recommender systems

Copyright © 2018-2021 Sundog Software LLC, DBA Sundog Education
All rights reserved worldwide.
Stock images licensed via Getty Images / iStockPhoto.com

recommender systems
getting set up

- install anaconda
- install scikit-surprise
- download course materials

let’s do this.

sundog-education.com/RecSys

setup walkthrough

course overview

- getting started
- intro to python
- evaluating recommender systems
- building a recommendation engine
- content-based filtering
- neighborhood-based collaborative filtering
- model-based methods
- intro to deep learning
- recommendations with deep learning
- scaling it up
- challenges of recommender systems
- case studies
- hybrid solutions
- more to explore

optional sections

- intro to python
- intro to deep learning

what is a recommender system

what it is not

A recommender system is NOT a system that “recommends” arbitrary values.

That describes machine learning in general.

for example

A system that “recommends” prices for a house you’re selling is **NOT** a recommender system.

A system that “recommends” whether a transaction is fraudulent is **NOT** a recommender system.

These are general *machine learning problems*, where you’d apply techniques such as
Regression, deep learning, XGBoost, or other techniques.

If that’s what you’re looking for, you want a more general machine learning course.

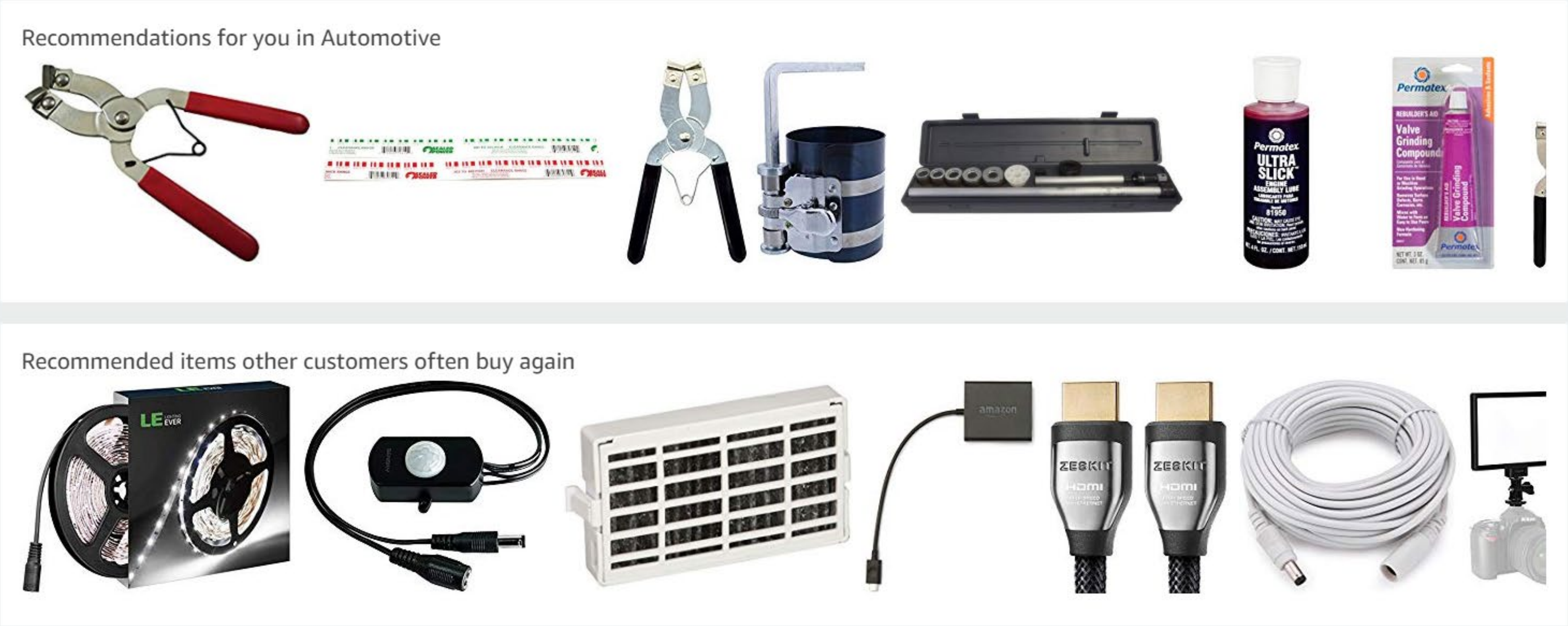
what it is

A system that predicts ratings or preferences a user might give to an item

Often these are sorted and presented as “top-N” recommendations

Also known as recommender engines, recommendation systems, recommendation platforms.

this is a recommender engine



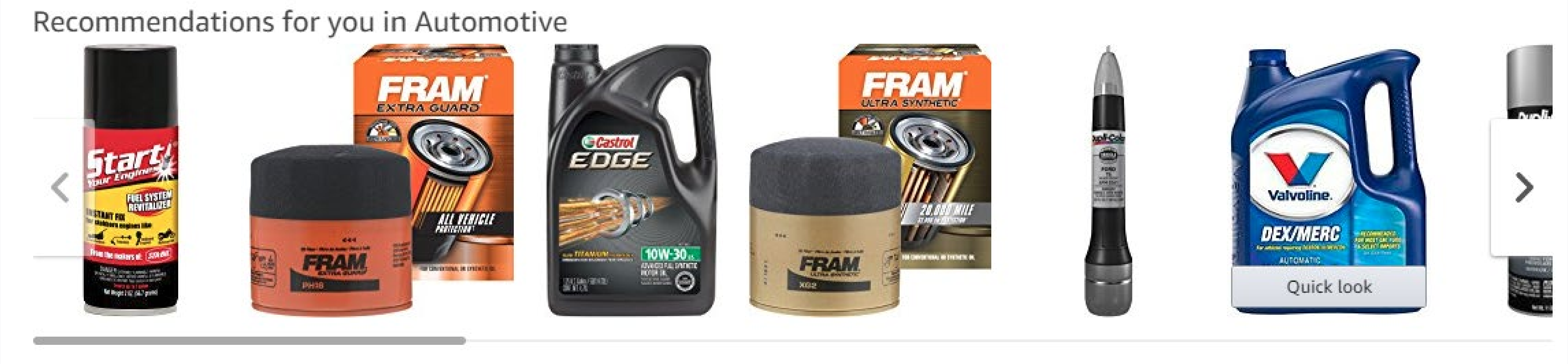
Recommendations for you in Automotive

Recommended items other customers often buy again

Permatex
ULTRA SLICK
Valve Grinding Compound
LE
amazon
ZESKIT
HDMI

many flavors of recommenders

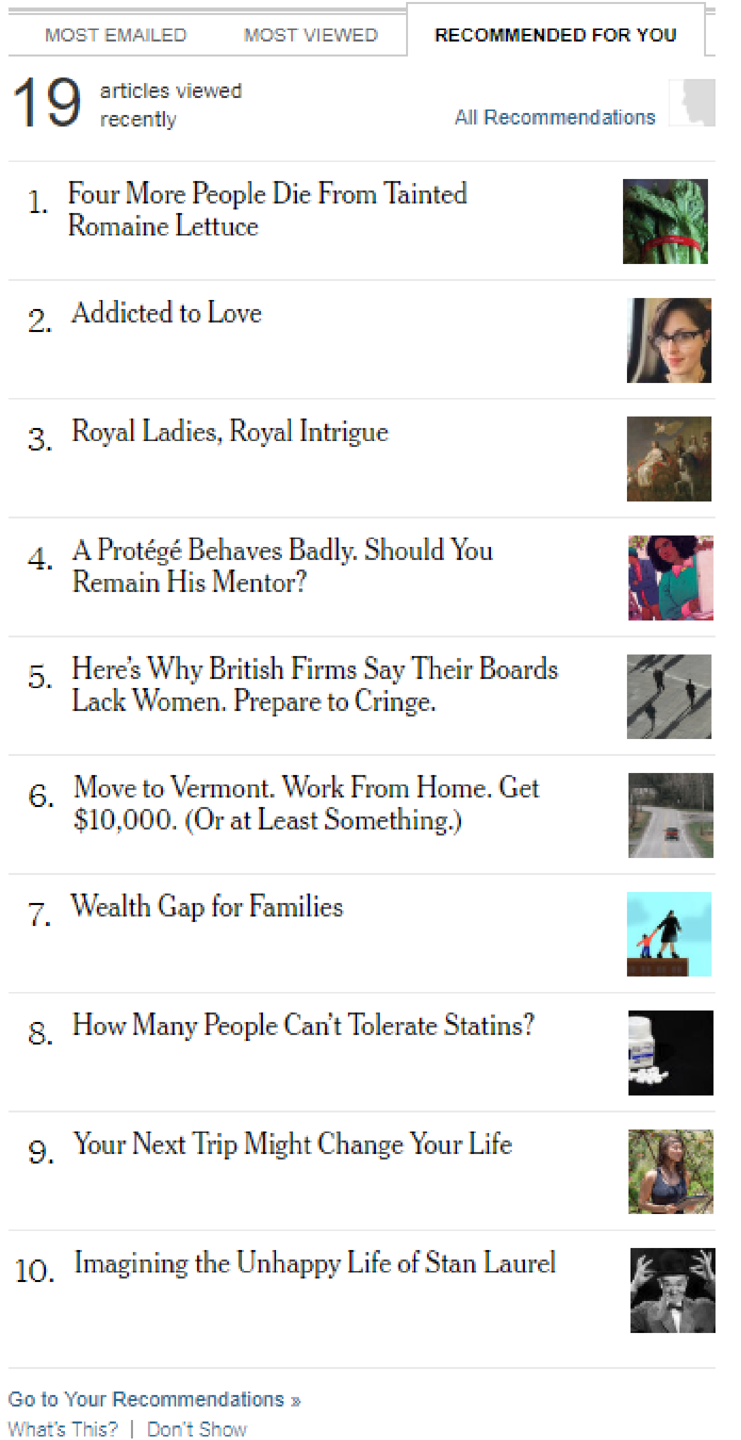
recommending things



Recommendations for you in Automotive

Start Your Engine
FUEL SYSTEM REVITALIZER
FRAM EXTRA GUARD
ALL VEHICLE PROTECTION
FRAM
PH16
Castrol EDGE
10W-30
FRAM ULTRA SYNTHETIC
FRAM
XG2
Valvoline
DEX/MERC
AUTOMATIC
Quick look

