Recommendations

David Anderson
@alpinegizmo

Impact of Recommendations

Some reported/rumored numbers:

Spotify–40% increase in # of artists listened to
Netflix—40% increase in videos viewed
Amazon—30% increase in sales
Yahoo News—40% increase in time on site

'Fiction is outperforming reality': how YouTube's algorithm distorts truth

An ex-YouTube insider reveals how its recommendation algorithm promotes divisive clips and conspiracy videos. Did they harm Hillary Clinton's bid for the presidency?

Facebook vows to protect Australian election against interference attempts

[Facebook] said it was committed to stopping "people who want to abuse our platform to harm the democratic process".

Sydney Morning Herald
22 July 2018

'I made Steve Bannon's psychological warfare tool': meet the data war whistleblower

Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach

Whistleblower describes how firm linked to former Trump adviser Steve Bannon compiled user data to target American voters

esh
@eshvk

Following

Humble Request: Don't invest in Machine Learning or "AI" before you invest in metrics, instrumentation and hypothesis testing.

RETWEETS 99 LIKES 213

5:50 AM - 11 May 2017

8 99 213

Many Techniques for Generating Recommendations

- Similarity
- Graph algorithms
- Matrix factorization
- Co-occurrence analysis
- Hybrids
- Deep learning

item-based

- for every item i that u has no preference for yet
  - for every item j that u has a preference for
    - compute similarity s between i and j
    - add u's preference for j, weighted by s, to a running average
- return the top items, ranked by weighted average

user-based

- for every item i that u has no preference for yet
  - for every other user v that has a preference for i
    - compute similarity s between u and v
    - add v's preference for i, weighted by s, to a running average
- return the top items, ranked by weighted average

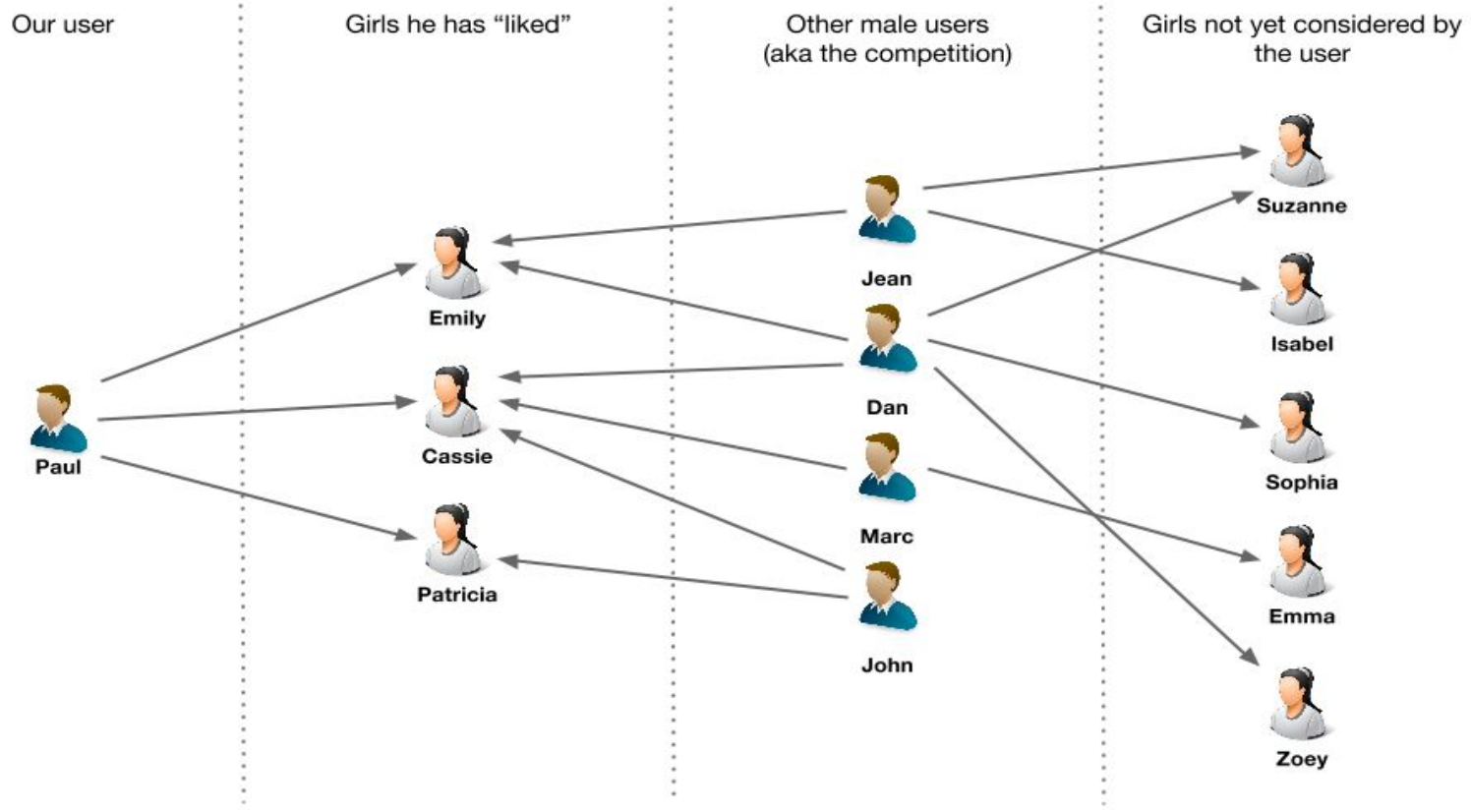
item-based vs user-based

- item-based is more scalable
  - Recommenders must scale with the number of items and users they must deal with. Which cost dominates will vary, depending on the situation.
  - Estimates of item similarities are likely to converge over time — whereas new info about a user can cause large changes in their nearest user neighborhood.
  - We can pre-compute and cache similarities that converge, which can give item-based recommenders a performance advantage.
- item-based is more easily explained
  - “this is similar to X, which you liked”

Case Study: Dating Site

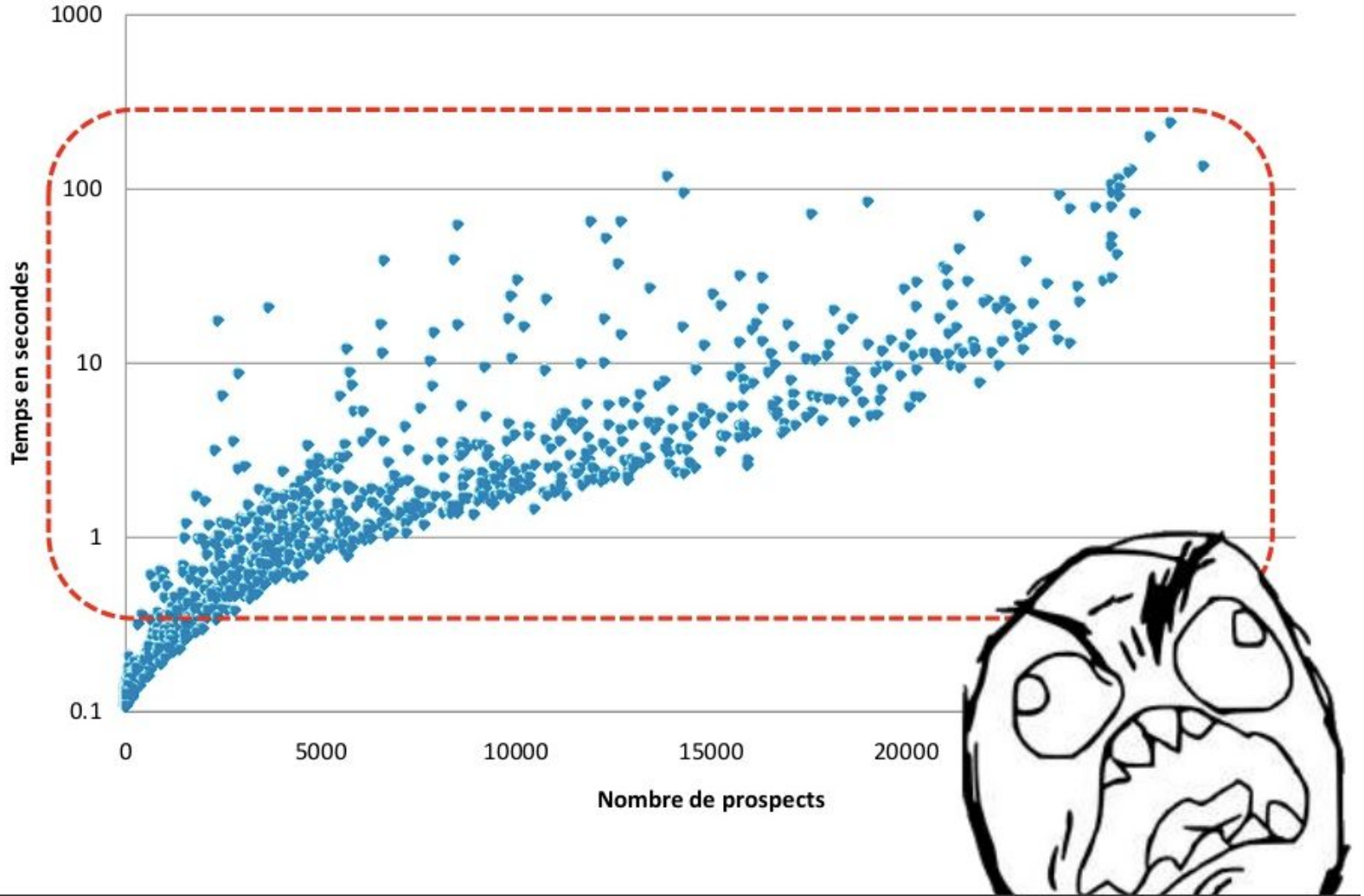
- users
  - age
  - who they have "liked"

