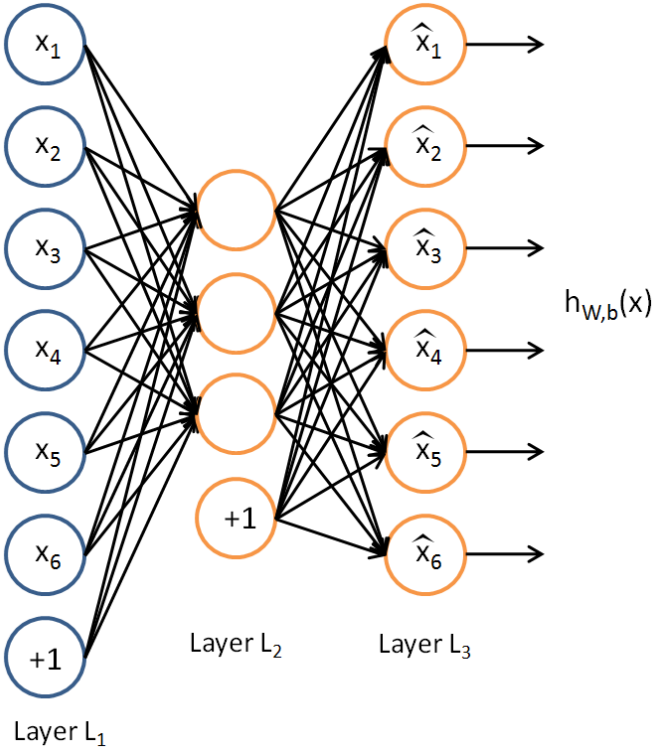
Self-Taught Learning

- The idea:
  - Give the algorithm a large amount of unlabeled data
  - The algorithm learns a feature representation of that data
  - If the end goal is to perform classification, one can find a small set of labeled instances to probe the model and adapt it to the supervised task

Autoencoders

- An artificial neural network used for learning efficient codings
- Can be used for dimensionality reduction
- Similar to a Multilayer Perceptron with an input layer and one or more hidden layers
- The difference between autoencoders and MLP:
  - Autoencoders have the same number of inputs and outputs
  - Instead of predict *y,* autoencoders try to reconstruct *x*

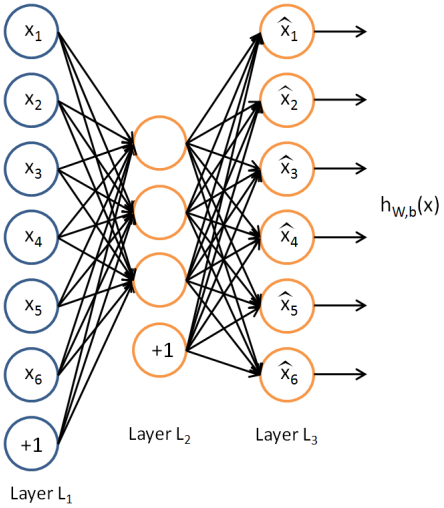
Autoencoders



If the hidden layers are narrower than input layer, then the activations of the final layers can be regarded as compressed representation of the input

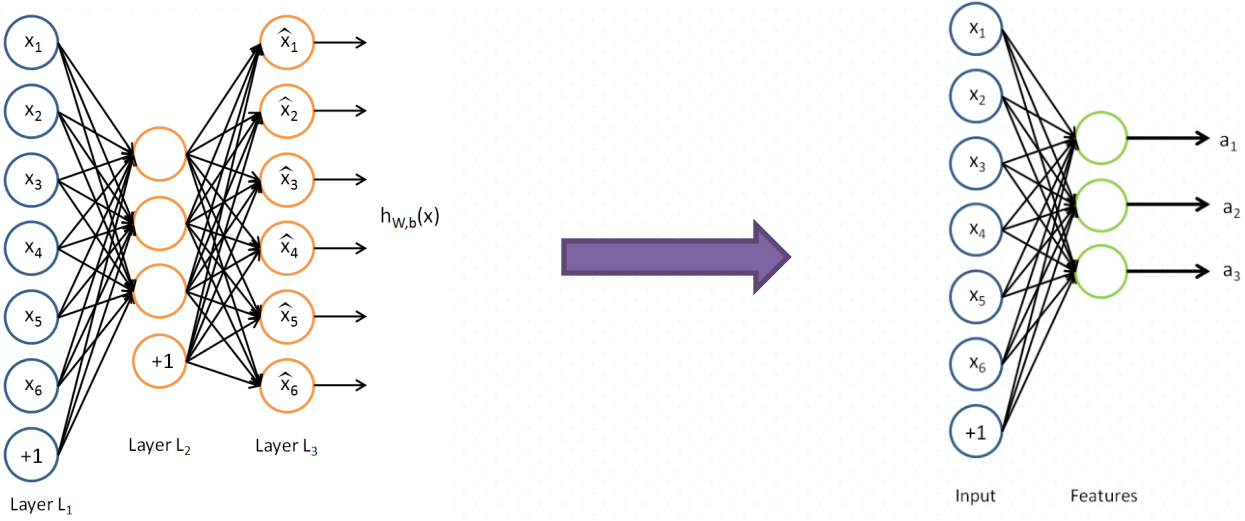
Self-Taught Learning in Practice

- Suppose we have an unlabeled training set $\{x_u^{(1)}, x_u^{(2)}, \ldots, x_u^{(m_u)}\}$ with m_u unlabeled instances
- Step 1: Train an autoencoder on this data



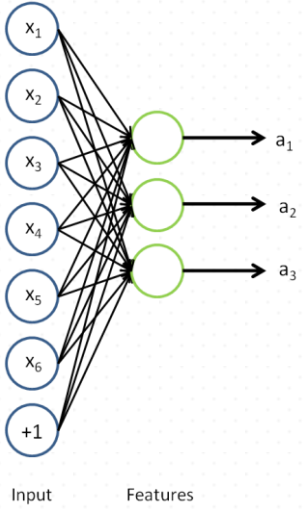
Self-Taught Learning in Practice

- After step 1, we will have learned all weight parameters for the network
- We can also visualize the algorithm for computing the features/activations a s the following neural network:



Self-Taught Learning in Practice

- Step 2:
  - Training set: $\left\{\left(x_l^{(1)}, y^{(1)}\right), \left(x_l^{(2)}, y^{(2)}\right), \ldots, \left(x_l^{(m_l)}, y^{(m_l)}\right)\right\}$ of m_l labeled examples
  - Feed training example $x_l^{(1)}$ to the autoencoder and obtain its corresponding vector of activations $a_l^{(1)}$
  - Repeat that for all training examples

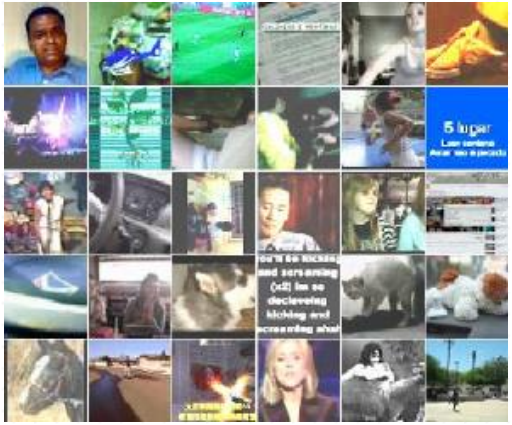


Self-Taught Learning in Practice

- Step 3:
  - Replace the original feature with $a_l^{(1)}$
  - The training set then becomes:
    - $\left\{\left(a_l^{(1)}, y^{(1)}\right), \left(a_l^{(2)}, y^{(2)}\right), \ldots, (a_l^{(m_l)}, y^{(m_l)})\right\}$
- Step 4:
  - Train a supervised algorithm using this new training set to obtain a function that makes predictions on the y values

An Example of Self-Taught Learning

- What Google did:
  - Trained on 10 million images (YouTube)
  - 1000 machines (16,000 cores) for 1 week.
  - 1.15 billion parameters
  - Test on novel images



Training set (YouTube)

Test set (FITW + ImageNet)