recommending content



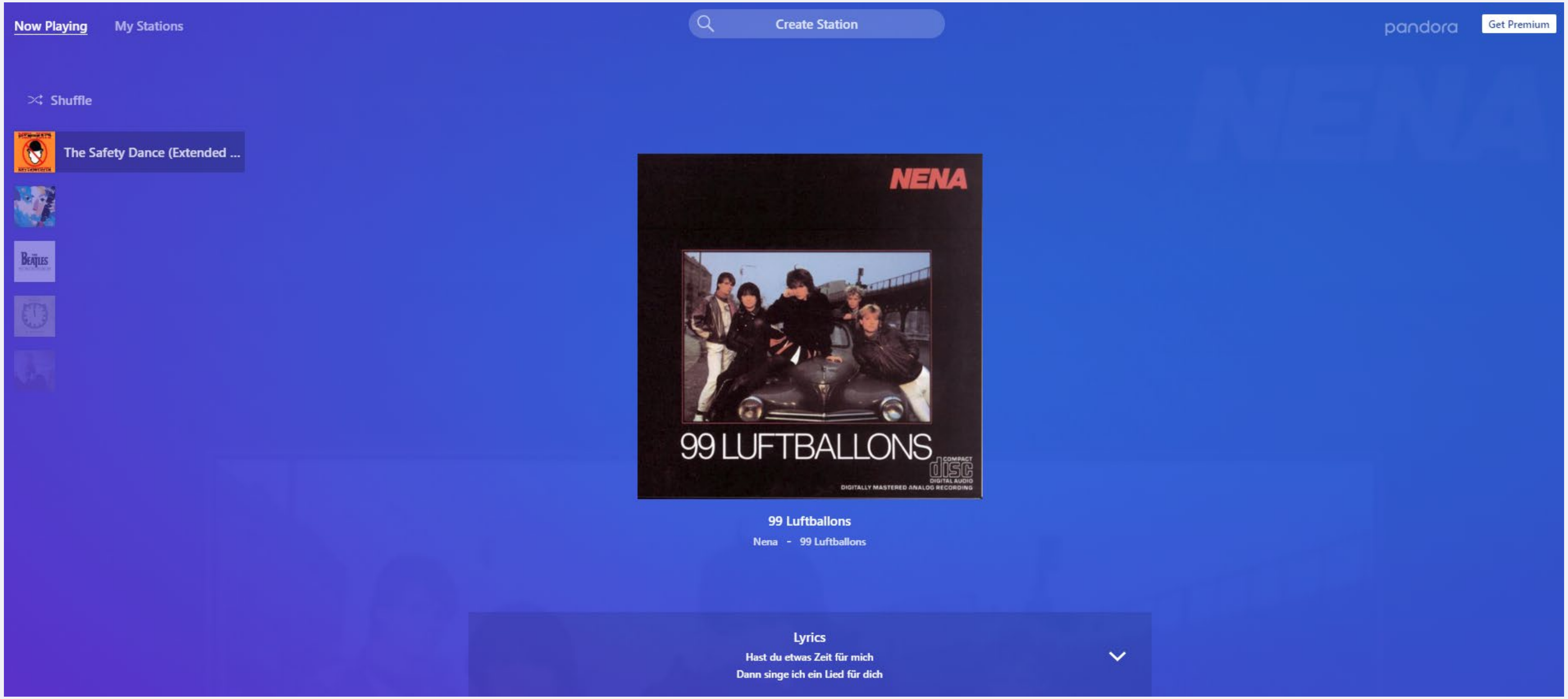
19 articles viewed recently

All Recommendations

1. Four More People Die From Tainted Romaine Lettuce
2. Addicted to Love
3. Royal Ladies, Royal Intrigue
4. A Protégé Behaves Badly. Should You Remain His Mentor?
5. Here's Why British Firms Say Their Boards Lack Women. Prepare to Cringe.
6. Move to Vermont. Work From Home. Get $10,000. (Or at Least Something.)
7. Wealth Gap for Families
8. How Many People Can't Tolerate Statins?
9. Your Next Trip Might Change Your Life
10. Imagining the Unhappy Life of Stan Laurel

Go to Your Recommendations »
What's This? | Don't Show

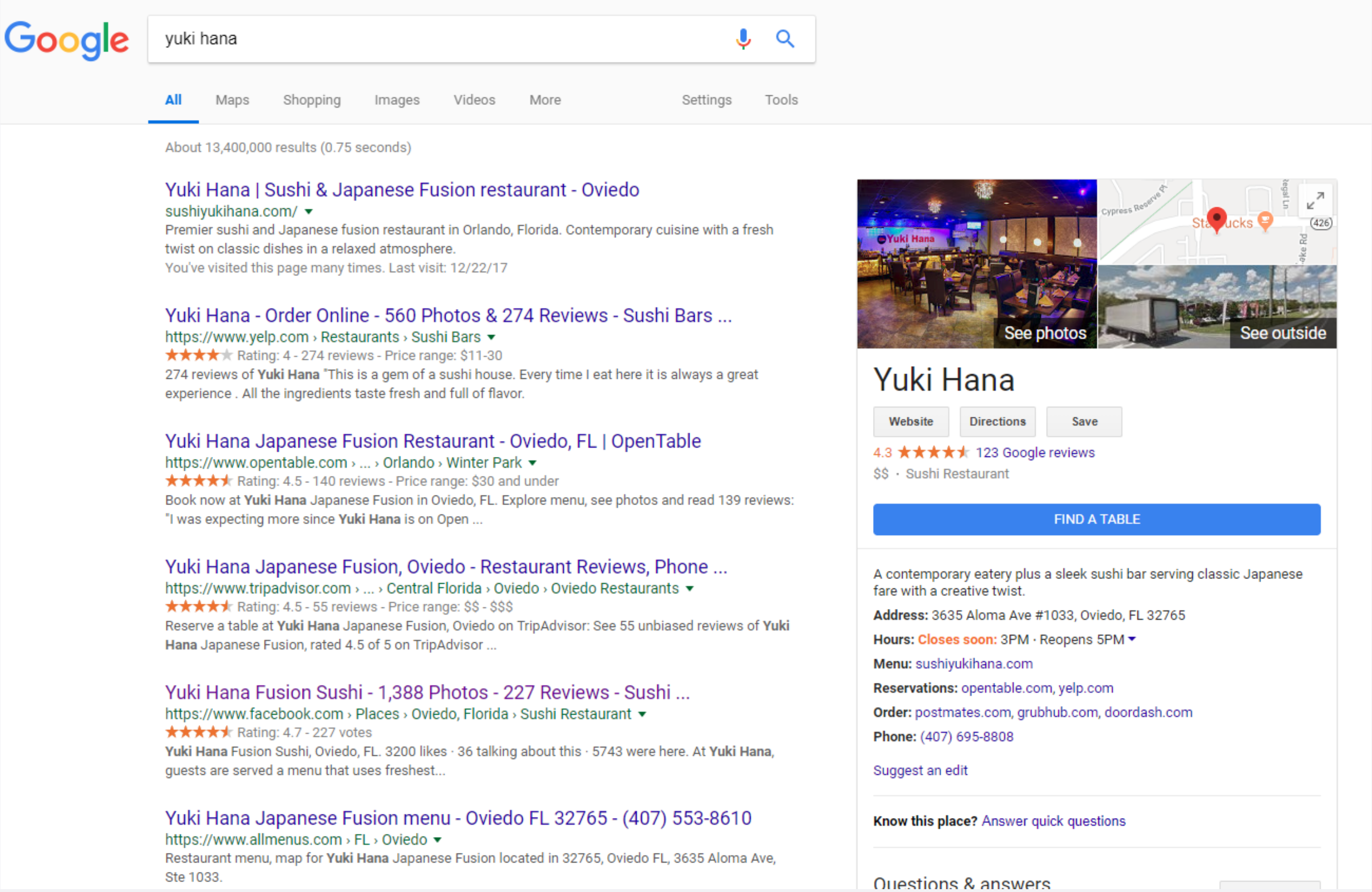
recommending music


Now Playing
My Stations
Create Station
pandora
Get Premium
Shuffle
The Safety Dance (Extended ...
NENA
99 LUFTBALLONS
99 Luftballons
Nena - 99 Luftballons
Lyrics
Hast du etwas Zeit für mich
Dann singe ich ein Lied für dich

recommending people

Online Dating
GO!

recommending
search results



About 13,400,000 results (0.75 seconds)

Yuki Hana | Sushi & Japanese Fusion restaurant - Oviedo
sushiyukihana.com/ ▾
Premier sushi and Japanese fusion restaurant in Orlando, Florida. Contemporary cuisine with a fresh twist on classic dishes in a relaxed atmosphere.
You've visited this page many times. Last visit: 12/22/17

Yuki Hana - Order Online - 560 Photos & 274 Reviews - Sushi Bars ...
https://www.yelp.com › Restaurants › Sushi Bars ▾
★★★★☆ Rating: 4 - 274 reviews - Price range: $11-30
274 reviews of **Yuki Hana** "This is a gem of a sushi house. Every time I eat here it is always a great experience . All the ingredients taste fresh and full of flavor.