Dating Graph

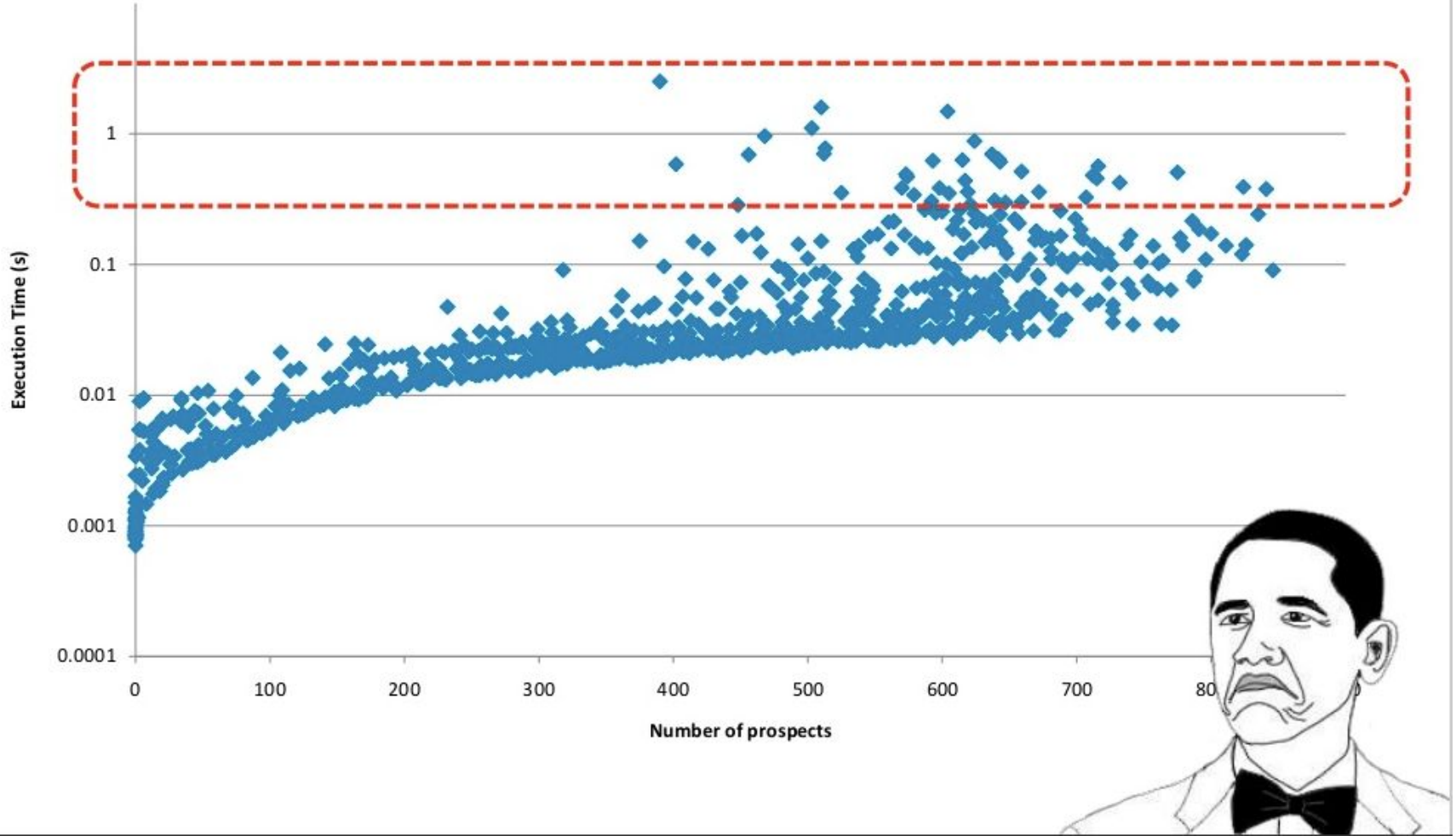


neo4j query

```
START me = node:node_auto_index(abo_id='170021316')
MATCH me-[:wink|favorite]->she<-[:wink|favorite]-slm-[:wink|favorite]->prospects
WHERE not(me = slm) and not(me--prospects)
RETURN count(*) AS prospectsWeight, prospects.attraction AS attraction, prospects.abo_id AS abo_id
ORDER BY count DESC, attraction DESC, abo_id DESC;
```


1000
100
10
1
0.1
Temps en secondes
0
5000
10000
15000
20000
Nombre de prospects

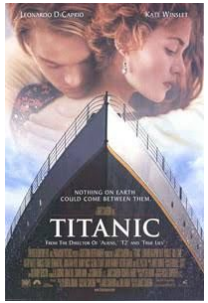
```
START me = node:node_auto_index(abo_id='170021316')
MATCH me-[:wink|favorite]->she<-[:wink|favorite]-slm
WHERE not(me = slm) AND slm.age > me.age -10 and slm.age < me.age + 5
AND slm.nbFavoriteOut + slm.nbWinkOut <  100
WITH me, slm, count(*) AS slmWeight
ORDER BY slmWeight DESC
LIMIT 20
MATCH slm-[:wink|favorite]->prospects
WITH me, prospects, sum(slmWeight) AS prospectsWeight
WHERE not(me--prospects)
RETURN prospectsWeight AS count, prospects.attraction AS attraction, prospects.abo_id AS abo_id
ORDER BY count DESC, attraction DESC, abo_id DESC
LIMIT 100;
```

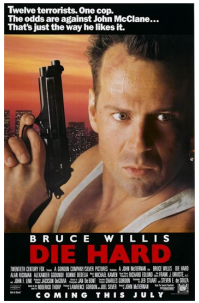

Execution Time (s)
1
0.1
0.01
0.001
0.0001
0
100
200
300
400
500
600
700
Number of prospects

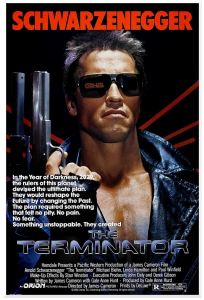
Collaborative Filtering

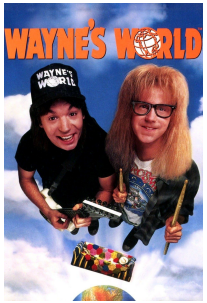
- A widely used technique
- Uses the aggregated behavior of the crowd to suggest relevant items to individual users
- Typically implemented via matrix factorization

The User-Item Matrix



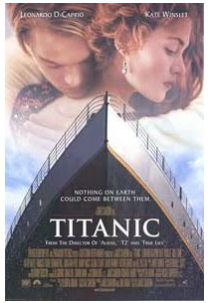


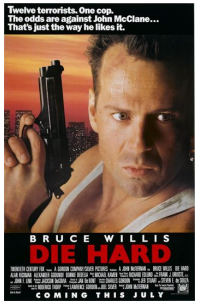


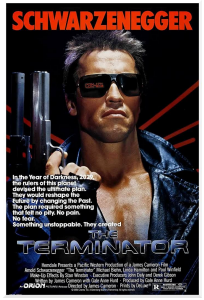


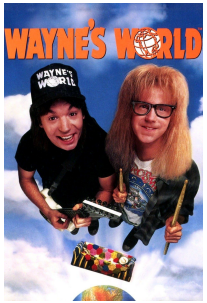
romo	5	5				
acto			5	5		
como					5	5
act-com1		1	5	5	5	
act-com2	1	1	5		5	5
rom-act1	5	5		5	1	1
rom-act2	5			5		1
rom-act3	5	5	5		1	

Predictions?



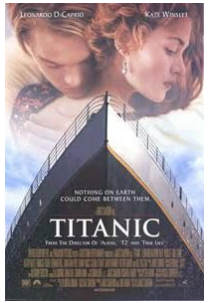


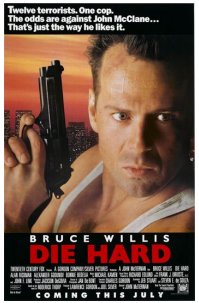


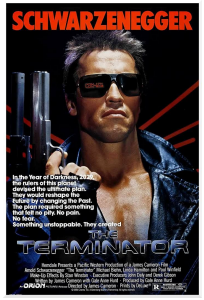


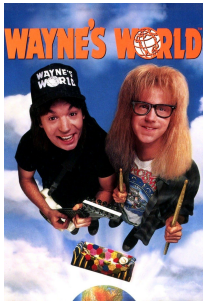
romo	5	5	?	?	?	?
acto			5	5		
como					5	5
act-com1		1	5	5	5	
act-com2	1	1	5		5	5
rom-act1	5	5		5	1	1
rom-act2	5			5		1
rom-act3	5	5	5		1	

Predictions?