Result Highlights

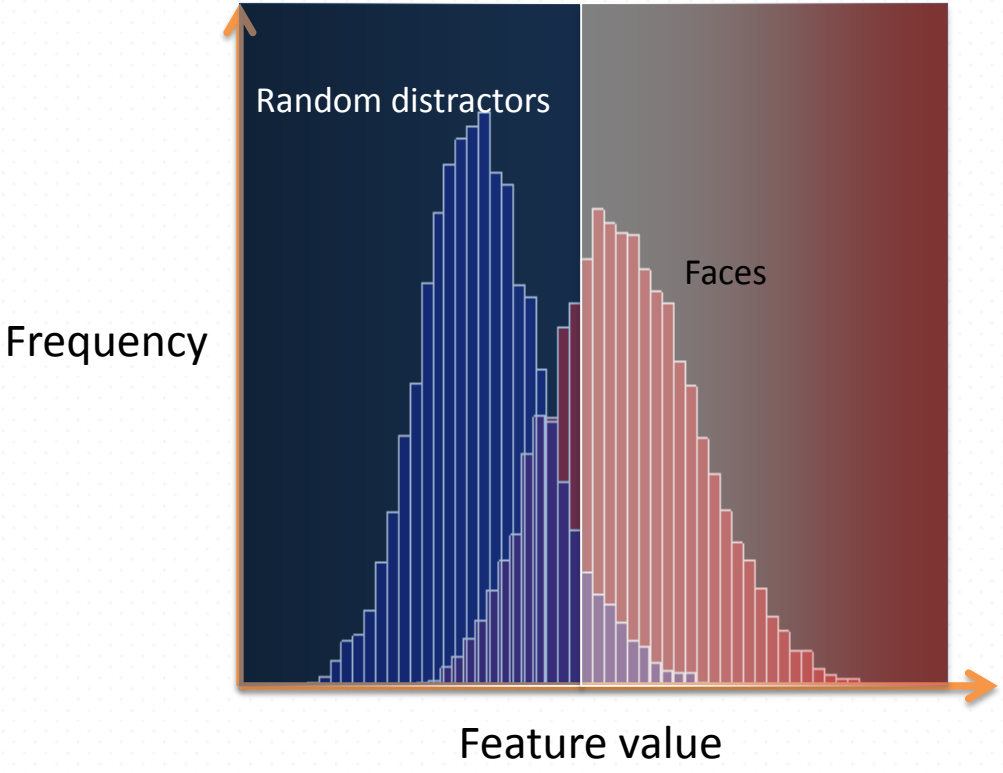
- The face neuron

Top stimuli from the test set

Optimal stimulus
by numerical optimization

Result Highlights

- The face neuron



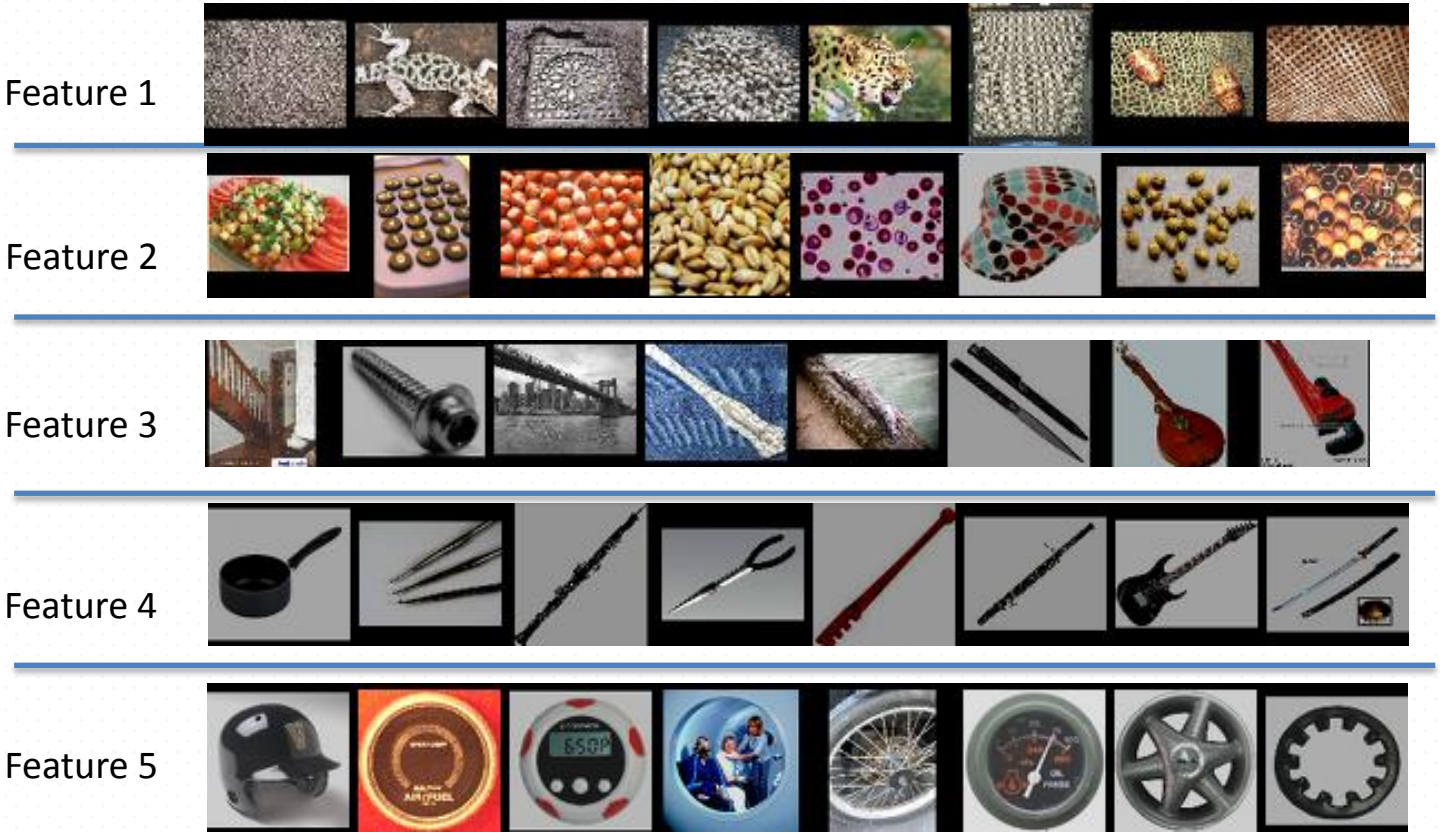
Result Highlights

- The cat neuron

Optimal stimulus
by numerical optimization

Result Highlights

Best stimuli



Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012

Best stimuli

Feature 6

Feature 7

Feature 8

Feature 9

Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012

Best stimuli

Feature 10

Feature 11

Feature 12

Feature 13

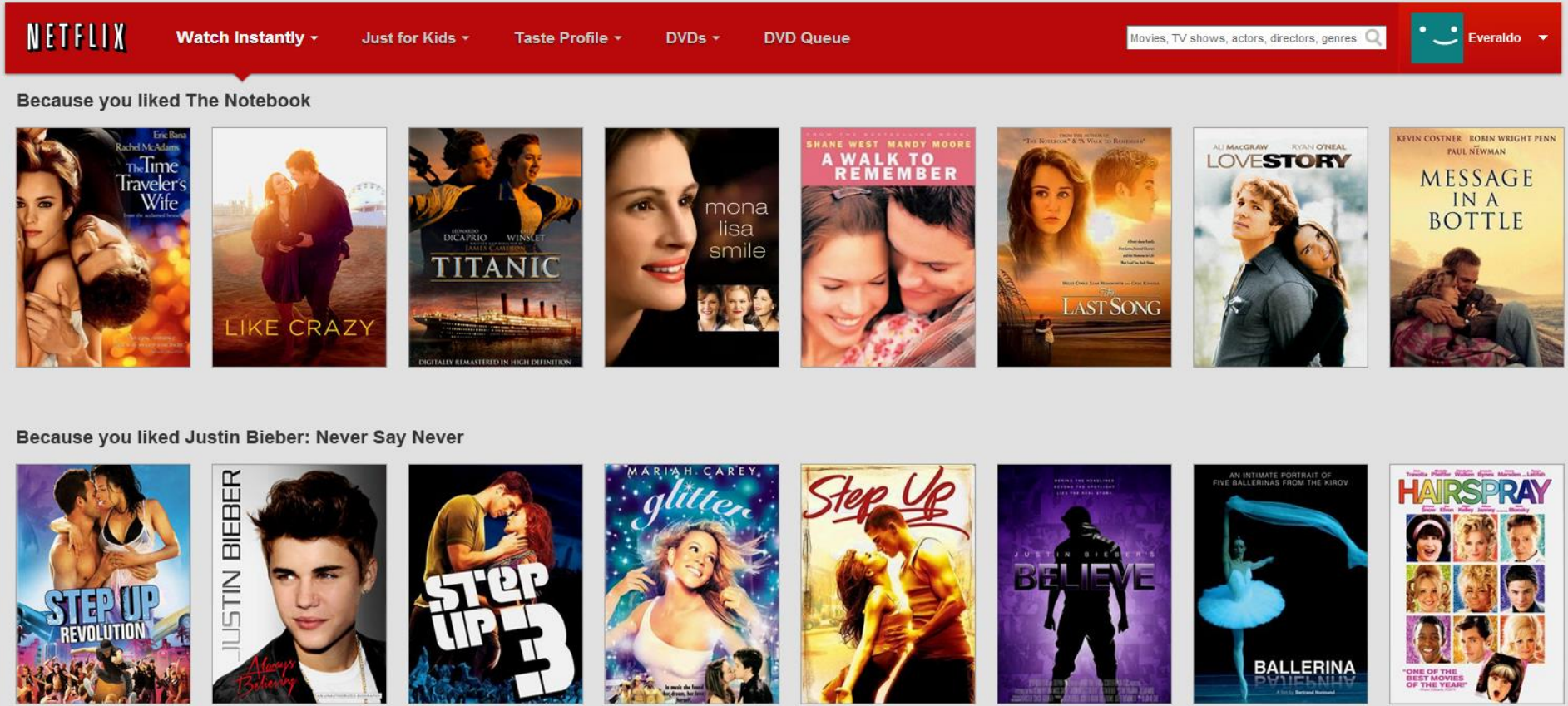
Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012

Advanced Recommendations with Collaborative Filtering

Remember Recommendations?

Let's review the basics.