Yuki Hana Japanese Fusion Restaurant - Oviedo, FL | OpenTable
https://www.opentable.com › ... › Orlando › Winter Park ▾
★★★★★ Rating: 4.5 - 140 reviews - Price range: $30 and under
Book now at **Yuki Hana** Japanese Fusion in Oviedo, FL. Explore menu, see photos and read 139 reviews: "I was expecting more since **Yuki Hana** is on Open ...

Yuki Hana Japanese Fusion, Oviedo - Restaurant Reviews, Phone ...
https://www.tripadvisor.com › ... › Central Florida › Oviedo › Oviedo Restaurants ▾
★★★★★ Rating: 4.5 - 55 reviews - Price range: $$ - $$$
Reserve a table at **Yuki Hana** Japanese Fusion, Oviedo on TripAdvisor: See 55 unbiased reviews of **Yuki Hana** Japanese Fusion, rated 4.5 of 5 on TripAdvisor ...

Yuki Hana Fusion Sushi - 1,388 Photos - 227 Reviews - Sushi ...
https://www.facebook.com › Places › Oviedo, Florida › Sushi Restaurant ▾
★★★★★ Rating: 4.7 - 227 votes
Yuki Hana Fusion Sushi, Oviedo, FL. 3200 likes · 36 talking about this · 5743 were here. At **Yuki Hana**, guests are served a menu that uses freshest...

Yuki Hana Japanese Fusion menu - Oviedo FL 32765 - (407) 553-8610
https://www.allmenus.com › FL › Oviedo ▾
Restaurant menu, map for **Yuki Hana** Japanese Fusion located in 32765, Oviedo FL, 3635 Aloma Ave, Ste 1033.

Yuki Hana

4.3 ★★★★★ 123 Google reviews
$$ · Sushi Restaurant

A contemporary eatery plus a sleek sushi bar serving classic Japanese fare with a creative twist.

Address: 3635 Aloma Ave #1033, Oviedo, FL 32765
Hours: Closes soon: 3PM · Reopens 5PM ▾
Menu: sushiyukihana.com
Reservations: opentable.com, yelp.com
Order: postmates.com, grubhub.com, doordash.com
Phone: (407) 695-8808

Suggest an edit

Know this place? Answer quick questions

Questions & answers

understanding you

understanding you… explicitly

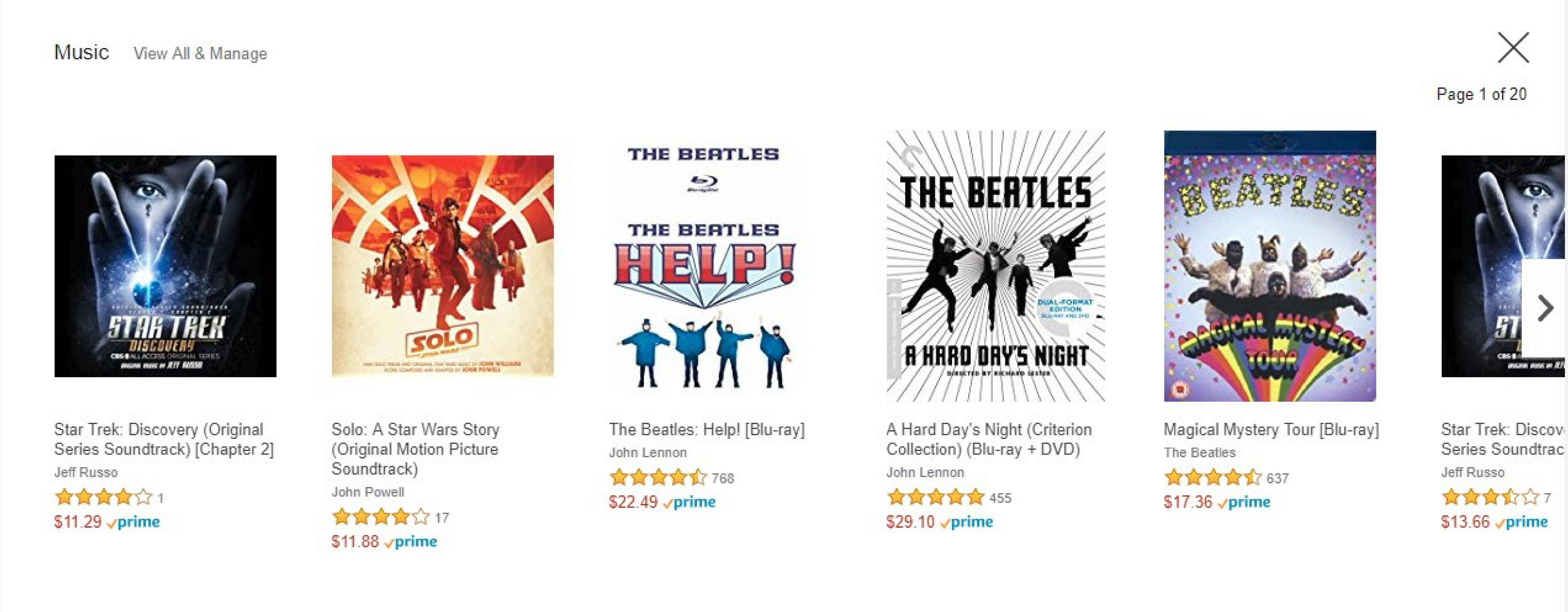
understanding you... implicitly

things you click on

things you purchase

things you consume

top-N recommenders



Music View All & Manage

Page 1 of 20

Star Trek: Discovery (Original Series Soundtrack) [Chapter 2]
Jeff Russo
★★★★☆ 1
$11.29 ✓prime

Solo: A Star Wars Story (Original Motion Picture Soundtrack)
John Powell
★★★★☆ 17
$11.88 ✓prime

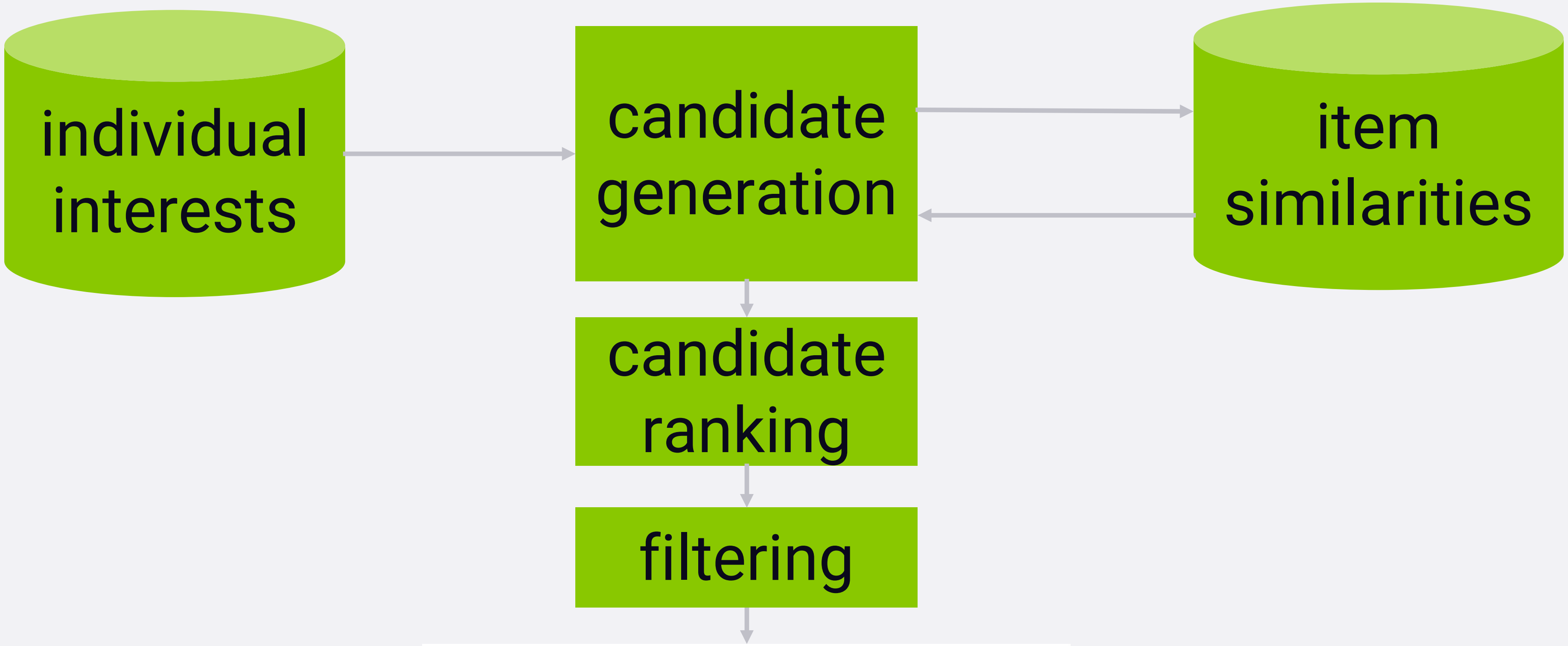
The Beatles: Help! [Blu-ray]
John Lennon
★★★★⯪ 768
$22.49 ✓prime

A Hard Day's Night (Criterion Collection) (Blu-ray + DVD)
John Lennon
★★★★★ 455
$29.10 ✓prime

Magical Mystery Tour [Blu-ray]
The Beatles
★★★★⯪ 637
$17.36 ✓prime

Star Trek: Discov
Series Soundtrac
Jeff Russo
★★★⯪☆ 7
$13.66 ✓prime

(one) anatomy of a top-N recommender

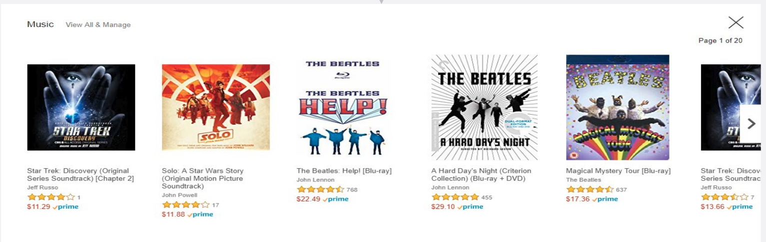

individual interests

candidate generation

item similarities

candidate ranking

filtering


Music View All & Manage

Page 1 of 20

Star Trek: Discovery (Original Series Soundtrack) [Chapter 2]
Jeff Russo
★★★★☆ 1
$11.29 ✓prime