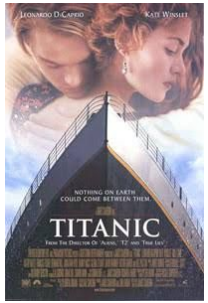


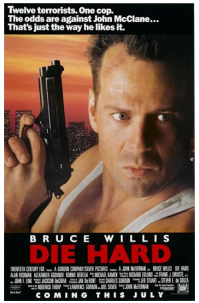


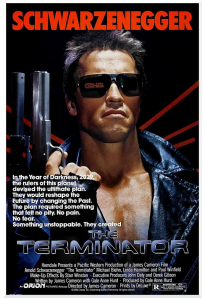


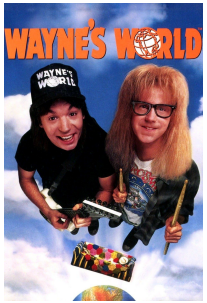
romo	5	5	✔	✔	✖	✖
acto			5	5		
como					5	5
act-com1		1	5	5	5	
act-com2	1	1	5		5	5
rom-act1	5	5		5	1	1
rom-act2	5			5		1
rom-act3	5	5	5		1	

Predictions?



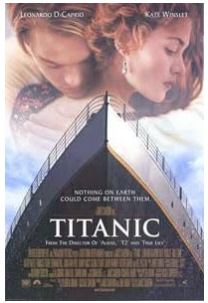


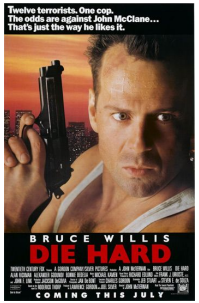


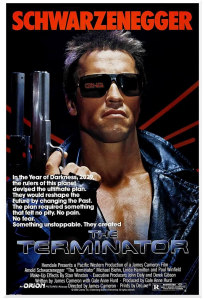


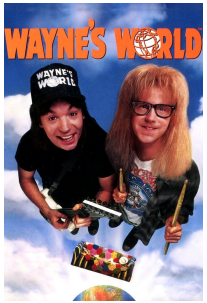
romo	5	5				
acto			5	5		
como	✖	✖	✔	✔	5	5
act-com1		1	5	5	5	
act-com2	1	1	5		5	5
rom-act1	5	5		5	1	1
rom-act2	5			5		1
rom-act3	5	5	5		1	

Predictions?



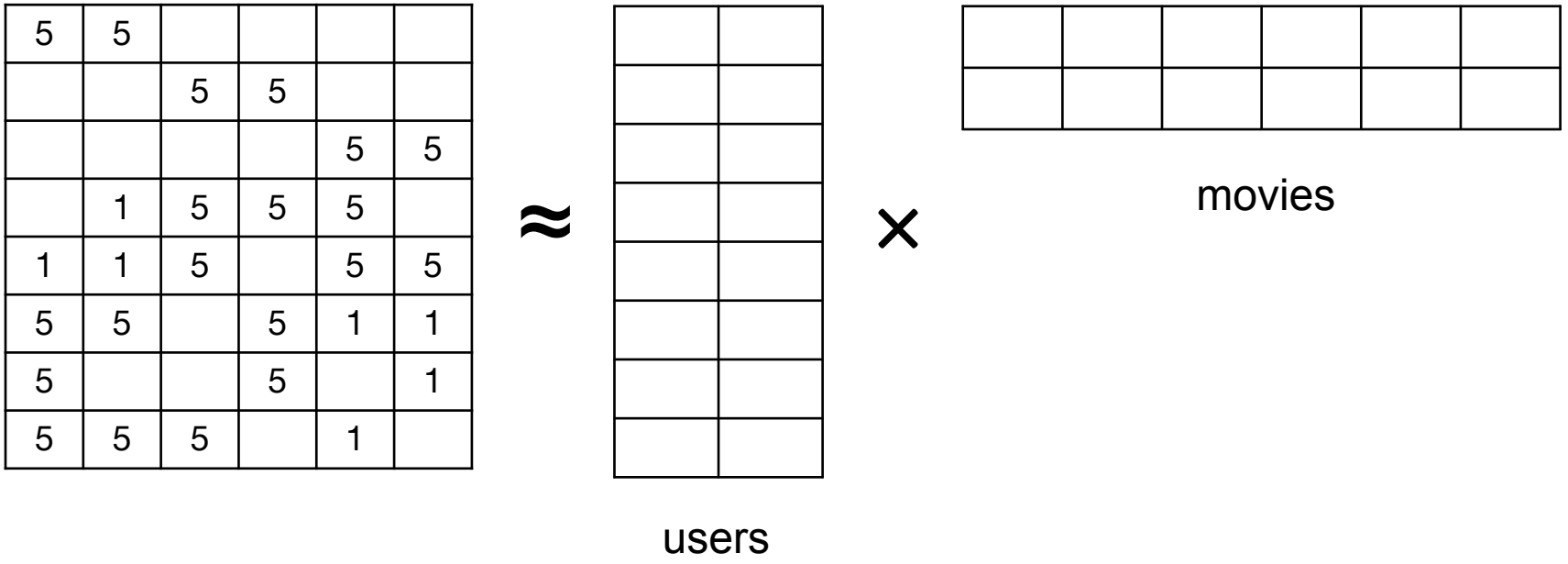




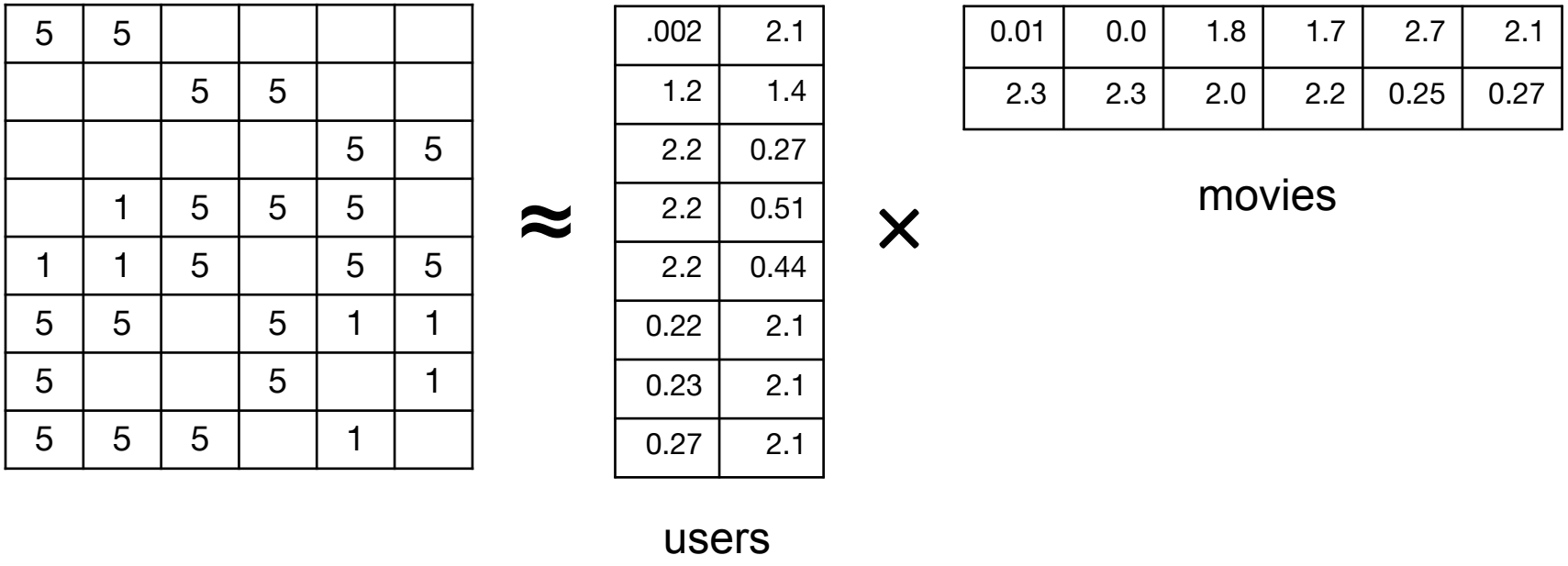


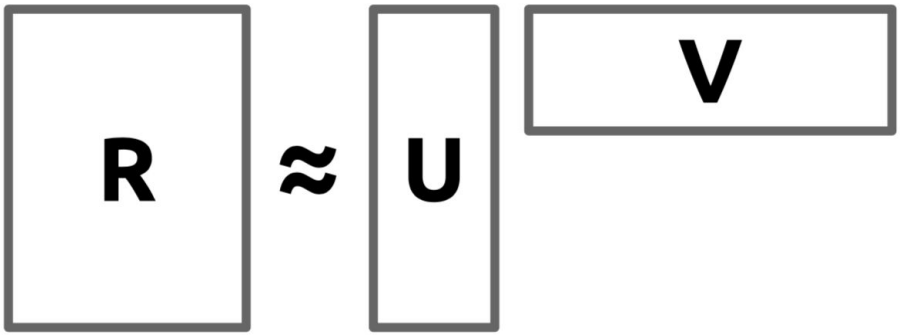
romo	5	5				
acto	?	?	5	5	?	?
como					5	5
act-com1		1	5	5	5	
act-com2	1	1	5		5	5
rom-act1	5	5		5	1	1
rom-act2	5			5		1
rom-act3	5	5	5		1	