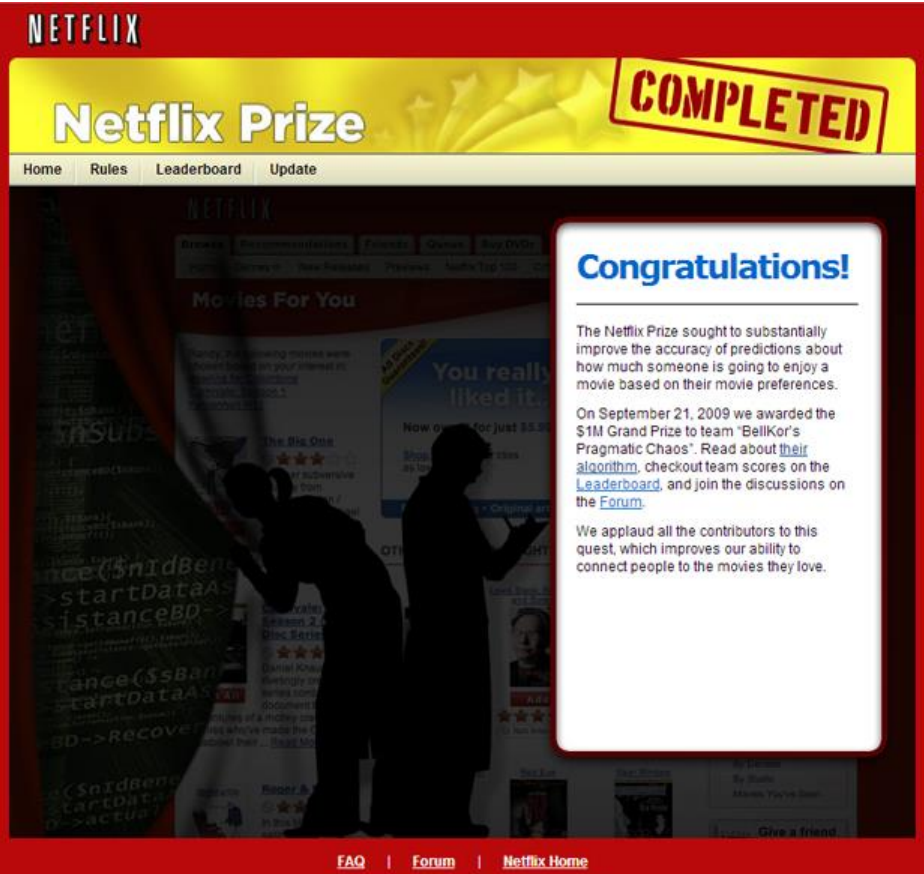
Recommendations



Recommendations are Everywhere

The Netflix Prize (2006-2009)

The Netflix Prize (2006-2009)



What was the Netflix Prize?

- In October, 2006 Netflix released a dataset containing 100 million anonymous movie ratings and challenged the data mining, machine learning, and computer science communities to develop systems that could beat the accuracy of its recommendation system, Cinematch.
- Thus began the Netflix Prize, an open competition for the best collaborative filtering algorithm to predict user ratings for films, solely based on previous ratings without any other information about the users or films.

The Netflix Prize Datasets

- Netflix provided a *training* dataset of 100,480,507 ratings that 480,189 users gave to 17,770 movies.
  - Each training rating (or instance) is of the form ⟨user, movie, data of rating, rating⟩.
  - The user and movie fields are integer IDs, while ratings are from 1 to 5 (integral) stars.

The Netflix Prize Datasets

- The *qualifying* dataset contained over 2,817,131 instances of the form ⟨user, movie, date of rating⟩, with ratings known only to the jury.
- A participating team's algorithm had to predict grades on the entire qualifying set, consisting of a *validation* and *test* set.
  - During the competition, teams were only informed of the score for a *validation* or *quiz* set of 1,408,342 ratings.
  - The jury used a *test* set of 1,408,789 ratings to determine potential prize winners.

The Netflix Prize Data

Movie Ratings

Users	1	2	. .	*m*
1	5	2	5	4
2	2	5		3
⋮	2	2	4	2
n	5	1	5	*?*

The Netflix Prize Data

Movie Ratings

	1	2	. .	m
1	5	2	5	4
2	2	5		3
⋮	2	2	4	2
n	5	1	5	*?*

Instances (samples, examples, observations)

The Netflix Prize Data

Features (attributes, dimensions)

Users	1	2	. .	*m*
1	5	2	5	4
2	2	5		3
⋮	2	2	4	2
n	5	1	5	*?*

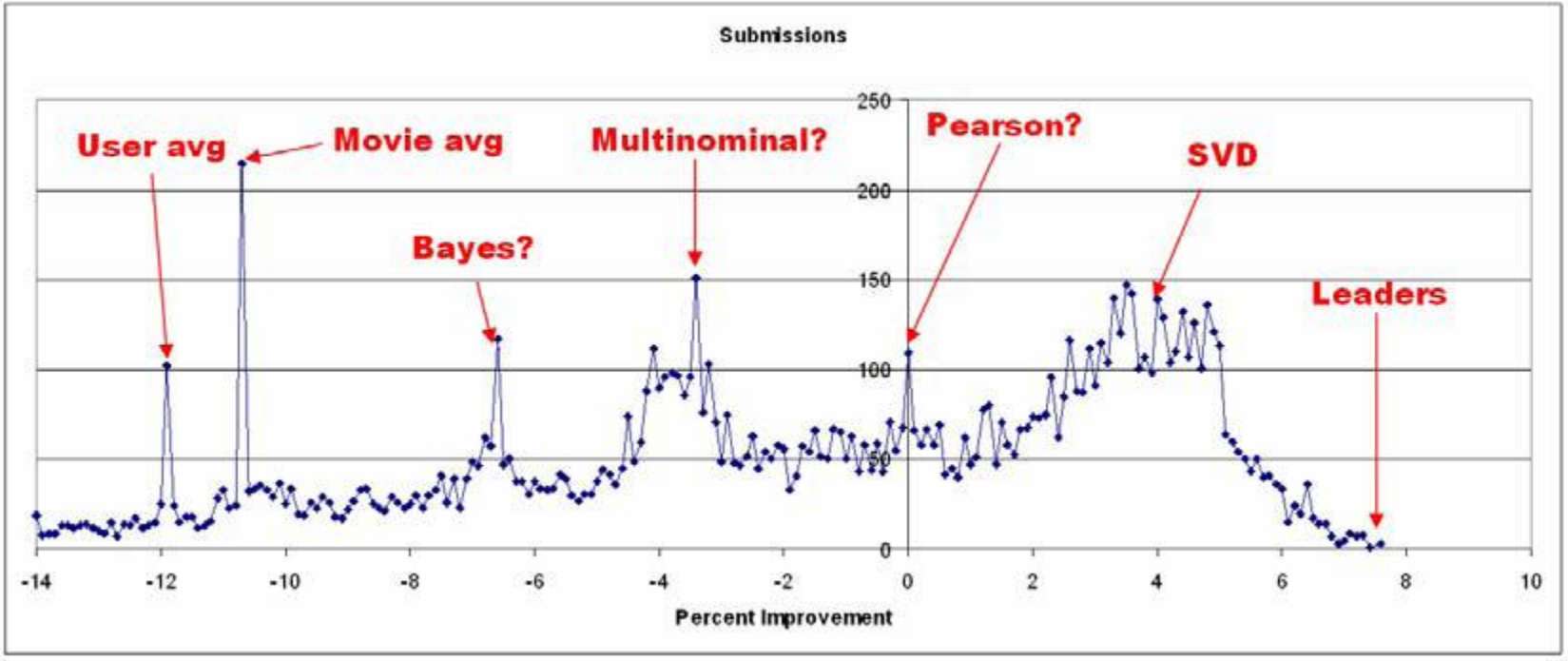
The Netflix Prize Goal

Movie Ratings

		Star Wars	Hoop Dreams	Contact	Titanic
Users	Joe	5	2	5	4
	John	2	5		3
	Al	2	2	4	2
	Everaldo	5	1	5	*?*