Solo: A Star Wars Story (Original Motion Picture Soundtrack)
John Powell
★★★★☆ 17
$11.88 ✓prime

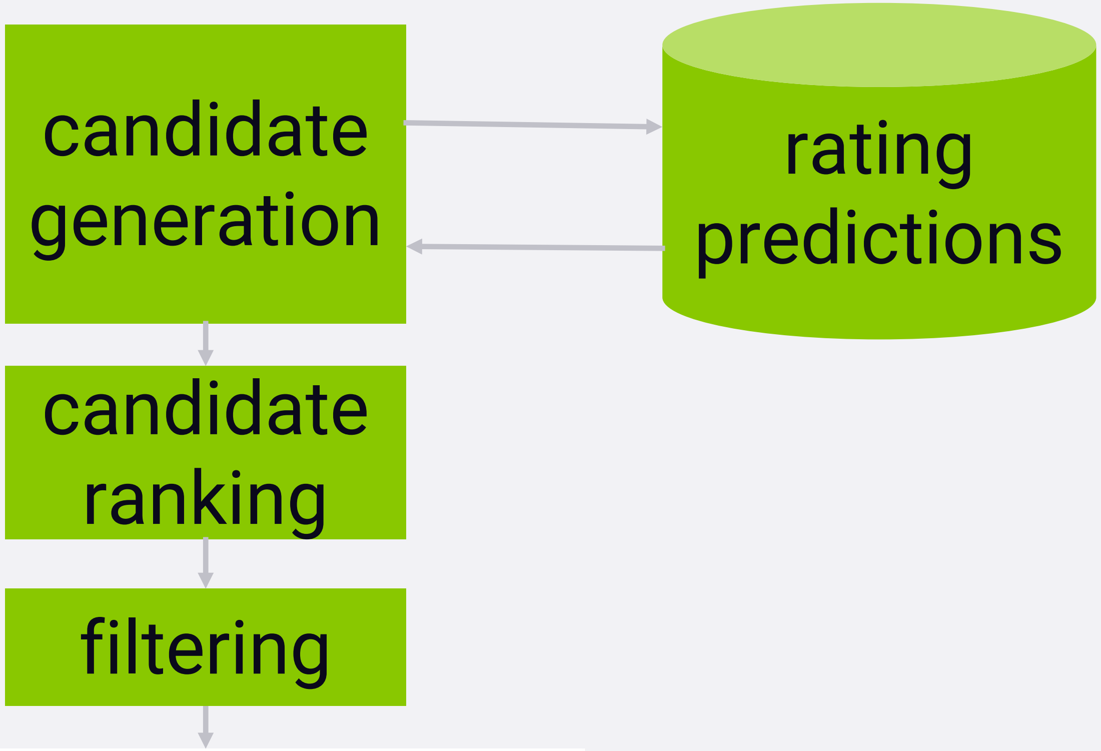
The Beatles: Help! [Blu-ray]
John Lennon
★★★★☆ 768
$22.49 ✓prime

A Hard Day's Night (Criterion Collection) (Blu-ray + DVD)
John Lennon
★★★★★ 455
$29.10 ✓prime

Magical Mystery Tour [Blu-ray]
The Beatles
★★★★☆ 637
$17.36 ✓prime

Star Trek: Discov
Series Soundtrac
Jeff Russo
★★★☆☆ 7
$13.66 ✓prime

another way to do it



candidate generation

rating predictions

candidate ranking

filtering

Music
View All & Manage
Page 1 of 20
Star Trek: Discovery (Original Series Soundtrack) [Chapter 2]
Jeff Russo
☆☆☆☆☆ 1
$11.29 ✓prime
Solo: A Star Wars Story (Original Motion Picture Soundtrack)
John Powell
☆☆☆☆☆ 17
$11.88 ✓prime
The Beatles: Help! [Blu-ray]
John Lennon
☆☆☆☆☆ 768
$22.49 ✓prime
A Hard Day's Night (Criterion Collection) (Blu-ray + DVD)
John Lennon
☆☆☆☆☆ 455
$29.10 ✓prime
Magical Mystery Tour [Blu-ray]
The Beatles
☆☆☆☆☆ 637
$17.36 ✓prime
Star Trek: Discov
Series Soundtrac
Jeff Russo
☆☆☆☆☆ 7
$13.66 ✓prime

quiz time

which of the following are examples of implicit ratings?

- star reviews
- purchase data
- video viewing data
- click data

which of the following are examples of implicit ratings?

- star reviews
- purchase data
- video viewing data
- click data

which are examples of recommender systems?

- netflix’s home page
- google search
- amazon’s “people who bought also bought…”
- pandora
- online radio stations
- youtube
- wikipedia search

02

which are examples of recommender systems?

- netflix’s home page
- google search
- amazon’s “people who bought also bought…”
- pandora
- online radio stations
- youtube
- wikipedia search

02

03

- netflix recommendation widgets
- google search
- amazon “people who bought also bought”

which are examples of “Top-N” recommenders?

03

- netflix recommendation widgets
- google search
- amazon “people who bought also bought”

which are examples of “Top-N” recommenders?

04

- candidate generation
- filtering
- candidate shuffling
- ranking

which are components of a top-N recommender?

04

- candidate generation
- filtering
- candidate shuffling
- ranking

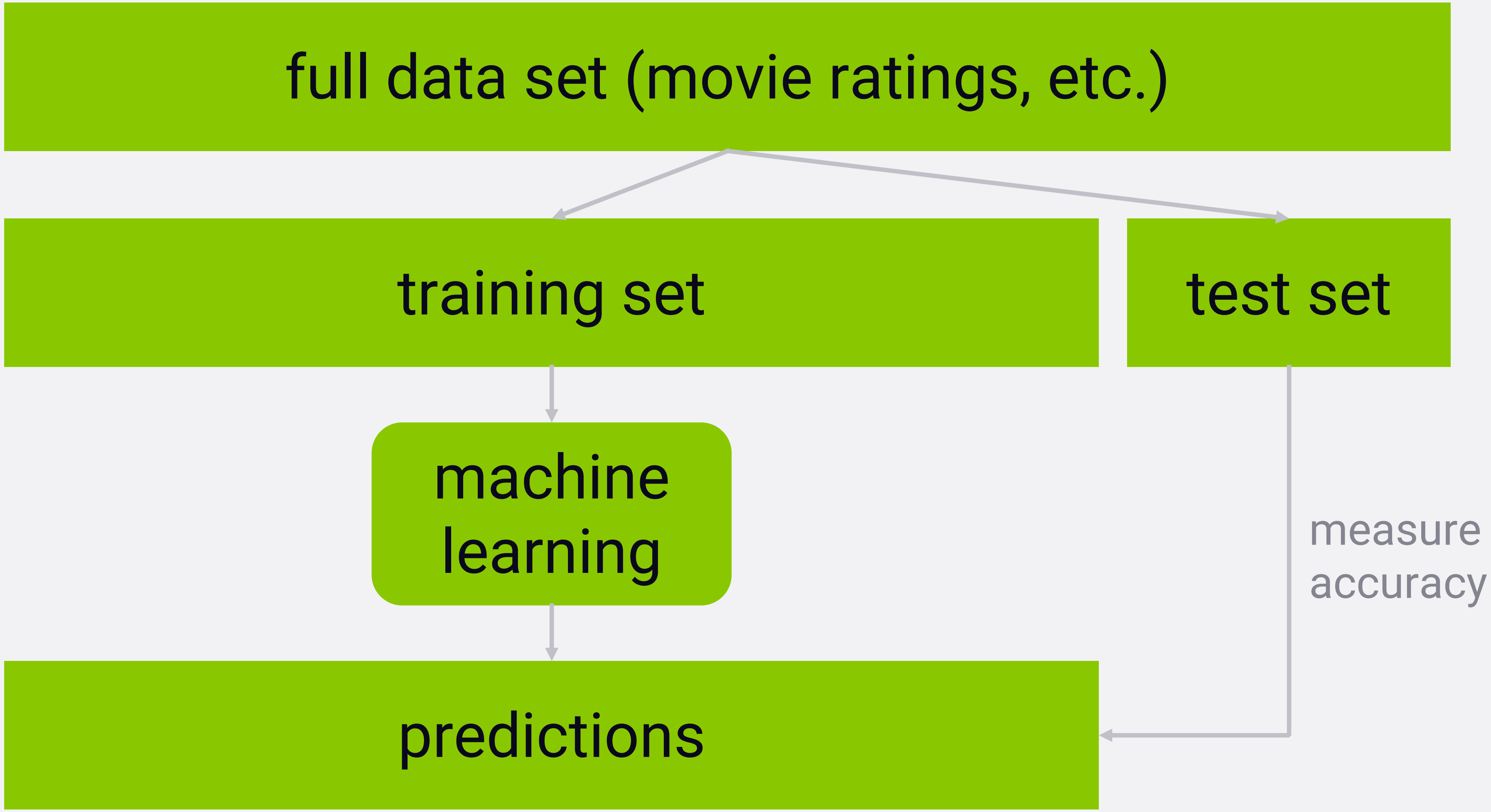
which are components of a top-N recommender?

intro to python

code walkthrough

evaluating recommender systems

train/test



full data set (movie ratings, etc.)