ALS Matrix Factorization



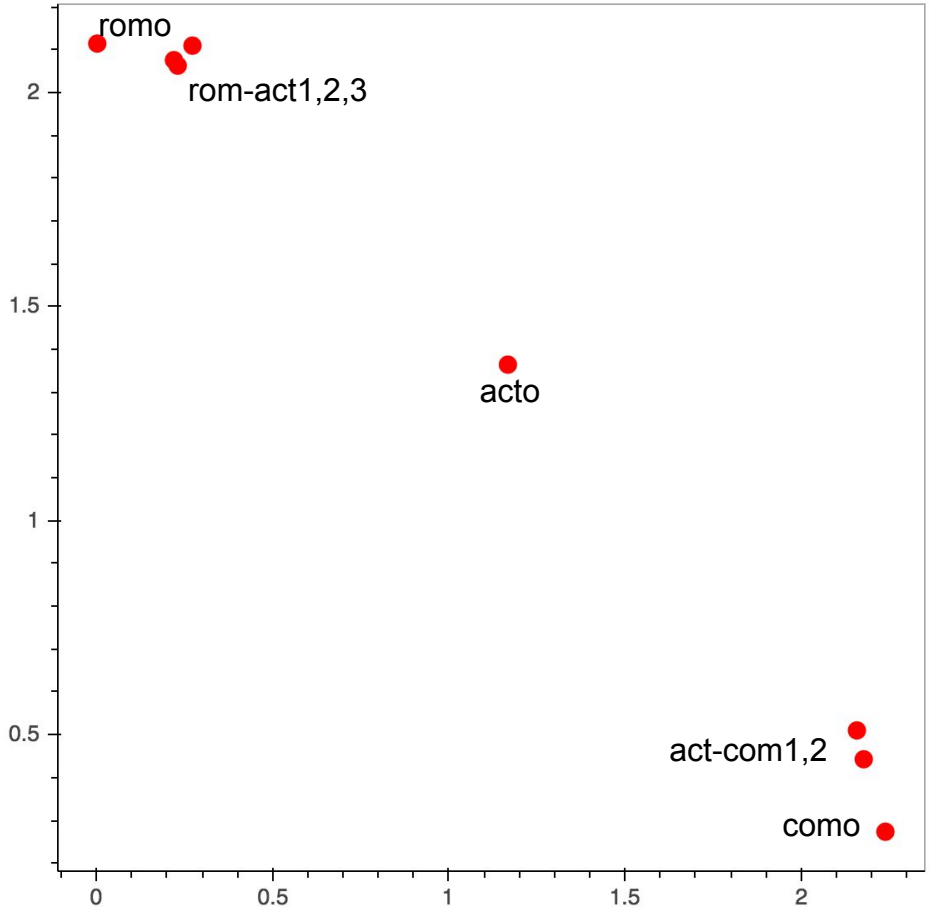
ALS Matrix Factorization

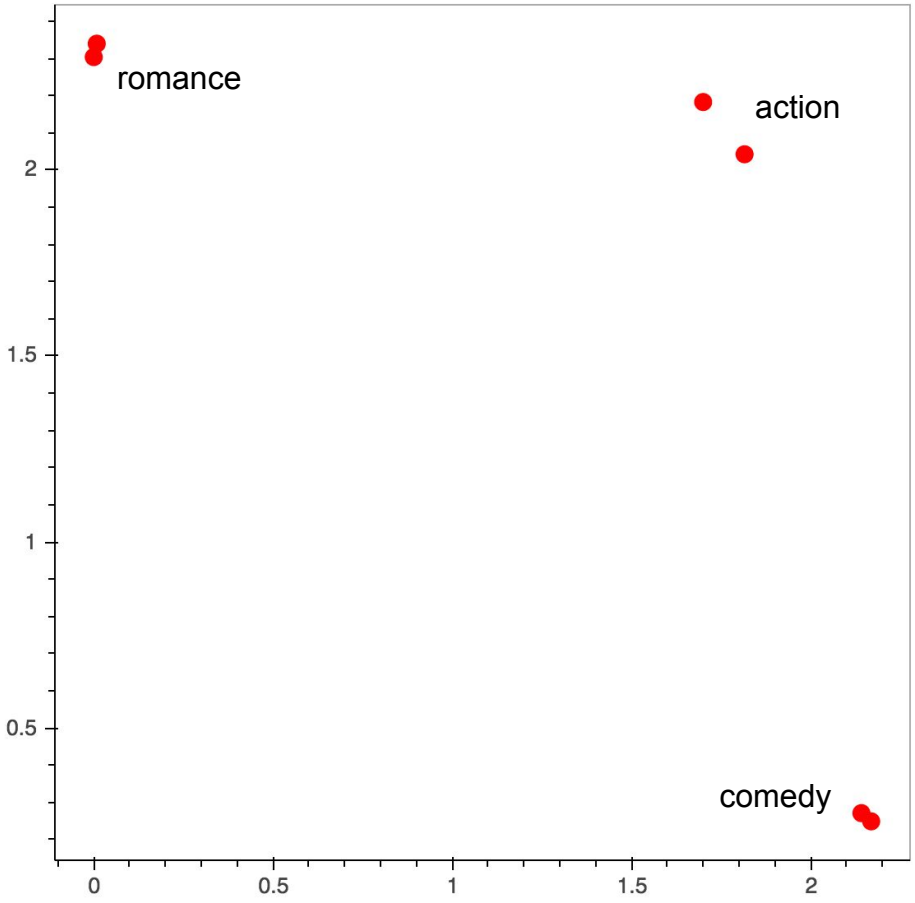




$$\min_{u,v} \sum_{i,j \in R} (r_{i,j} - u_i^T v_j) + \lambda(\|u_i\|^2 + \|v_j\|^2)$$

```
val users = model.
  userFeatures.
  collect.sortBy(_._1)
```






```
val movies = model.
  productFeatures.
  collect.sortBy(_._1)
```

Using the ALS Model

```
val usersProductsRDD = dataRDD.map(x => (x(0), x(1)))
val predictions = model.predict(usersProductsRDD)
```

.002	2.1
1.2	1.4
2.2	0.27
2.2	0.51
2.2	0.44
0.22	2.1
0.23	2.1
0.27	2.1

users

×

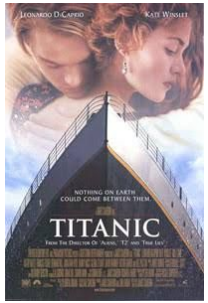
0.01	0.0	1.8	1.7	2.7	2.1
2.3	2.3	2.0	2.2	0.25	0.27

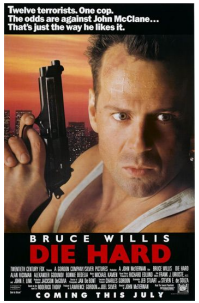
movies

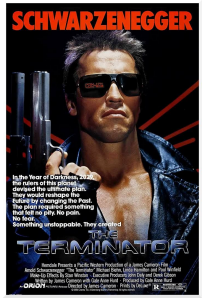
=

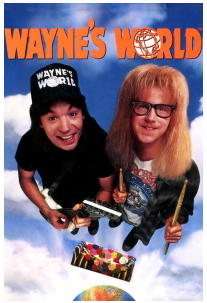
5	5	4	4	0	0
3	3	5	5	3	3
0	0	4	4	5	5
1	1	5	5	5	5
1	1	5	5	5	5
5	5	5	5	1	1
5	5	5	5	1	1
5	5	5	5	1	1