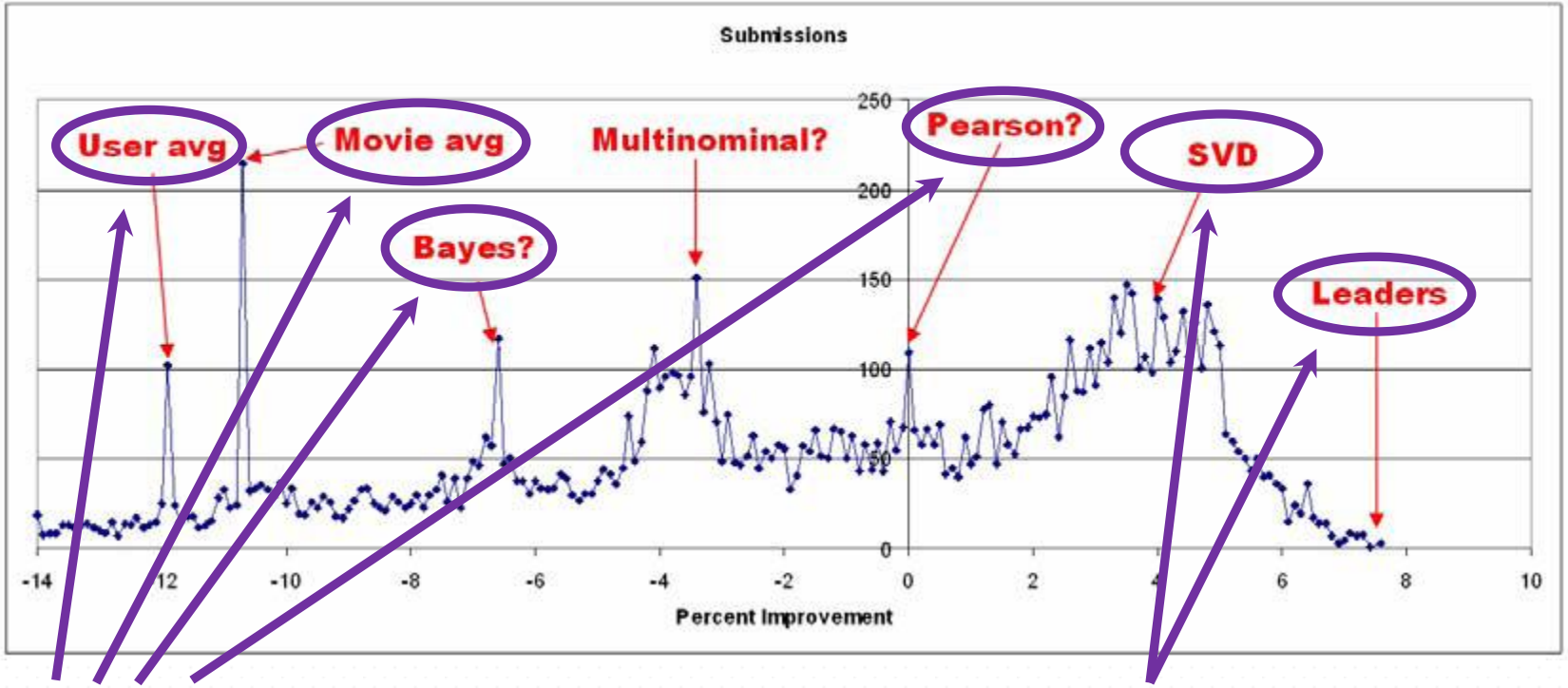
Goal: Predict *?* (a movie rating) for a user

The Netflix Prize Methods



Bennett, James, and Stan Lanning. "The Netflix Prize." *Proceedings of KDD Cup and Workshop*. Vol. 2007. 2007.

The Netflix Prize Methods



We discussed these methods.

We will discuss these methods now.

All of these methods are based upon collaborative filtering.

What was that again?

Key to Collaborative Filtering

Common insight: personal tastes are *correlated*

If Alice and Bob both like *X* and Alice likes *Y*, then Bob is more likely to like *Y*, especially (perhaps) if Bob knows Alice.

Types of Collaborative Filtering

1. Neighborhood- or Memory-based
2. Model-based
3. Hybrid

Types of Collaborative Filtering

1. **Neighborhood- or Memory-based**

   We'll talk about this type now.

2. **Model-based**

3. **Hybrid**

Neighborhood-based CF

A subset of users are chosen based on their similarity to the active users, and a weighted combination of their ratings is used to produce predictions for this user.

Neighborhood-based CF

It has three steps:

1. Assign a weight to all users with respect to similarity with the active user

2. Select *k* users that have the highest similarity with the active user—commonly called the *neighborhood*.

3. Compute a prediction from a weighted combination of the selected neighbors' ratings.

Neighborhood-based CF

Step 1

In step 1, the weight $w_{a,u}$ is a measure of similarity between the user u and the active user a. The most commonly used measure of similarity is the Pearson correlation coefficient between the ratings of the two users:

$$w_{a,u} = \frac{\sum_{i \in I}(r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I}(r_{a,i} - \bar{r}_a)^2 \sum_{i \in I}(r_{u,i} - \bar{r}_u)^2}}$$

where I is the set of items rated by both users, $r_{u,i}$ is the rating given to item i by user u, and $\bar{r}_u$ is the mean rating given by user u.

Neighborhood-based CF

Step 2

In step 2, some sort of threshold is used on the similarity score to determine the "neighborhood."

Neighborhood-based CF

Step 3

In step 3, predictions are generally computed as the weighted average of deviations from the neighbor's mean, as in:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K} (r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}}$$

where $p_{a,i}$ is the prediction for the active user a for item i, $w_{a,u}$ is the similarity between users a and u, and K is the neighborhood or set of most similar users.

But how do we compute the similarity $w_{a,u}$?

Item-to-Item Matching

- An extension to neighborhood-based CF.
- Addresses the problem of high computational complexity of searching for similar users.
- The idea:

Rather than matching similar users, match a user's rated items to similar items.

Item-to-Item Matching

In this approach, similarities between pairs of items i and j are computed off-line using Pearson correlation, given by:

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U}(r_{u,i} - \bar{r}_i)^2 \sum_{u \in U}(r_{u,j} - \bar{r}_j)^2}}$$

where U is the set of all users who have rated both items i and j, $r_{u,i}$ is the rating of user u on item i, and $\bar{r}_i$ is the average rating of the ith item across users.

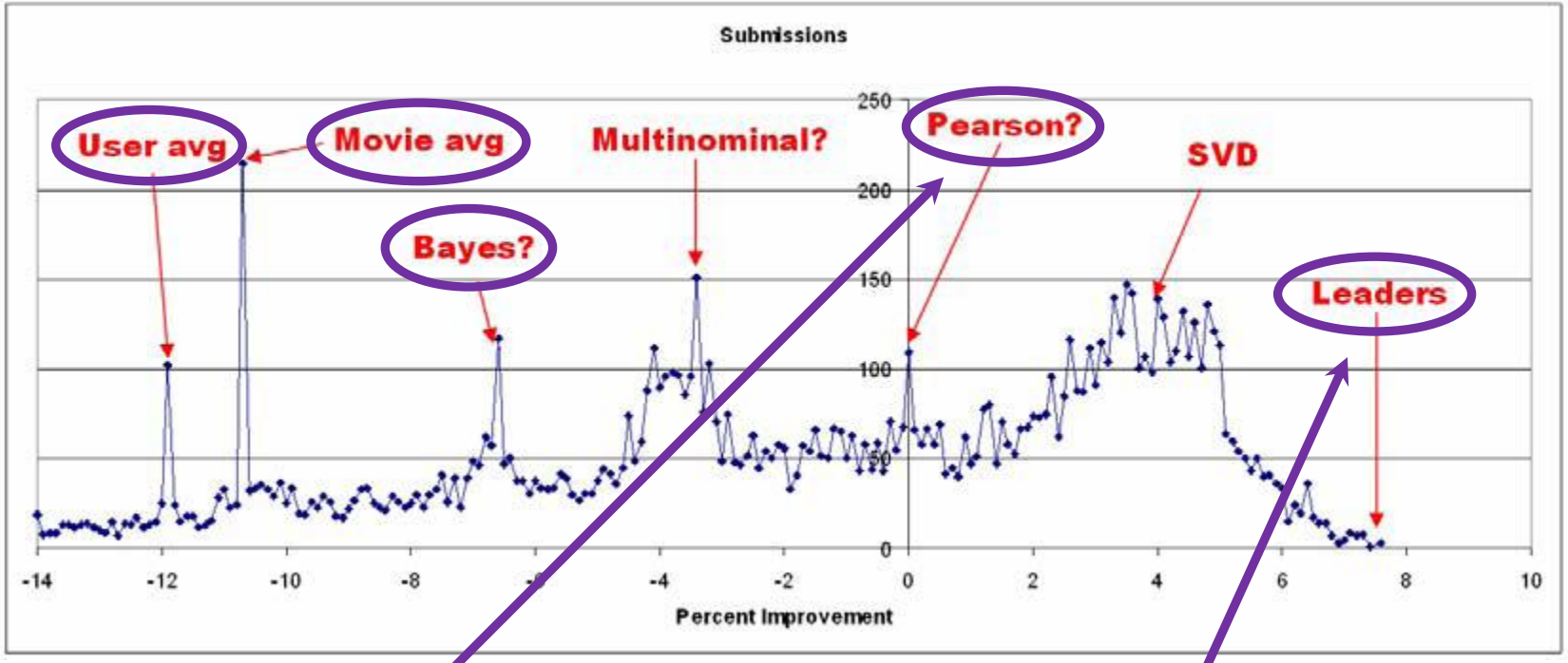
Item-to-Item Matching

Now, the rating for item i for user a can be predicted using a simple weighted average, as in:

$$p_{a,i} = \frac{\sum_{j \in K} r_{u,i} w_{i,j}}{\sum_{j \in K} |w_{i,j}|}$$

where K is the neighborhood set of the k items rated by a that are most similar to i.

The Netflix Prize Methods



Item-oriented collaborative filtering using Pearson correlation gets us right about here.

So how do we get here?