training set

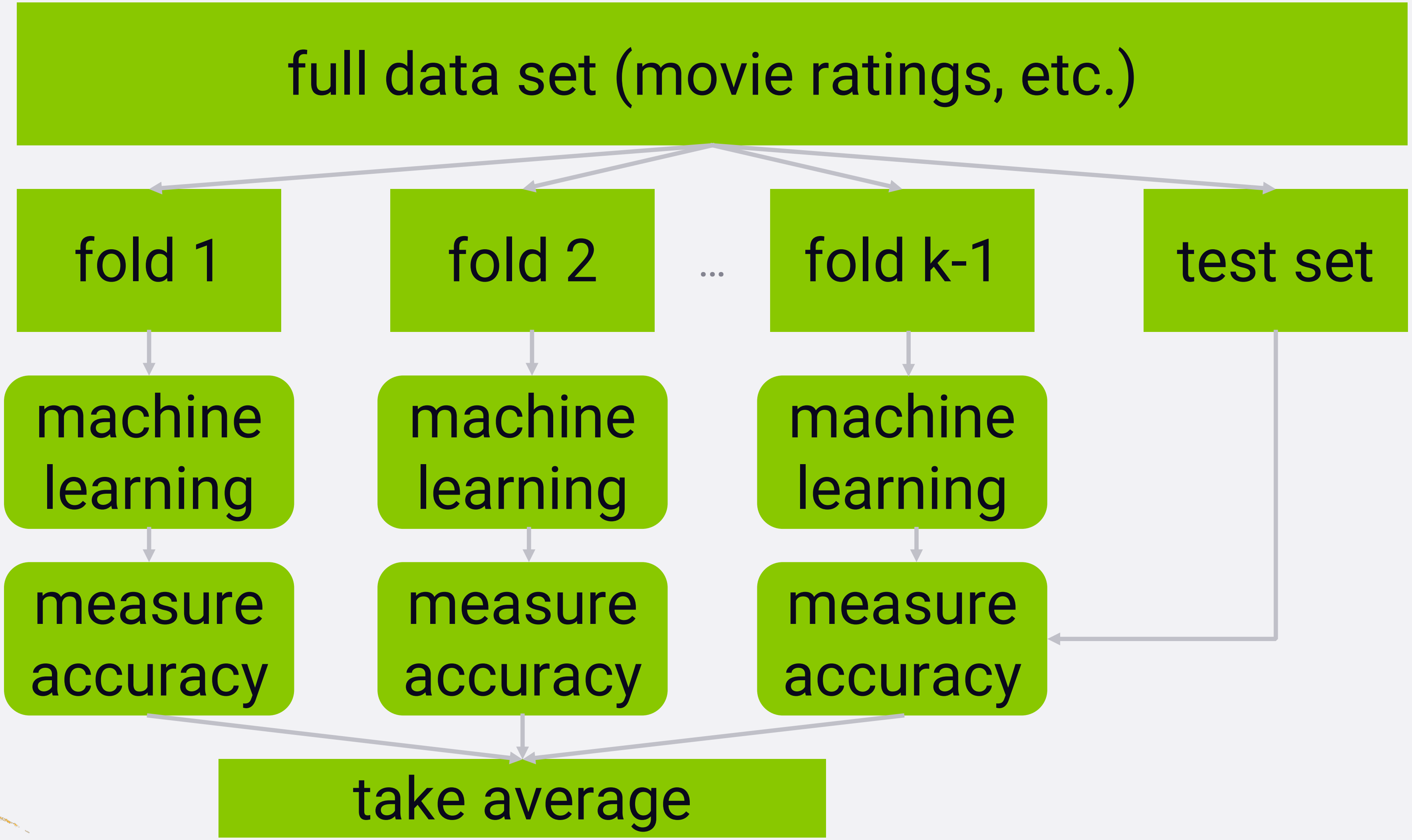
test set

machine learning

measure accuracy

predictions

k-fold cross-validation



full data set (movie ratings, etc.)

fold 1

fold 2

…

fold k-1

test set

machine learning

machine learning

machine learning

measure accuracy

measure accuracy

measure accuracy

take average

measuring accuracy

mean absolute error (MAE)

$$\frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

predicted rating	actual rating	error
5	3	2
4	1	3
5	4	1
1	1	0

MAE = (2+3+1+0)/4 = 1.5

root mean square error (RMSE)

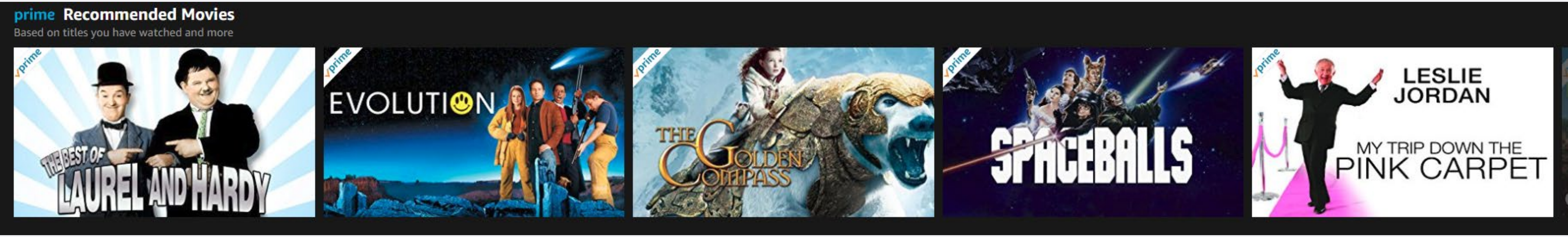
$$\sqrt{\frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n}}$$

predicted rating	actual rating	$error^2$
5	3	4
4	1	9
5	4	1
1	1	0

RMSE= $\sqrt{(4+9+1+0)/4}$ = 1.87

how did we get here?

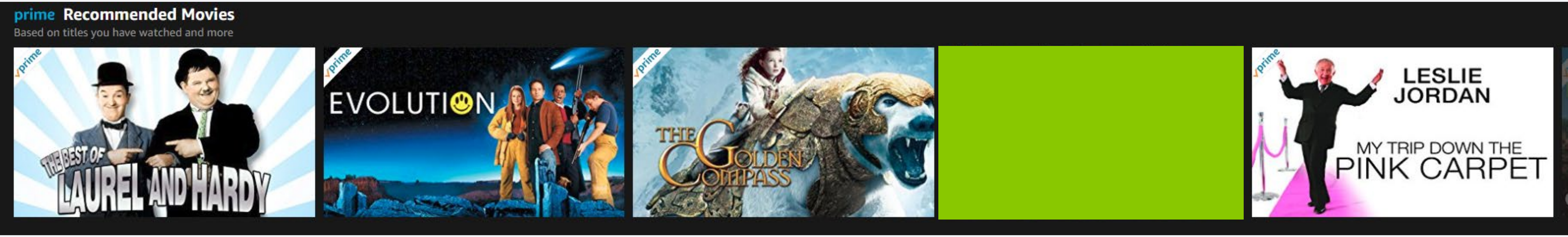
evaluating top-n recommenders


prime Recommended Movies
Based on titles you have watched and more
prime
THE BEST OF LAUREL AND HARDY
EVOLUTION
THE GOLDEN COMPASS
SPACEBALLS
LESLIE JORDAN
MY TRIP DOWN THE PINK CARPET

hit rate

$$\frac{hits}{users}$$

leave-one-out cross validation



prime Recommended Movies

Based on titles you have watched and more

prime
THE BEST OF LAUREL AND HARDY
EVOLUTION
THE GOLDEN COMPASS
LESLIE JORDAN
MY TRIP DOWN THE PINK CARPET

average reciprocal hit rate (ARHR)

$$\frac{\sum_{i=1}^{n} \frac{1}{rank_i}}{Users}$$

rank	reciprocal rank
3	1/3
2	1/2
1	1

cumulative hit rate (cHR)

hit rank	predicted (or actual) rating
4	5.0
~~2~~	~~3.0~~
1	5.0
~~10~~	~~2.0~~

rating hit rate (rHR)