Resulting Predictions







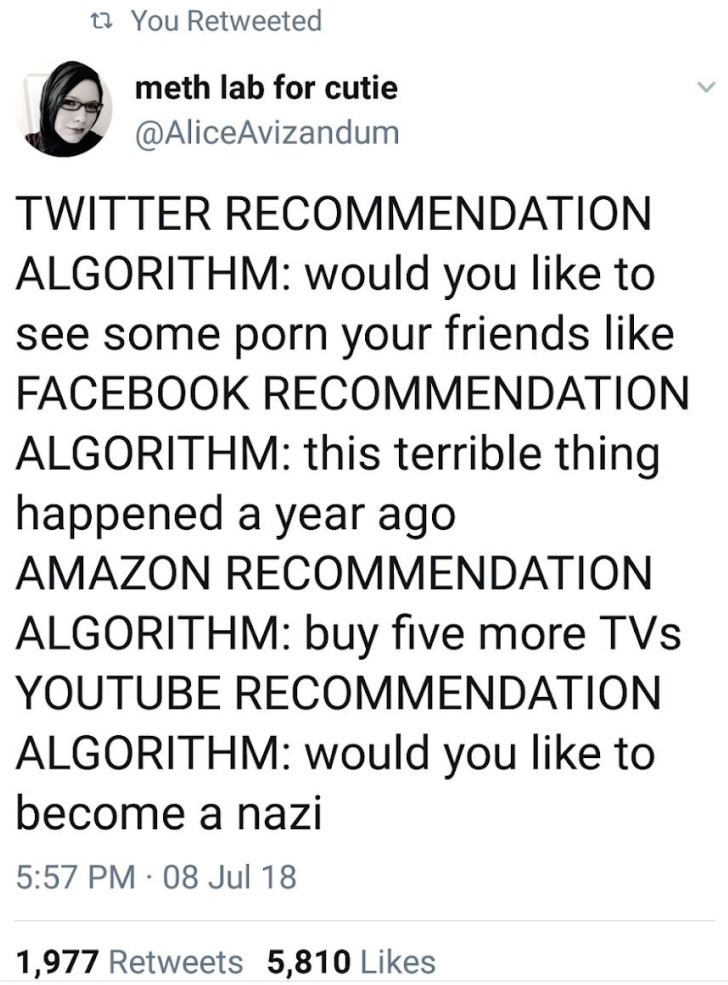


romo	5	5	4	4	0	0
acto	3	3	5	5	3	3
como	0	0	4	4	5	5
act-com1	1	1	5	5	5	5
act-com2	1	1	5	5	5	5
rom-act1	5	5	5	5	1	1
rom-act2	5	5	5	5	1	1
rom-act3	5	5	5	5	1	1

Evaluating Recommendations

Conventional wisdom:

All recommenders are lousy



Why do evaluation?

1. To drive hyperparameter search
2. To maximize effectiveness in a business sense

RMSE – Root Mean Squared Error

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \left(P_i - O_i\right)^2}{n}}$$

P_i: predictions
O_i: observations

- ALS seeks to minimize RMSE
- We want hyperparameters that minimize RMSE

Other Evaluation Criteria

- Coverage
- Diversity
- Serendipity
- Business concerns
- Impact on KPIs
  - revenue, user engagement, ...

Coverage

- What fraction of the product catalog is being recommended?
- Are you able to surface products with strong appeal for a limited audience?

Diversity

- Is the set of products recommended to a given user concentrated in only a few categories, or spread across the entire range of products?

Serendipity

- Can your recommendations surprise and delight?
- Can you tell?

Business Concerns

- Seasonality
- Out of stock
- The “cold start” problem
  - new products, new users

Implicit Feedback

Explicit feedback:

- supplied by the user, e.g., ★★★
- may be positive or negative
- values indicate preference

Explicit feedback:

- supplied by the user, e.g., ★★★
- may be positive or negative
- values indicate preference

Implicit feedback:

- inferred from users' actions
  - e.g., purchases, page views, time spent watching
- no negative feedback; hard to know what a user didn't like
- noisy data
  - e.g., purchase may've been a gift
- values indicate confidence

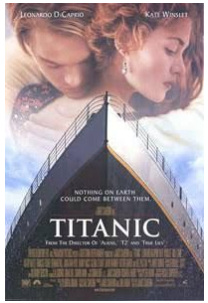
Explicit feedback:

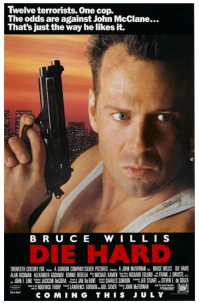
- supplied by the user, e.g., ★★★
- may be positive or negative
- values indicate preference
- data density is low

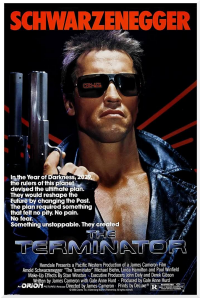
Implicit feedback:

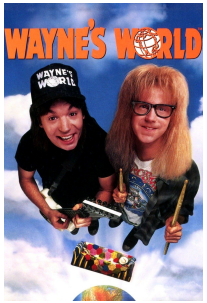
- inferred from users' actions
  - e.g., purchases, page views, time spent watching
- no negative feedback; hard to know what a user didn't like
- noisy data
  - e.g., purchase may've been a gift
- values indicate confidence
- we can assign values to all user-item pairs

Implicit User-Item Matrix









romo	1	1	0	0	0	0
acto	0	0	1	1	0	0
como	0	0	0	0	1	1
act-com1	0	0	1	1	1	0
act-com2	0	0	1	0	1	1
rom-act1	1	1	0	1	0	0
rom-act2	1	0	0	1	0	0
rom-act3	1	1	1	0	0	0

Setup with Implicit Data

```
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel
import org.apache.spark.mllib.recommendation.Rating

val rawdata = """1,1,0,0,0,0
              |0,0,1,1,0,0
              |0,0,0,0,1,1
              |0,0,1,1,1,0
              |0,0,1,0,1,1
              |1,1,0,1,0,0
              |1,0,0,1,0,0
              |1,1,1,0,0,0"""
```

Training the Implicit ALS Model

```
val dataRDD = sc.parallelize(rawdata.split('\n').map(_.split(',').map(_.toInt)))

val ratingsRDD = dataRDD.zipWithIndex.flatMap {
  x => x match { case (userRatings, user) => userRatings.zipWithIndex.map {
    y => y match {
      case (rating, item) => Rating(user.toInt, item.toInt, rating.toDouble)
    } } } }

val als = new ALS().setRank(2).setIterations(20).setLambda(0.01).setNonnegative(true)
als.setImplicitPrefs(true).setAlpha(40)
val model = als.setSeed(4242).run(ratingsRDD)
```

Using the Implicit ALS Model

1	1	0	0	0	0
0	0	1	1	0	0
0	0	0	0	1	1
0	0	1	1	1	0
0	0	1	0	1	1
1	1	0	1	0	0
1	0	0	1	0	0
1	1	1	0	0	0

1	1	1	1	0	0
1	1	1	1	1	1
0	0	1	1	1	1
0	0	1	1	1	1
0	0	1	1	1	1
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	0	0

Alpha, the confidence parameter

1	1	0	1	0	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	0	1	0	1	1
1	1	0	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0

alpha = 0.1

1	1	0	1	0	0
0	0	1	1	1	0
0	0	1	0	1	1
0	0	1	1	1	1
0	0	1	0	1	1
1	1	0	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

alpha = 1
(Spark default)

1	1	1	1	0	0
1	1	1	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	0	1	1	1	1
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	0	0

alpha = 5

1	1	1	1	0	0
1	1	1	1	1	1
0	0	1	1	1	1
0	0	1	1	1	1
0	0	1	1	1	1
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	0	0

alpha = 40
(suggested in
original paper)

Hyperparameter Tuning

Where to Learn More

- ALS-WR for explicit feedback
  - *Large-scale Parallel Collaborative Filtering for the Netflix Prize,* Zhou, etal, 2008
- ALS for implicit feedback
  - *Collaborative Filtering for Implicit Feedback Datasets*, Hu, Koren, and Volinsky, 2008

Overall Plan of Attack

- Use cross-validation
- Measure RMSE on the validation set
- Stop searching when you reach the point of diminishing returns

numIterations

- This is the only available stopping criteria
  - as opposed to continuing until RMSE stops decreasing
- You'll probably be happy somewhere between 5 and 20 – you'll have to do some experiments

rank

- Number of latent factors
- Start small (5 or 10) and increase gradually
- Increasing rank will decrease RMSE
- Increasing rank is computationally expensive

lambda

- Regularization parameter – fights against overfitting
- Spark implements weighted regularization, which allows you to search for an optimal value of lambda on a subset of your dataset
- You'll only need to try a few values

alpha (confidence parameter)

- Only relevant with implicit rating data
- See the previous lesson
- You may need to implement a different evaluation function – RMSE may not be appropriate for your application

rank and lambda

- Search for lambda using a subset of your data
- Using that lambda, search for rank (using the full dataset)
- Having chosen the rank, check if adjusting lambda helps
- Also double-check your assumptions about numIterations

Tricks with Spark ALS

Similarity

- The learned ALS model can be used to compute similarities between items, and between users
- Cosine similarity (using the latent feature vectors) is typically used

Segmentation

- Cluster the users in the n-dimensional space defined by their latent factors
- The resulting segmentation may be useful for marketing purposes, for example
- This may also help bootstrap recommendations for new users (the “cold start” problem)

Representative Products

- Cluster the products using their latent feature vectors
- Select the most popular product in each cluster

Explanations

- How to explain the recommendation of an item for a user?
  - Find the item the user has already interacted with that is the most similar to the recommended item
  - In *Collaborative Filtering for Implicit Feedback Datasets*, Hu etal outline a more principled and more powerful approach