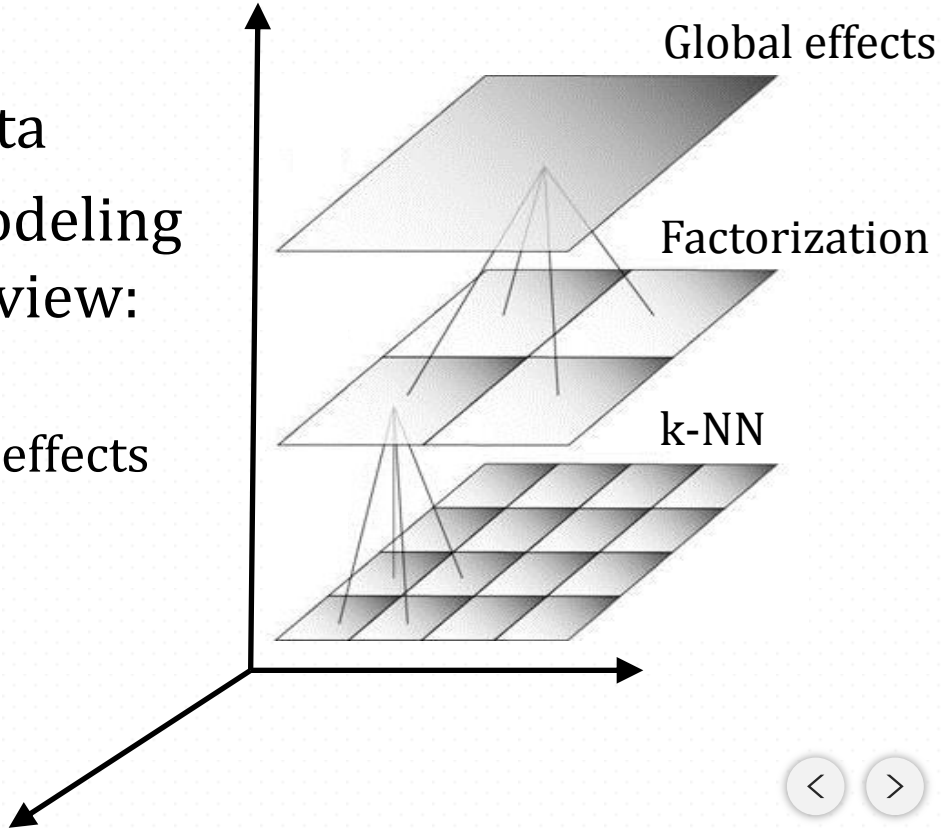
Generalizing the Recommender System

- Use an ensemble of complementing predictors.
- Many seemingly different models expose similar characteristics of the data, and will not mix well.
- Concentrate efforts along three axes.
  - Scale
  - Quality
  - Implicit/explicit

The First Axis: Scale

The first axis:

- Multi-scale modeling of the data
- Combine top level, regional modeling of the data, with refined, local view:
  - *kNN*: Extracts local patterns
  - Factorization: Addresses regional effects

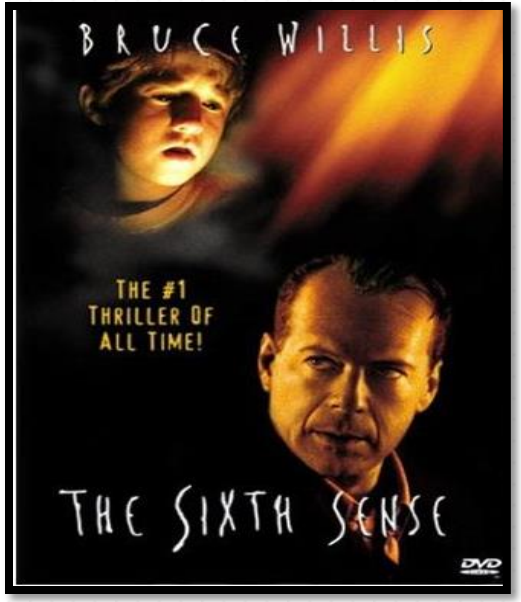


Multi-Scale Modeling: 1st Tier

Global effects:

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

→ Baseline estimation:
Joe will rate *The Sixth Sense* 4 stars

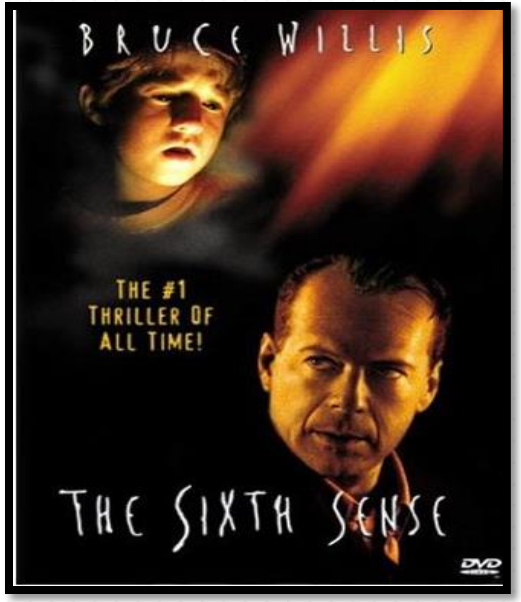


Multi-Scale Modeling: 2nd Tier

Factors model:

- Both *The Sixth Sense* and *Joe* are placed high on the "Supernatural Thrillers" scale

→ Adjusted estimate:
Joe will rate *The Sixth Sense* 4.5 stars

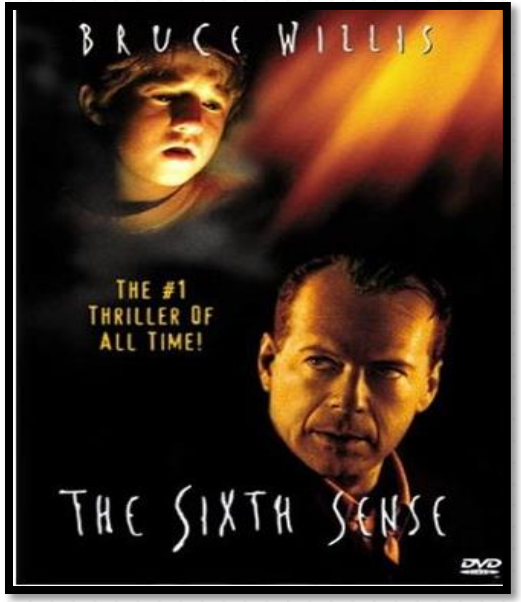


Multi-Scale Modeling: 3rd Tier

Neighborhood Model

- *Joe* didn't like related movie *Signs*

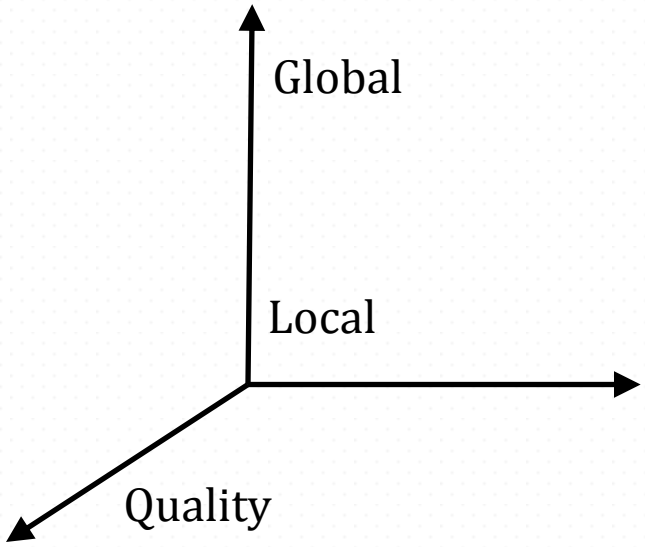
→ Final estimate:
Joe will rate *The Sixth Sense* 4.2 stars



The Second Axis: Model Quality

The second axis:

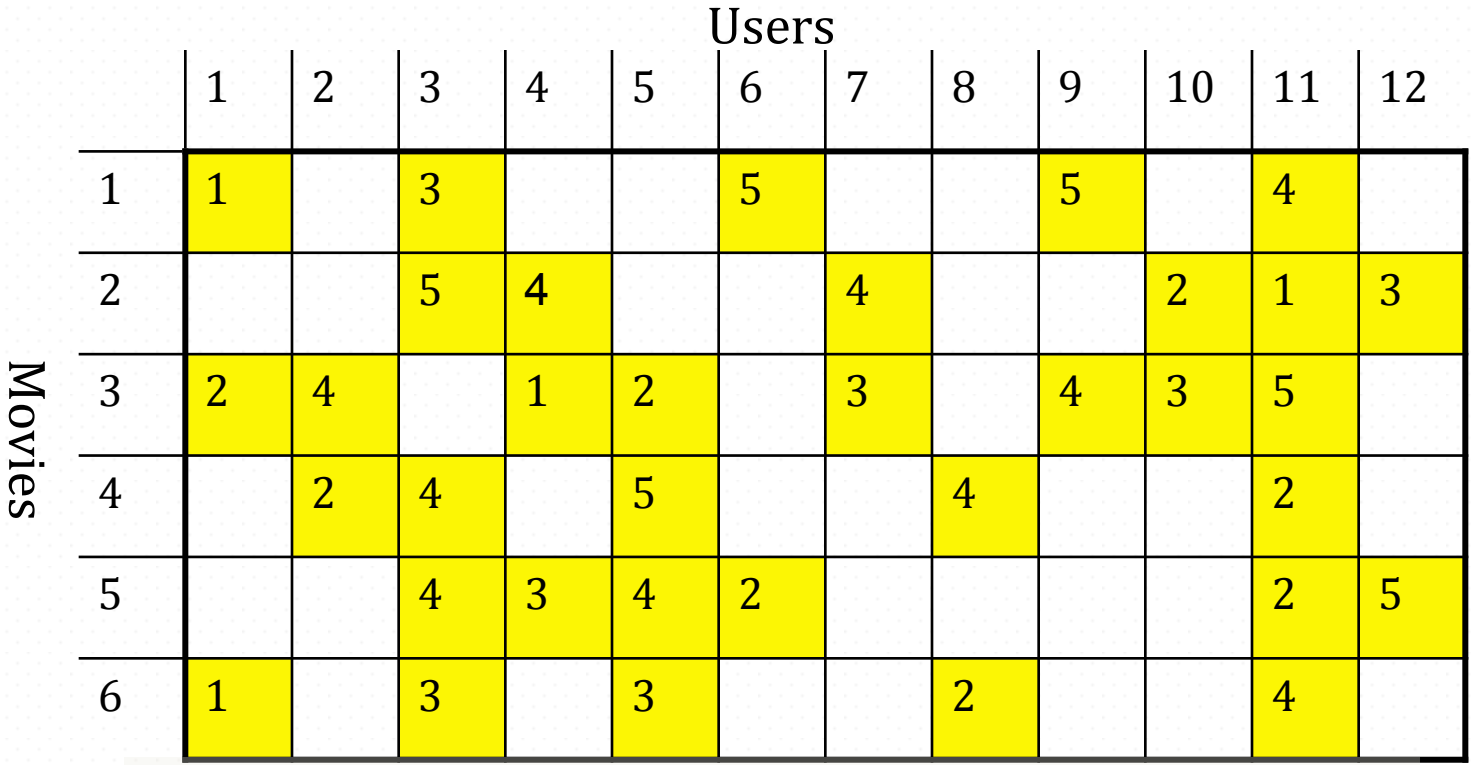
- Quality of modeling
- Make the best out of a model
- Strive for:
  - Fundamental derivation
  - Simplicity
  - Avoid overfitting
  - Robustness against number of iterations, parameter settings, etc.
- Optimizing is good, but don't overdo it!