rating	hit rate
5.0	0.001
4.0	0.004
3.0	0.030
2.0	0.001
1.0	0.0005

coverage

% of <user, item> pairs that can be predicted

diversity

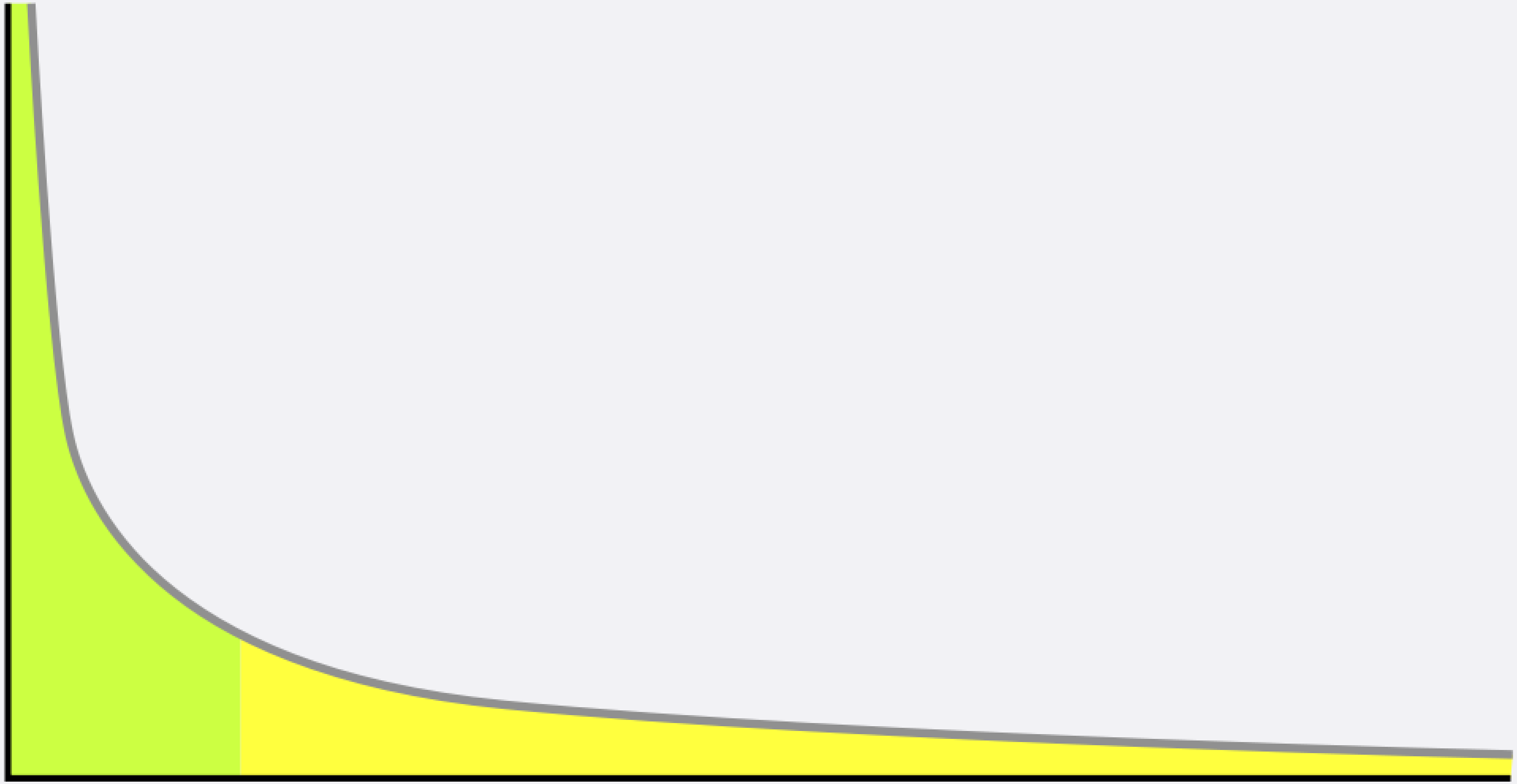
$$(1 - S)$$

S = avg similarity between recommendation pairs

novelty

mean popularity rank of recommended items

the long tail



churn

how often do recommendations change?

responsiveness

how quickly does new user behavior influence your recommendations?

what's important?

online A/B tests!

perceived quality

quiz time

which metric was used to evaluate the netflix prize?

01

which metric was used to evaluate the netflix prize?

root mean squared error (RMSE)

what’s a metric for top-n recommenders that accounts for the rank of predicted items?

02

02

what’s a metric for top-n recommenders that accounts for the rank of predicted items?

average reciprocal hit rank

03

which metric measures how popular or obscure your recommendations are?

03

novelty

which metric measures how popular or obscure your recommendations are?

04

which metric would tell us if we’re recommending the same types of things all the time?

04

diversity

which metric would tell us if we’re recommending the same types of things all the time?

which metric matters more than anything?

05

which metric matters more than anything?

the results of online a/b tests

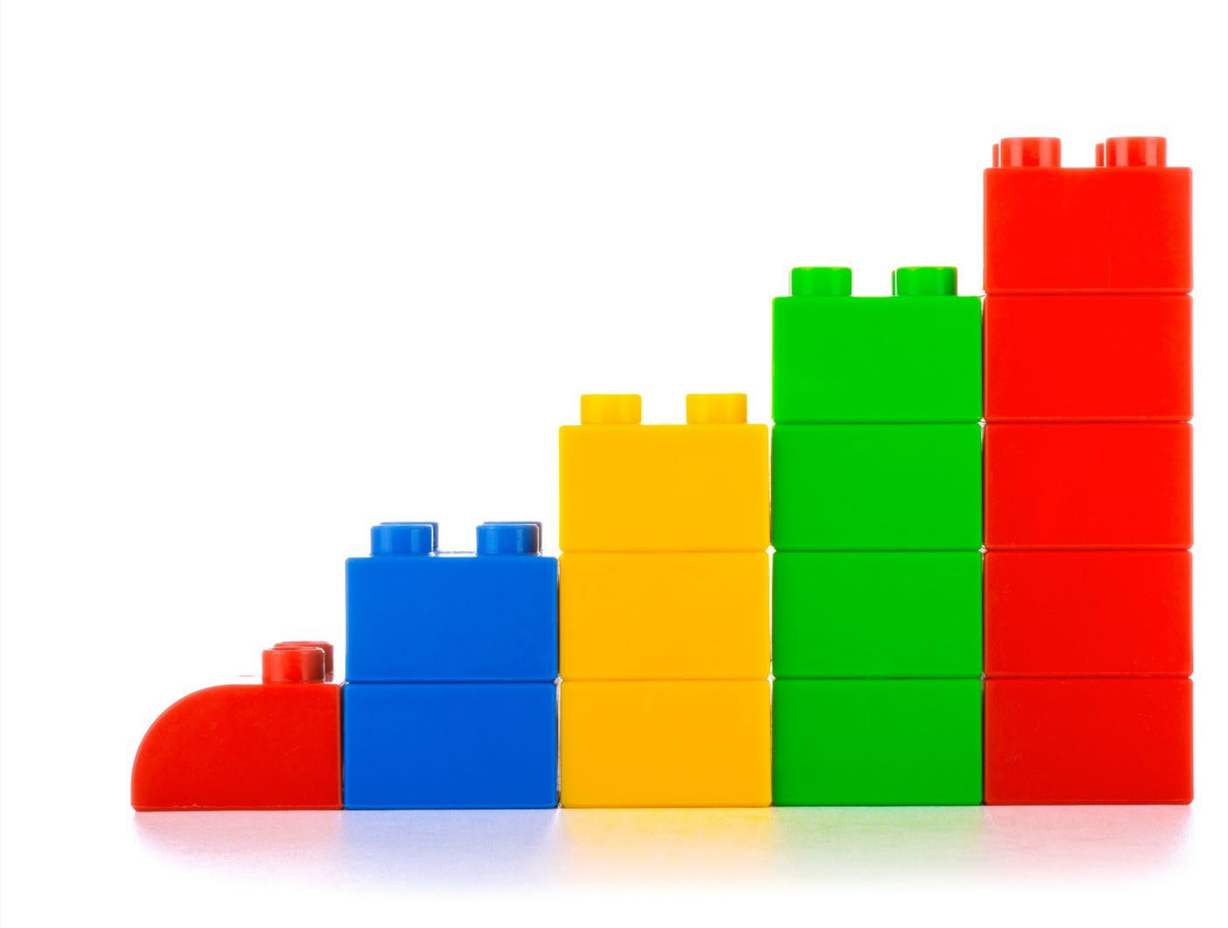
05

code walkthrough

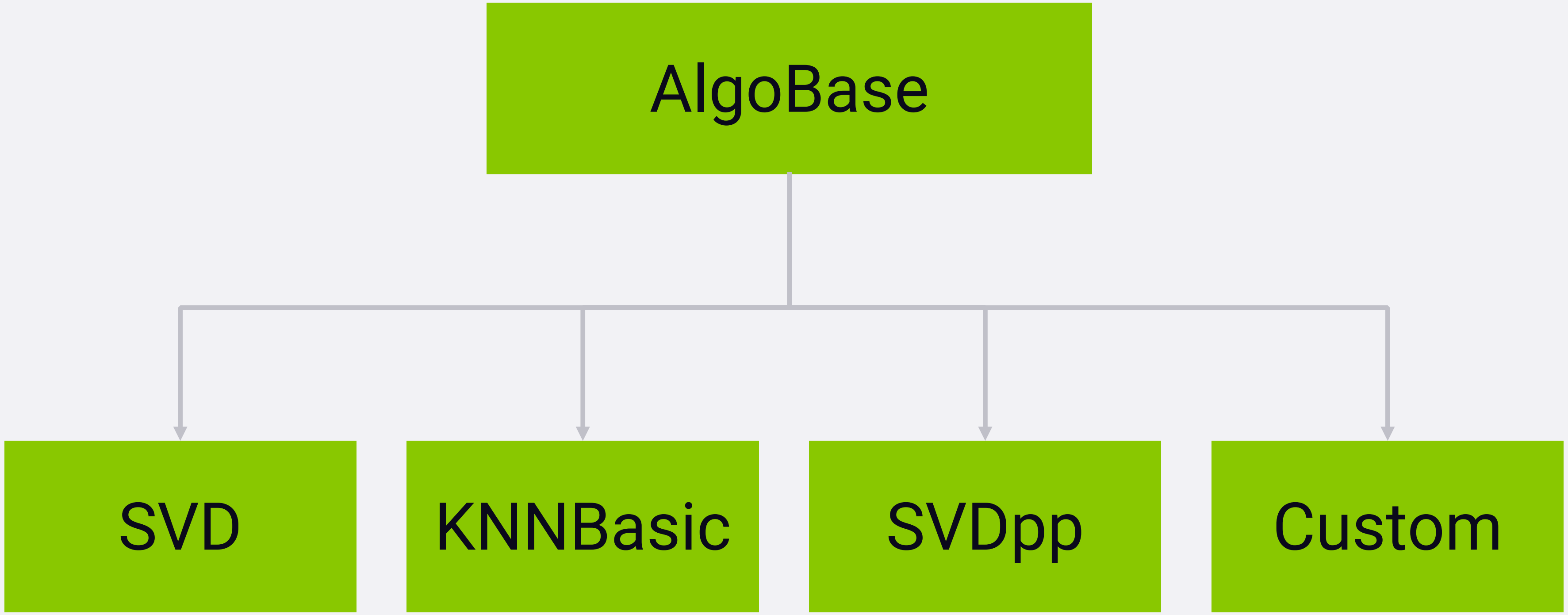
code walkthrough

code walkthrough

building a
recommender engine



surpriselib algorithm base class



AlgoBase

SVD

KNNBasic

SVDpp

Custom

creating a custom algorithm

implement an estimate function

```
class MyOwnAlgorithm(AlgoBase):

    def __init__(self):
        AlgoBase.__init__(self)

    def estimate(self, user, item):
        return 3
```

building on top of surpriselib

EvaluatedAlgorithm(AlgoBase)

algorithm: AlgoBase
Evaluate(EvaluationData)