Surprises

- For a given user, find items the user rated highly that the model predicts would be disliked
- For users who are being modeled poorly, consider an alternative approach for generating their recommendations

Real-World Recommendations

David Anderson

Challenges

- Provide a good experience to every user
- Strenuously avoid doing anything contrary to the business' objectives
- Measure impact

Issues in Recommender Systems

- Satisfying business objectives (e.g. low-margin items or overstock)
- Scaling
- Cold start – new users, new products
- Expired items
- Seasonality
- Stale preferences
- Real-time updates
- Grey sheep – users with unusual preferences
- Combine evidence from many interaction types (view, purchase, rate/review)
- Shilling attacks (fake ratings)
- Items some users find offensive

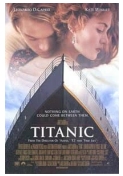
A Taxonomy of Recommenders

- collaborative filtering
  - based on observed interactions
  - performs better than content-based
  - new users and new items suffer from “cold start”
- content-based
  - find neighbors based on similarity of meta-data
  - item-based or user-based
- hybrids
  - ad-hoc
  - principled

Case Study: XING Jobs

- jobs
  - start and end date
  - title
  - description, requirements (text)
  - location
- users
  - titles and descriptions of jobs held
  - skills (keywords)
  - degree(s), major field(s) of study
  - years of experience
  - current employment status
  - jobs previously applied for

Cooccurrence Matrix

						
		3	1	2	0	0
	3		1	1	0	0
	1	1		2	2	1
	2	1	2		1	0
	0	0	2	1		2
	0	0	1	0	2	

Which is the most anomalous co-occurrence?

	A	not A
B	***13***	***1000***
not B	1000	100,000

	A	not A
B	***1***	***0***
not B	0	2

	A	not A
B	***1***	***0***
not B	0	10,000

	A	not A
B	***10***	***0***
not B	0	100,000

Which one is the most anomalous co-occurrence?

0.90

	A	not A
B	*13*	*1000*
not B	1000	100,000

1.95

	A	not A
B	*1*	*0*
not B	0	2

4.52

	A	not A
B	*1*	*0*
not B	0	10,000

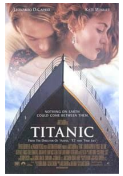
14.3

	A	not A
B	*10*	*0*
not B	0	100,000

Log Likelihood Ratio (LLR)

Dunning Ted, Accurate Methods for the Statistics of Surprise and Coincidence, *Computational Linguistics* vol 19 no. 1 (1993)

LLR

						
		1.6	-0.13	0.77	-1.0	-0.58
			-0.14	0.35	-0.89	-0.50
				0.52	0.82	0.90
					0.35	-0.58
						2.0
						

Recomendations from co-occurrence

$$r = (P^t P)h_p$$

r = recommendations

h_p = a user's history of some action
(purchase for instance)

P = the history of all users' primary action
rows are users, columns are items

$(P^t P)$ = compares column to column using
log-likelihood based correlation test

The “Universal Recommender”

- Traditional collaborative filtering uses only one indicator of preference
- Nothing stops us from going further:

$$r = (P^tP)h_p + (P^tV)h_v + (P^tC)h_c + \ldots$$

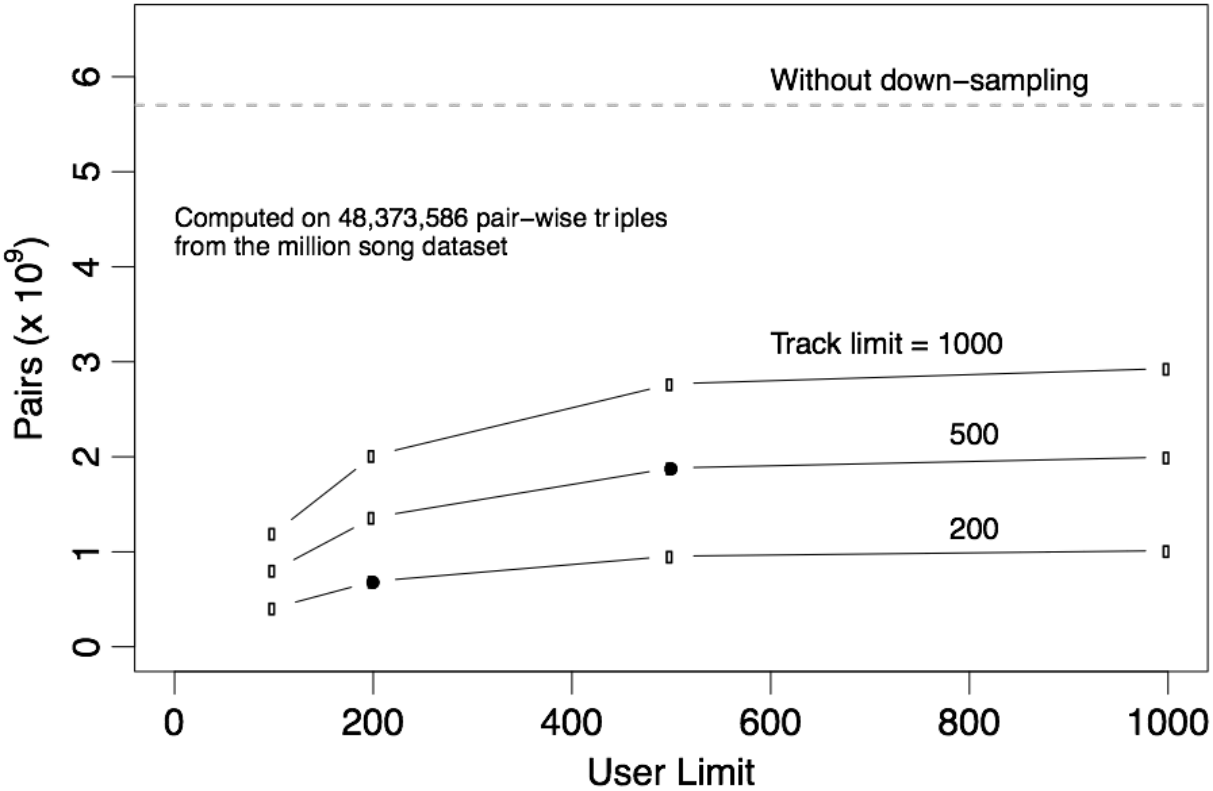
- With this approach, everything we know about the user can be used to improve recommendations—purchases, views, category-preferences, location-preferences, device-preferences, gender, ...
- This *correlated cross-occurrence* technique can also be combined with topic modeling (for example)

LLR Notebook

Scalability of Co-occurrence

- full co-occurrence doesn’t scale
- can be made more scalable by downsampling (bounding) the interactions