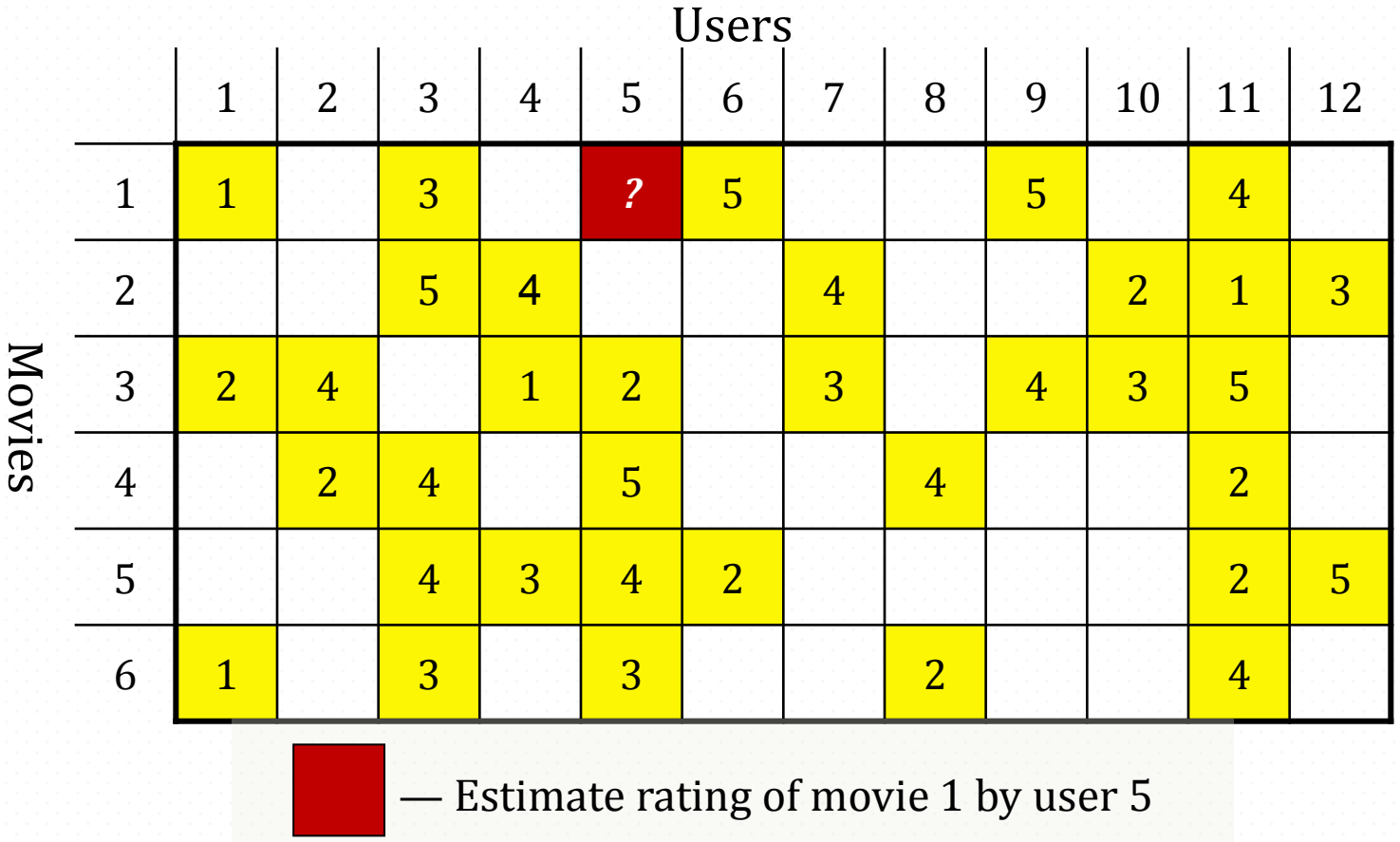
Local Modeling via *k*NN

- Earliest and most popular collaborative filtering method.
- Derive unknown ratings from those of "similar" items (movie-movie variant).
- A parallel user-user flavor.
  - Rely on ratings of like-minded users

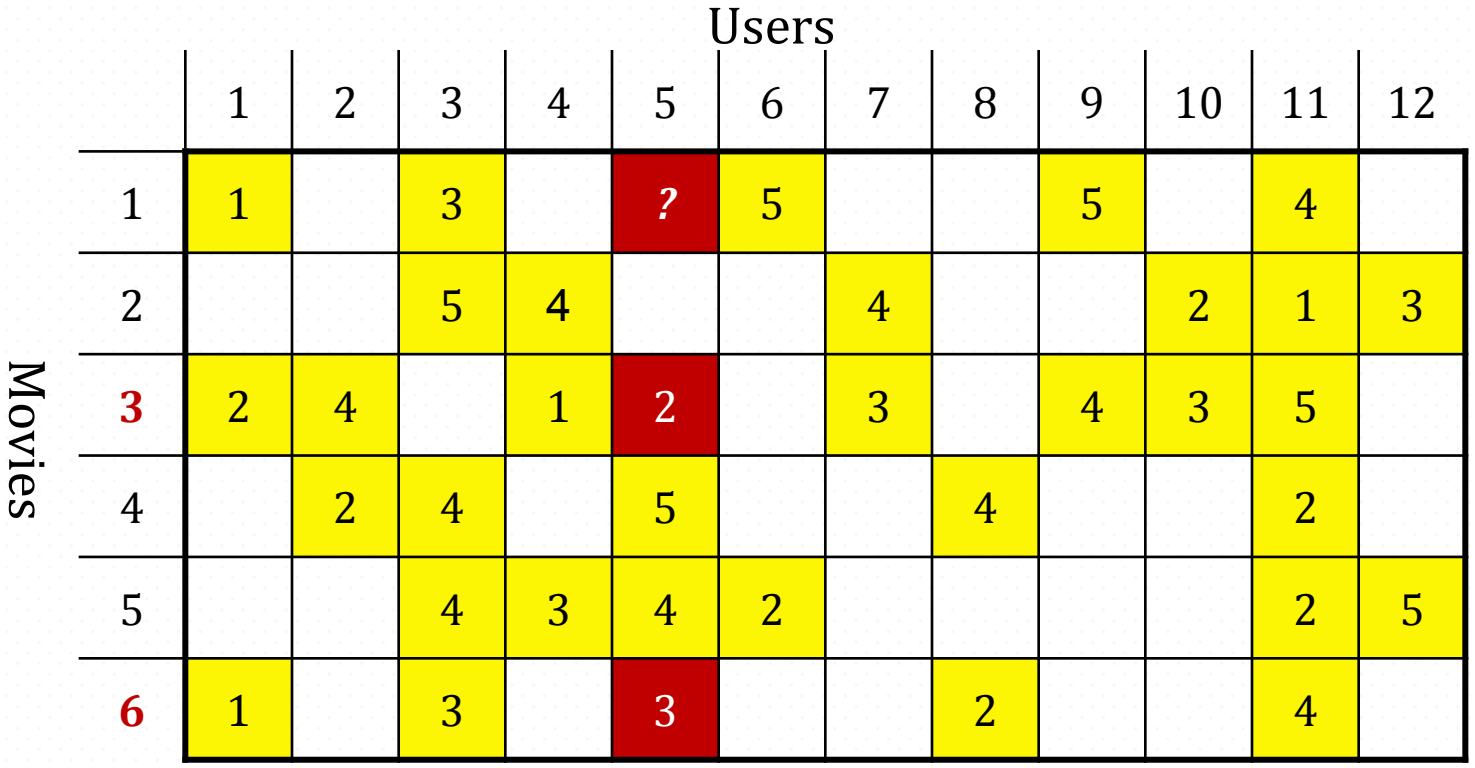
Collaborative Filtering with *k*NN



Collaborative Filtering with *k*NN



Collaborative Filtering with *k*NN



Neighbor selection: Identify movies similar to 1, rated by user 5

Collaborative Filtering with *k*NN

Movies \ Users	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Compute similarity weights: $s_{13} = 0.2, s_{16} = 0.3$