EvaluationData(Dataset)

GetTrainSet()
GetTestSet()
…

RecommenderMetrics

algorithm bake-offs

Evaluator(DataSet)

AddAlgorithm(algorithm)
Evaluate()

dataset: EvaluatedDataSet
algorithms: EvaluatedAlgorithm[]

it’s just this easy

```
# Load up common data set for the recommender algorithms
(evaluationData, rankings) = LoadMovieLensData()

# Construct an Evaluator to, you know, evaluate them
evaluator = Evaluator(evaluationData, rankings)

# Throw in an SVD recommender
SVDAlgorithm = SVD(random_state=10)
evaluator.AddAlgorithm(SVDAlgorithm, "SVD")

# Just make random recommendations
Random = NormalPredictor()
evaluator.AddAlgorithm(Random, "Random")


# Fight!
evaluator.Evaluate(True)
```

let’s jump in

code walkthrough

content-based filtering

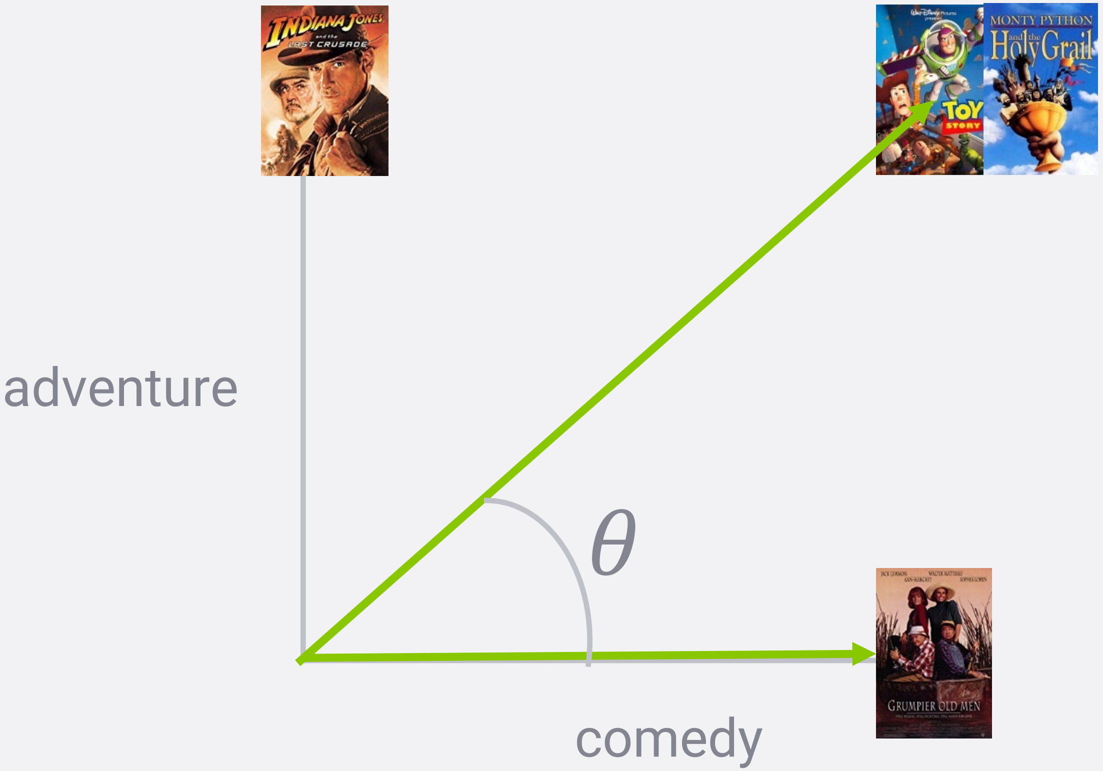
examples of movie attributes

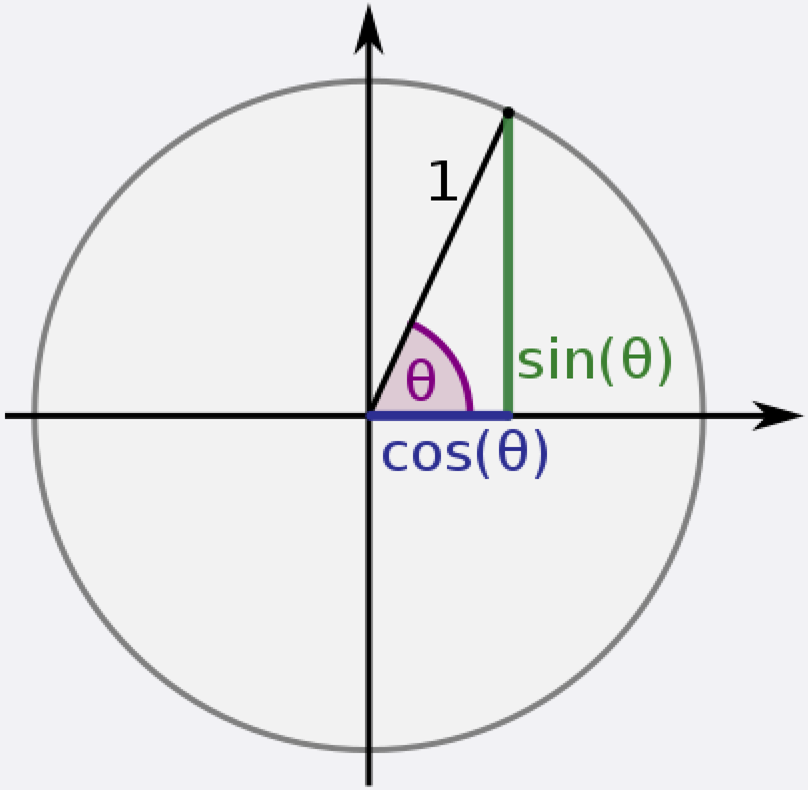
movielens genre data

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy

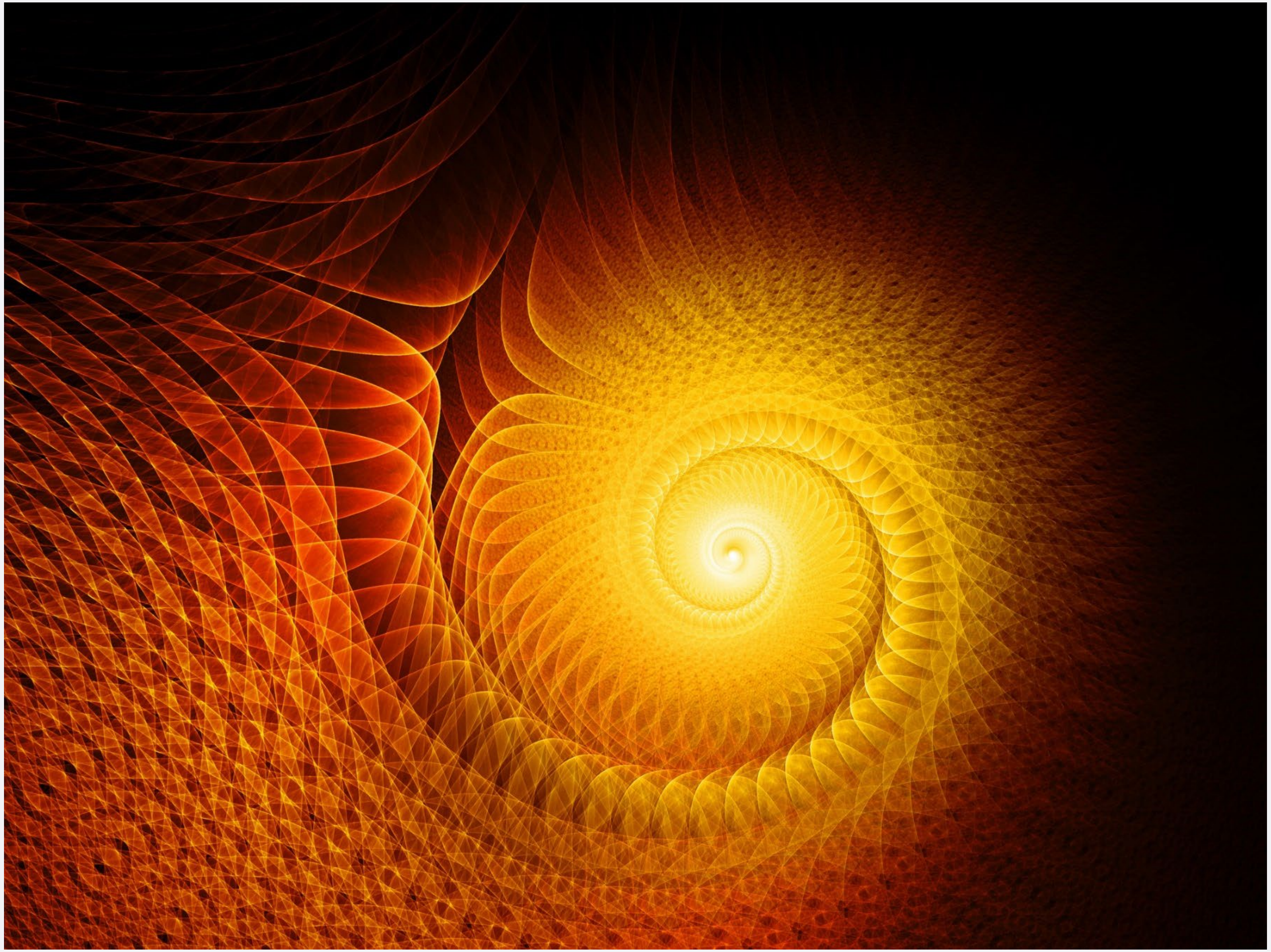
Action* Adventure* Animation* Children's* Comedy* Crime* Documentary* Drama* Fantasy* Film-Noir* Horror* Musical* Mystery* Romance* Sci-Fi* Thriller* War* Western

cosine similarity


adventure
θ
comedy


1
θ
sin(θ)
cos(θ)

multi-dimensional space!