Benefit of down–sampling

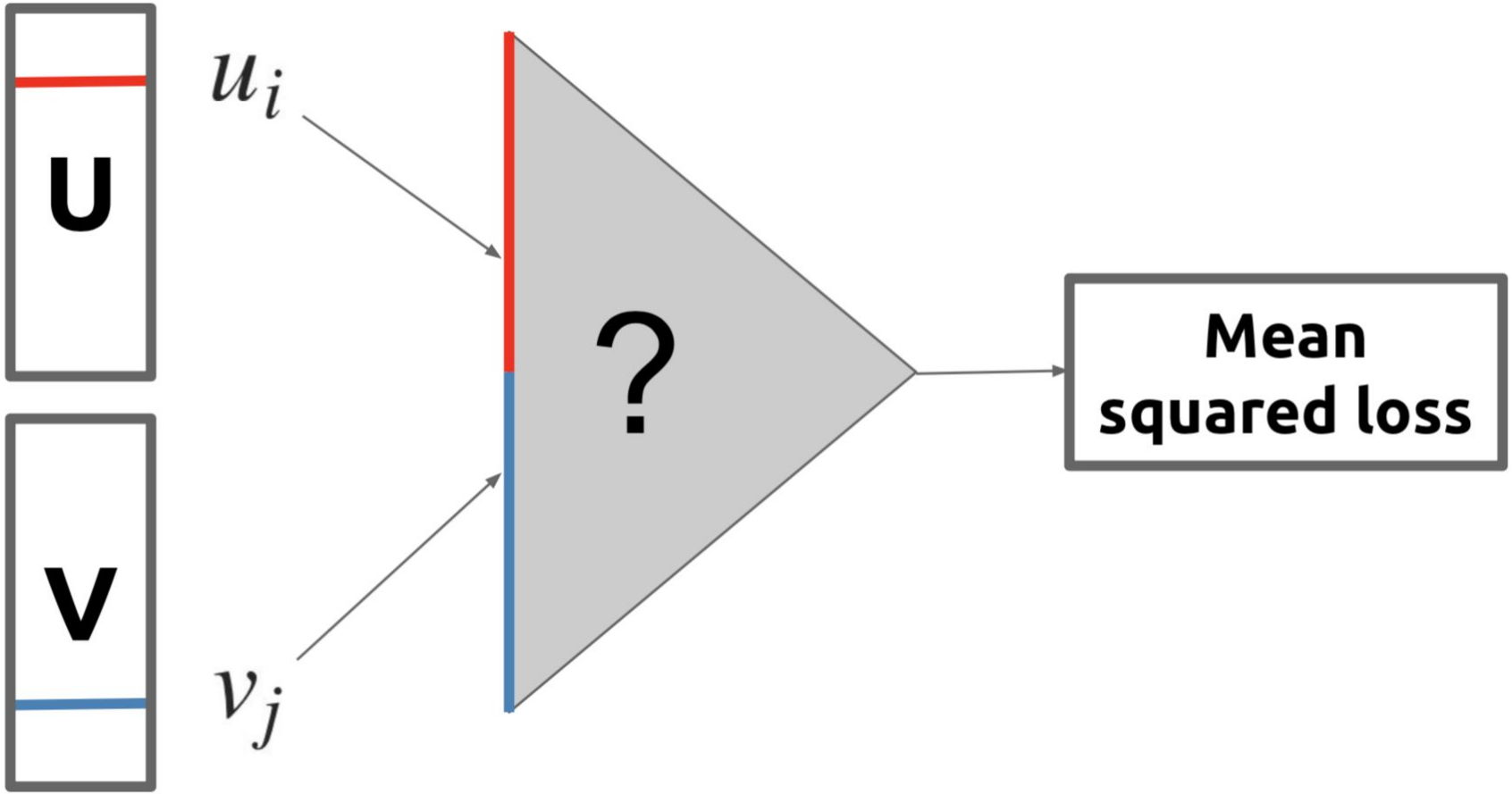


Deep Learning and Recommendations

- Heterogeneous data handled easily
- Sequential behavior modeling (RNNs)
- Feature extraction directly from the content
- More accurate (richer) representations of users and items

Learning Embeddings

- Embedding: a (learned) vector representing an entity
  - “latent feature vector”
  - similar users (or items) have similar embeddings
- Matrix Factorization can be viewed as a simplistic NN
  - hidden layer: item/user feature vector


U
V
u_i
v_j
?
Mean squared loss

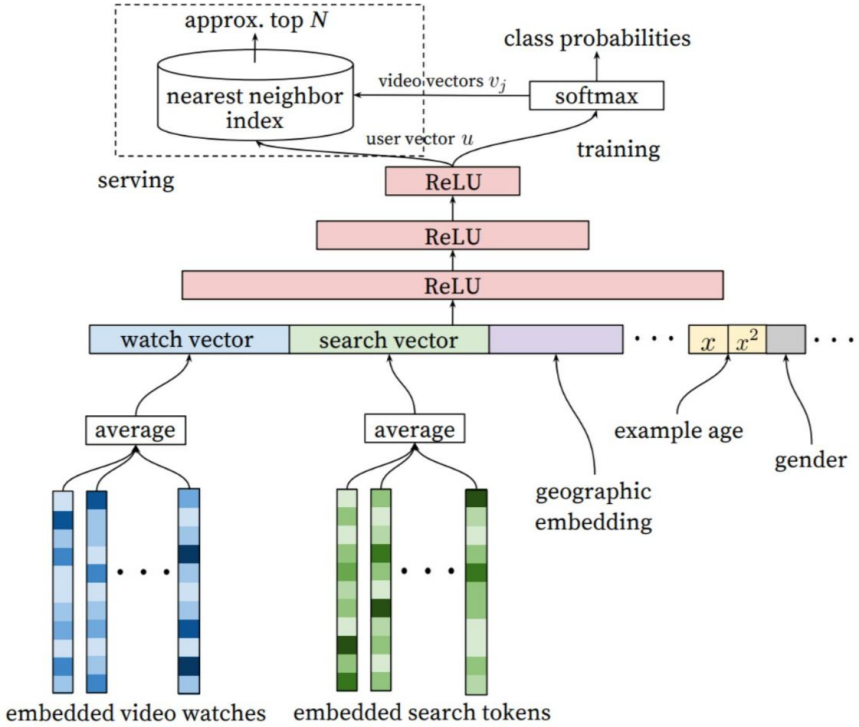
Learning Embeddings, MF vs NN

- Both are doing roughly the same thing: embeddings, MSE loss, gradient descent
- NN can explore combinations other than dot product
- … but require a very large model
- So not much benefit

YouTube Recommender

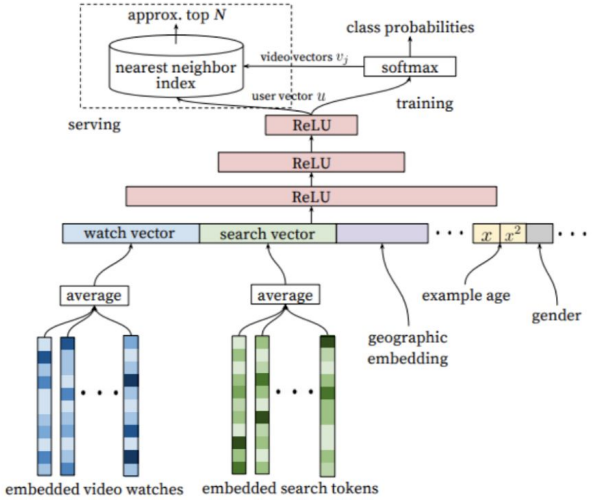
- Two stage ranker: candidate generation (shrinking set of items to rank) and ranking (classifying actual impressions)
- Two feed-forward, fully connected, networks with hundreds of features

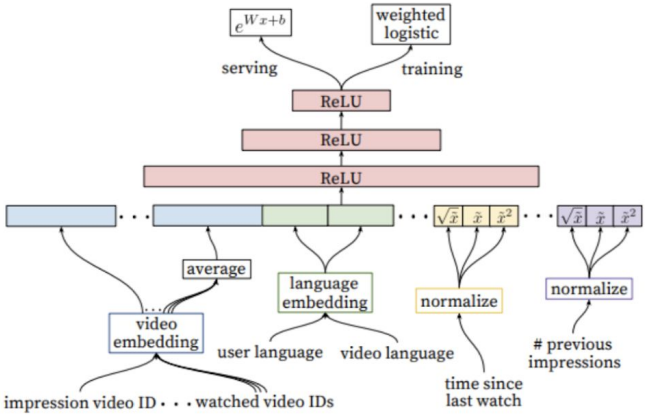
[Covington et. al., 2016]



YouTube Recommender [Covington et. al, 2016]

- Two networks
- Candidate generation
  - Recommendations as classification
    - Items clicked / not clicked when were recommended
  - Feedforward network on many features
    - Average watch embedding vector of user (last few items)
    - Average search embedding vector of user (last few searches)
    - User attributes
    - Geographic embedding
  - Negative item sampling + softmax
- Reranking
  - More features
    - Actual video embedding
    - Average video embedding of watched videos
    - Language information
    - Time since last watch
    - Etc.
  - Weighted logistic regression on the top of the network





Feature Extraction

- Images (CNNs)
  - Fashion, Video
- Text (RNNs)
  - News, Books, Publications
- Audio (CNNs and/or RNNs)
  - Music

Content-based feature learning

- Problem: new items have no usage history (cold start)
- Idea: use content-information to predict latent factors
- Successfully demonstrated with music
- Implementation: deep learning with CNNs

https://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf

Session-based Recommendations

- User is not logged-in
- Simple yet effective approach:
  - recommend items similar to whatever the user last clicked on
- Or try to use the session history (while ignoring order):
  - recommend items close to the average of the latent feature vectors of the items visited
- Recurrent Neural Networks

https://arxiv.org/pdf/1511.06939.pdf

Collaborative Topic Regression

- Collaborative Filtering (matrix factorization) + Topic Modeling (LDA)
- Best results come from jointly optimizing both models
- By combining topic modeling with latent features, the recommendations are more explainable

https://link.springer.com/article/10.1007/s10994-016-5599-z

Some open-source recommenders

- Matrix factorization via ALS on Spark
- Mahout on Spark – Ted Dunning's (MAPR) item-based based recommender with LLR (co-occurrence) as the similarity measure
- The Universal Recommender – https://github.com/pferrel/template-scala-parallel-universal-recommendation
- Implicit – https://github.com/benfred/implicit

Open Source (Deep Learning)

- Spotlight – https://maciejkula.github.io/spotlight/
- Amazon Deep Scalable Sparse Tensor Network Engine – https://github.com/amzn/amazon-dsstne

- https://github.com/robi56/Deep-Learning-for-Recommendation-Systems

Backup Slides

https://recsys.acm.org/challenges/

The ACM Conference Series on **Recommender Systems**

2017: job recommendations (cold start)
winning paper: http://www.cs.toronto.edu/~mvolkovs/recsys2017_challenge.pdf

2016: job recommendations (general)
winners: http://2016.recsyschallenge.com/pdf/slides-xiao.pdf

2015: e-commerce click stream
a solution: https://github.com/romovpa/ydf-recsys2015-challenge
winner: https://www.slideshare.net/romovpa/recsys-challenge-2015-ensemble-learning-with-categorical-features