Collaborative Filtering with *k*NN

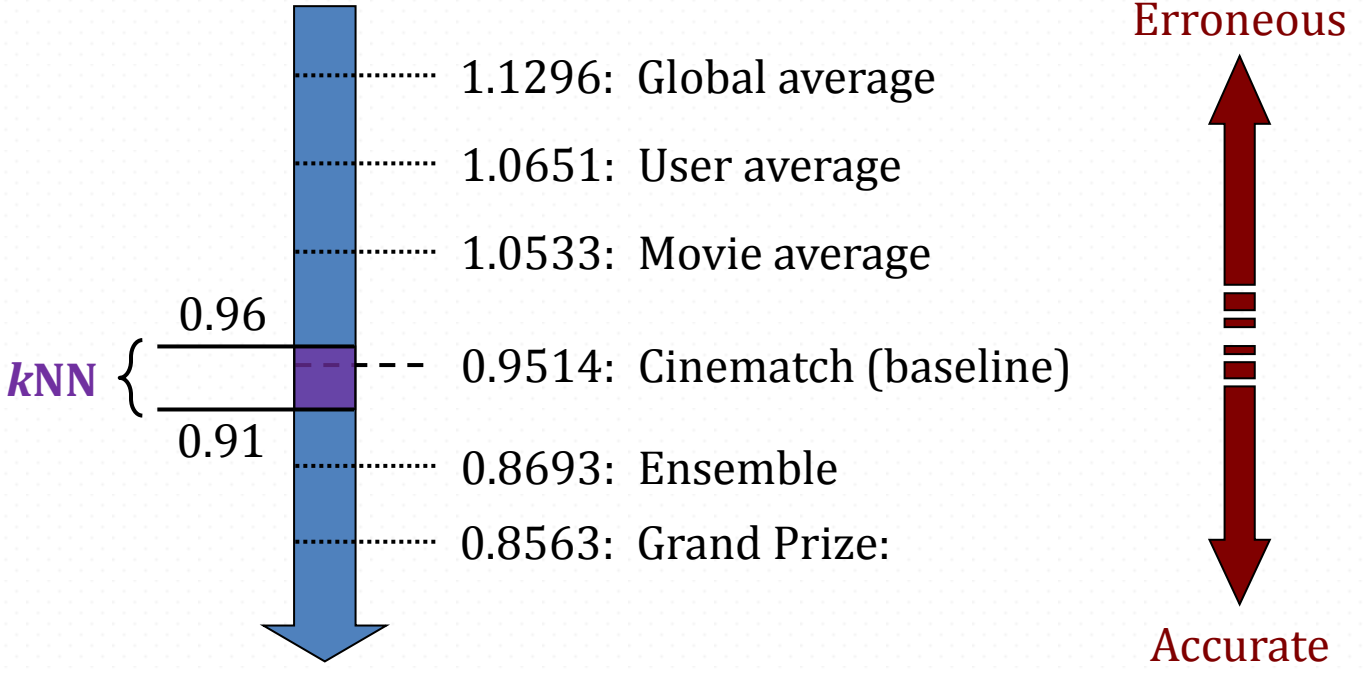
Movies \ Users	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		***2.6***	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Predict by taking weighted average: $(0.2 \times 2 + 0.3 \times 3)/(0.2 + 0.3) = 2.6$

Properties of *k*NN

- Intuitive.
- No substantial preprocessing is required.
- Easy to explain reasoning behind a recommendation.
- Accurate?

*k*NN on the Error (RMSE) Scale



Item-Oriented *k*NN CF

- Problems:
  - Suppose that a particular item is predicted perfectly by a subset of the neighbors, where the predictive subset should receive all the weight. Pearson correlation cannot do this.
  - Suppose the neighbors set contains three movies that are highly correlated with each other. Basic neighborhood methods do not account for interactions among neighbors.
  - Suppose that an item has no useful neighbors rated by a particular user. The standard formula uses a weighted average of rates for the uninformative neighbors.

Interpolation Weights

To address the problem of arbitrary similarity measures, we can use a weighted sum rather than a weighted average:

$$p_{a,i} = \bar{r}_a + \sum_{u \in K} \left(r_{u,i} - \bar{r}_u\right) \times w_{a,u}$$

Now, we can allow $\sum_{u \in K} w_{a,u} \neq 1$.

Interpolation Weights

To address the other problems, we can model relationships between item i and its neighbors. This can be learned through a least squares problem from all other users that rated i:

$$\min_{w} \sum_{v \neq K} \left((r_{vi} - b_{vi}) - \sum_{u \in K} w_{a,u} (r_{vu} - b_{vu}) \right)^2$$

Interpolation Weights

The Result:

- Interpolation weights derived based on their role; no use of an arbitrary similarity measure.
- Explicitly account for interrelationships among the neighbors.

Challenges:

- Dealing with missing values.
- Avoiding overfitting.
- Efficient implementation.

From Local to Latent Trends

Inherently, nearest neighbors is a local technique.
What about capturing non-local, or latent, trends?

Latent Factor Models

- Decompose user ratings on movies into separate item and movie matrices to capture latent factors. Frequently performed using singular value decomposition (SVD).
- Estimate unknown ratings as inner-products of factors.

Ratings

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

~

Movies

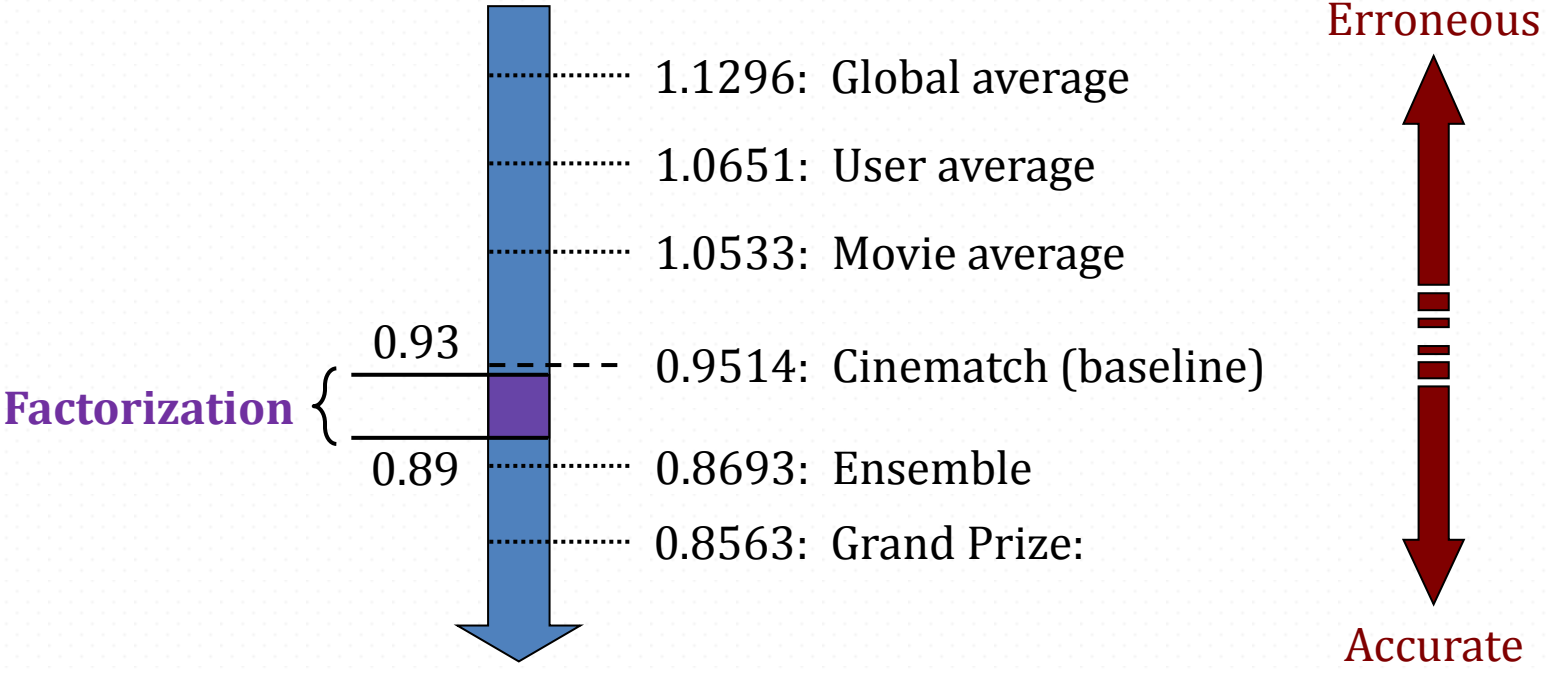
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Users

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

- Very powerful model, but can easily overfit.

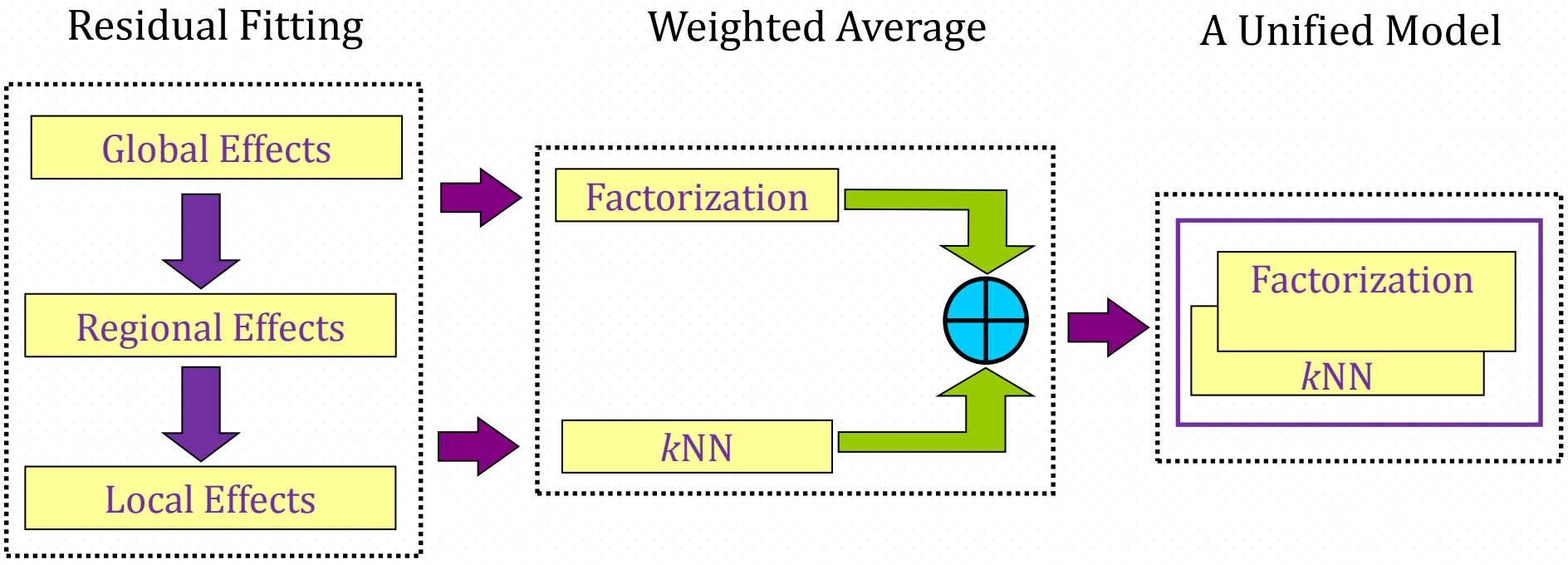
Factorization on the Error (RMSE) Scale



Ensemble Creation

- Factorization and *k*NN models are used at various scales.
- These models can be combined to form an ensemble.
- Stacked generalization or blending is used.
  - A linear regression model can be trained over the base model predictions.
  - Models can be weighted differently at different scales.

Combining Multi-Scale Views



Seek Alternative Perspectives

The previous models all address the movies.

The problem, however, is about users!

The Third Axis: Implicit Information

- Improve accuracy by exploiting implicit feedback.
- Implicit behavior is abundant and easy to collect:
  - Rental history, search patterns, browsing history, etc.
- Allows predicting personalized ratings for users that never rated.

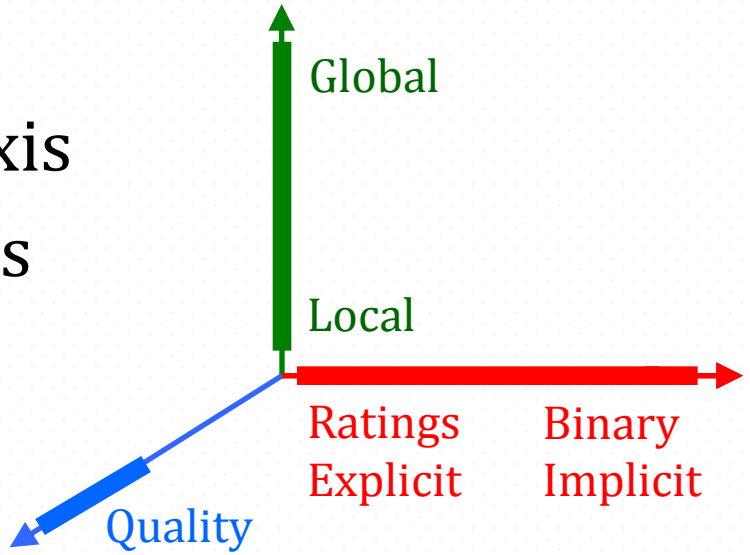
The Idea:

Characterize users by *which* movies they rated, rather than *how* they rated.

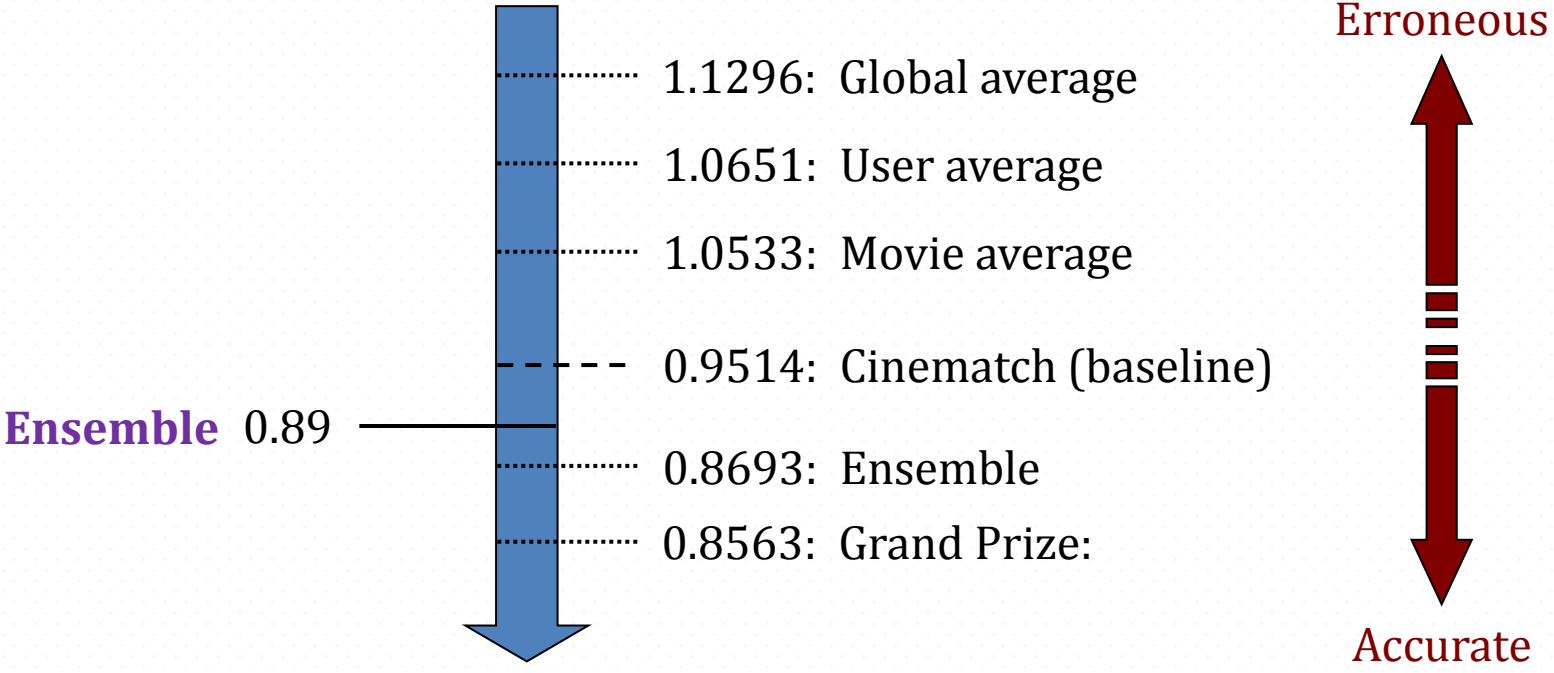
The Big Picture

Where do you want to be?

- All over the global-local axis
- Relatively high on the quality axis
- All over the explicit-implicit axis



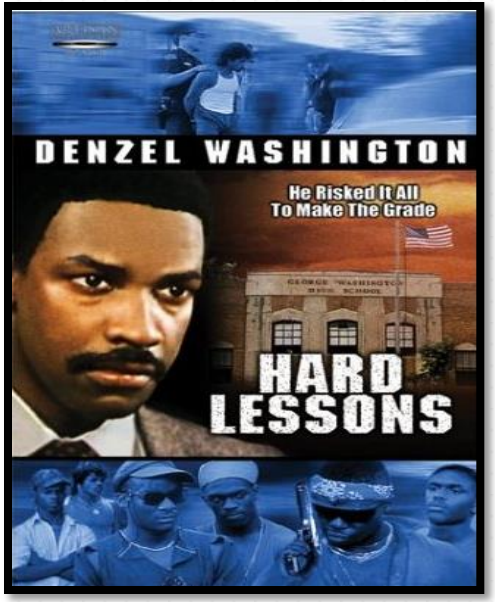
Ensemble on the Error (RMSE) Scale



The Take-Away Messages

Solving challenging data mining and data science problems require you to:

1. **Think deeply**
   - Design better, more innovative algorithms.
2. **Think broadly**
   - Use ensembles of multiple predictors.
3. **Think differently**
   - Model the data from different perspectives and in different ways.



Follow me of LinkedIn for more:
Steve Nouri
https://www.linkedin.com/in/stevenouri/