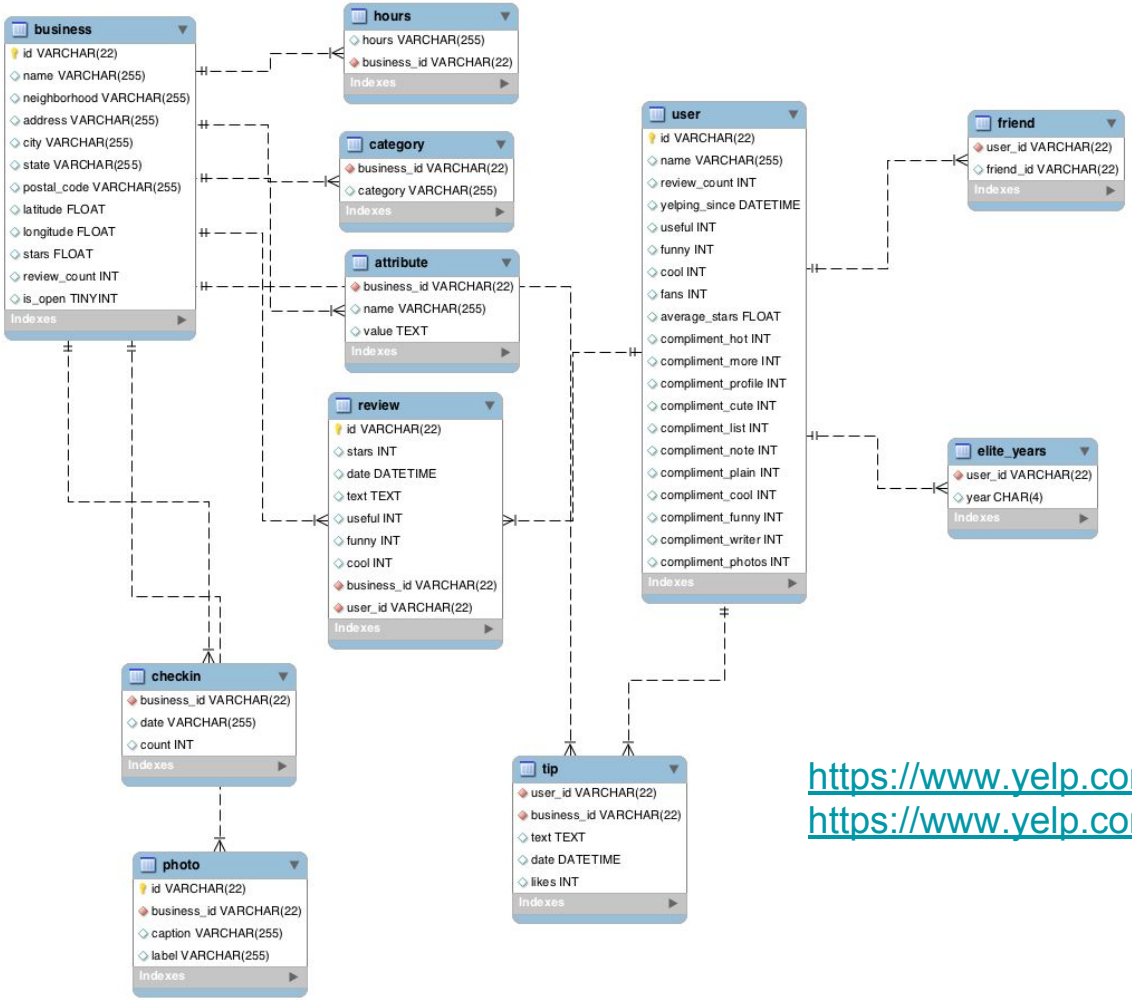
2014: optimize user engagement
winner: https://hal.inria.fr/hal-01077986/document

2011: contextual information
http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=16477
winner: https://arxiv.org/pdf/1207.6379.pdf

- **Adaptive, Personalized Diversity for Visual Discovery (Amazon)**
- **Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text (Stanford)**
- **Trust-aware Recommender Systems (IRST/FBK, Simon Fraser)**
- **Recommender Systems for Self-Actualization (Clemson)**
- **Deep Learning in Modern Recommendation Systems (Contata Solutions)**

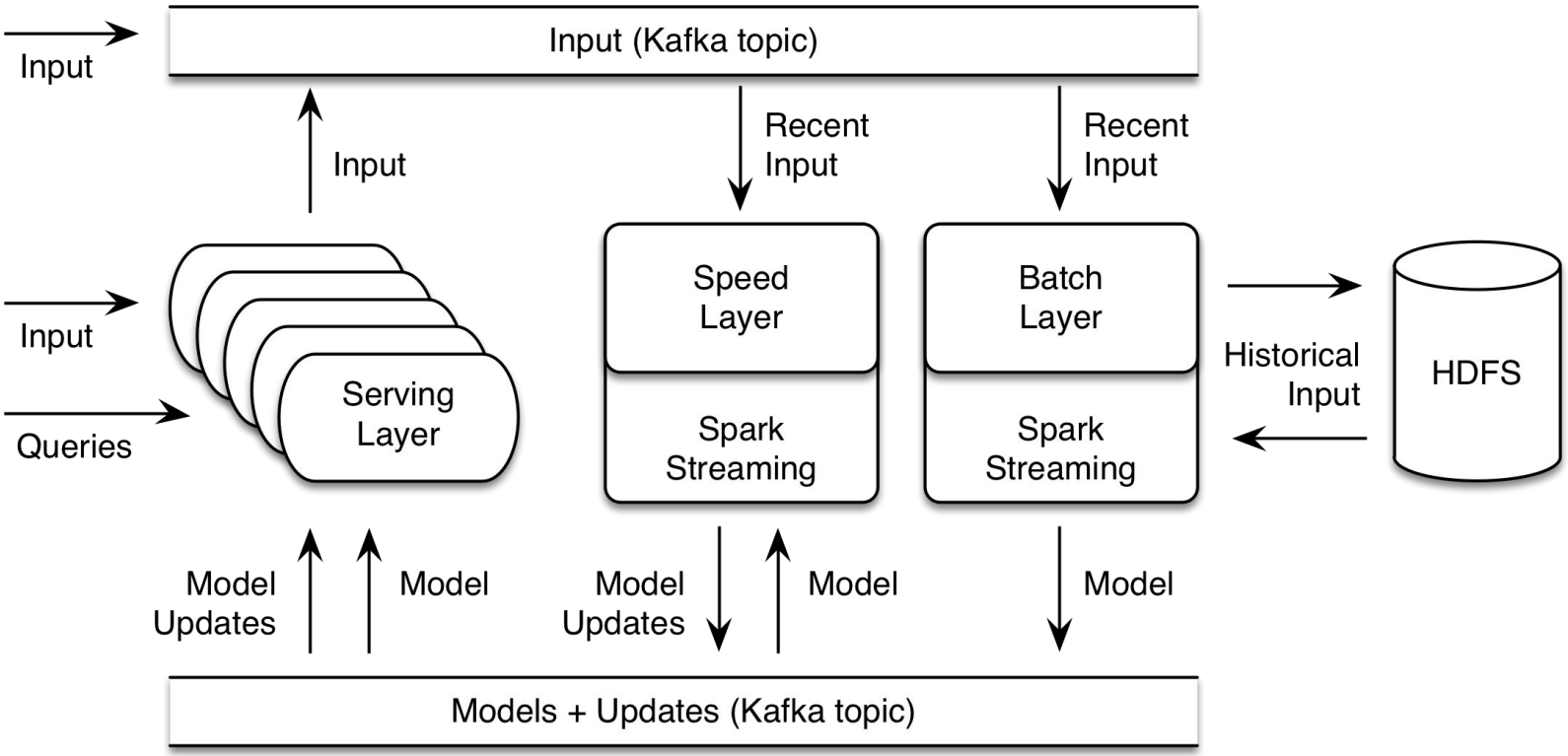
YELP Dataset Challenge

https://www.yelp.com/dataset_challenge/



https://www.yelp.com/dataset/documentation/sql
https://www.yelp.com/dataset/documentation/json

Oryx 2


Input
Input (Kafka topic)
Input
Recent Input
Recent Input
Input
Queries
Serving Layer
Speed Layer
Spark Streaming
Batch Layer
Spark Streaming
Historical Input
HDFS
Model Updates
Model
Model Updates
Model
Model
Models + Updates (Kafka topic)

- How will you generate recommendations?
  - How much of the data you have will you use?
  - What additional data do you wish you had?
  - Will it scale?
  - How often will you update the model?
- What pitfalls do you see?
- How will you evaluate the results?

https://www.theguardian.com/technology/2018/feb/02/youtube-algorithm-election-clinton-trump-guillaume-chaslot

https://algotransparency.org

https://algorithms-tour.stitchfix.com/

https://multithreaded.stitchfix.com/blog/2018/06/28/latent-style/


THE SELFISH LEDGER

http://time.com/collection-post/3950525/facebook-news-feed-algorithm/

https://blog.hootsuite.com/facebook-algorithm/