convert genres to dimensions

0	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance

Movie	action	adventure	animation	children's	comedy	crime	documentary	drama	fantasy	film-noir	horror	musical	western	mystery	romance	sci-fi	thriller	war	western2
Toy Story	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Jumanji	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Grumpier Old Men	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Waiting to Exhale	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0
Father of the Bride	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

multi-dimensional cosines

$$CosSim(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

turning it into **code**

$$CosSim(x,y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

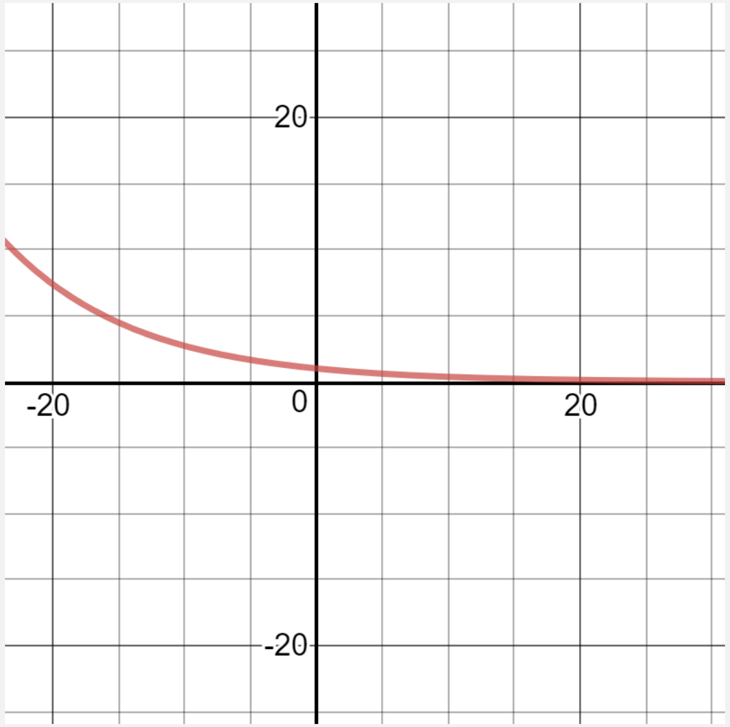
```
def computeGenreSimilarity(self, movie1, movie2, genres):
        genres1 = genres[movie1]
        genres2 = genres[movie2]
        sumxx, sumxy, sumyy = 0, 0, 0
        for i in range(len(genres1)):
            x = genres1[i]
            y = genres2[i]
            sumxx += x * x
            sumyy += y * y
            sumxy += x * y

        return sumxy/math.sqrt(sumxx*sumyy)
```

release years

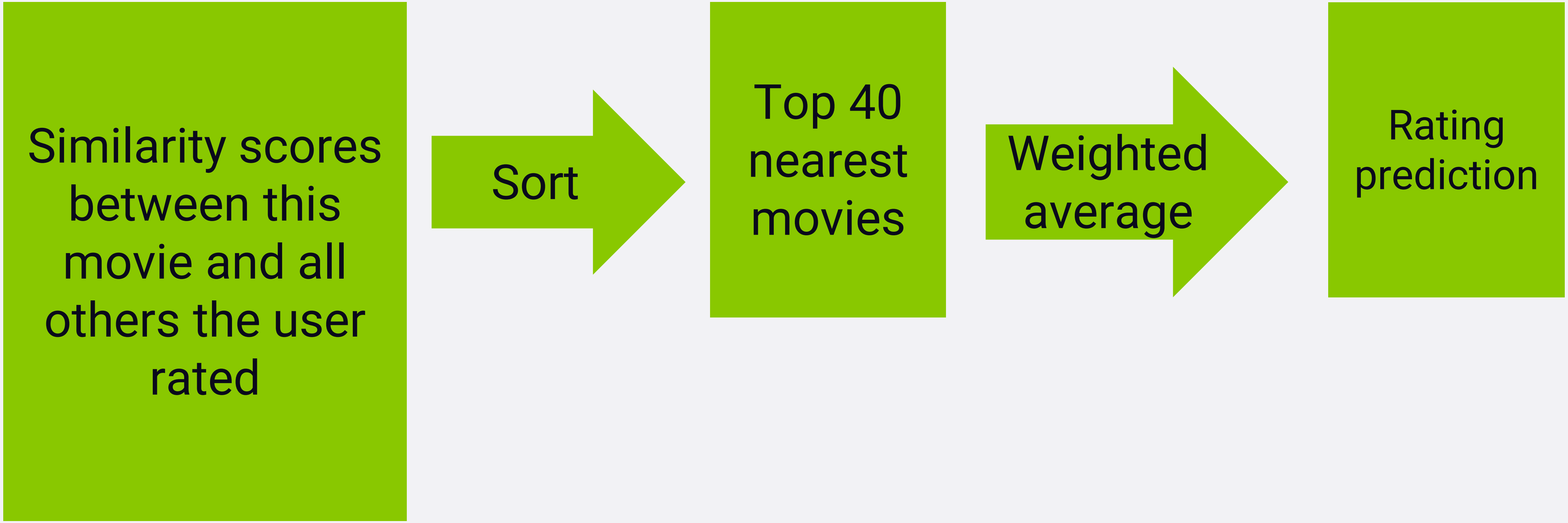
Toy Story (1995)
Jumanji (1995)
Grumpier Old Men (1995)
Waiting to Exhale (1995)
Father of the Bride Part II (1995)
Heat (1995)
Sabrina (1995)

time similarity

```
def computeYearSimilarity(self, movie1, movie2, years):
        diff = abs(years[movie1] - years[movie2])
        sim = math.exp(-diff / 10.0)
        return sim
```



k-nearest-neighbors



Similarity scores between this movie and all others the user rated

Sort

Top 40 nearest movies

Weighted average

Rating prediction

knn code

```
# Build up similarity scores between this item and everything the user rated
neighbors = []
for rating in self.trainset.ur[u]:
    genreSimilarity = self.similarities[i,rating[0]]
    neighbors.append( (genreSimilarity, rating[1]) )

# Extract the top-K most-similar ratings
k_neighbors = heapq.nlargest(self.k, neighbors, key=lambda t: t[0])

# Compute average sim score of K neighbors weighted by user ratings
simTotal = weightedSum = 0
for (simScore, rating) in k_neighbors:
    if (simScore > 0):
        simTotal += simScore
        weightedSum += simScore * rating

if (simTotal == 0):
    raise PredictionImpossible('No neighbors')

predictedRating = weightedSum / simTotal

return predictedRating
```

let’s dive in

code walkthrough

implicit ratings

a note about implicit ratings.

the algorithms we cover work just as well with *implicit* ratings as *explicit* ratings.

implicit ratings would be things like clicking on a link, purchasing something – doing something that is an implicit indication of interest instead of an explicit rating.

implicit data can be powerful

it tends to be plentiful

implicit purchase ratings can be higher quality than explicit ratings

using implicit data

just model a click / purchase / whatever as a positive rating of some arbitrary (yet consistent) value.

do NOT model the absence of a click / purchase as a negative rating – it's just missing data.

the math generally works out the same.

not all implicit ratings are created equal.

purchases good.

clicks not so much.

bleeding edge alert!

mise en scène

mise en scène data

Column #	Column Name	Description
1	ML_ID	MovieLens movie ID
2	f1	Average shot length
3	f2	Mean of color variance across the key Frames
4	f3	Standard deviation of color variance across the key Frames
5	f4	Mean of motion average across all the frames
6	f5	Mean of motion standard deviation across all the frames
7	f6	Mean of lighting key across the key frames
8	f7	Number of shots

code walkthrough

credits

*Yashar Deldjoo, Mehdi Elahi, Paolo Cremonesi "**Using Visual Features and Latent Factors for Movie Recommendation**", ACM RecSys Workshop on New Trends in Content-based Recommender Systems (CBRecSys), ACM RecSys 2016, Massachusetts Institute of Technology (MIT), September 15-19, 2016*

http://recsys.deib.polimi.it/?page_id=353

exercise

which content attribute is most powerful in producing “good” recommendations?

genre, release year, or mise en scene?

my results

genre

RMSE: 0.9552

Black Mask (Hak hap) (1996)
Joy Ride (2001)
What's Up, Tiger Lily? (1966) Missing, The (2003)
City of God (Cidade de Deus) (2002)
24: Redemption (2008)
The Hateful Eight (2015)
Wyatt Earp (1994)
True Grit (2010)
Shooter, The (1997)

year

RMSE: 0.9626

Clerks (1994)
Disclosure (1994)
Ed Wood (1994)
Houseguest (1994)
Legends of the Fall (1994)
Madness of King George, The (1994)
Mary Shelley's Frankenstein (Frankenstein) (1994)
Quiz Show (1994)
Secret of Roan Inish, The (1994)
Shallow Grave (1994)

mise en scene

RMSE: 1.0663

Pain & Gain (2013)
Bring It On (2000)
Young Master, The (Shi di chu ma) (1980)
Celebrity (1998)
Yi Yi (2000)
Eating Raoul (1982)
Stuck on You (2003)
Cat Returns, The (Neko no ongaeshi) (2002)
Reckless (1984)
Sunless (Sans Soleil) (1983)

better year-based recs

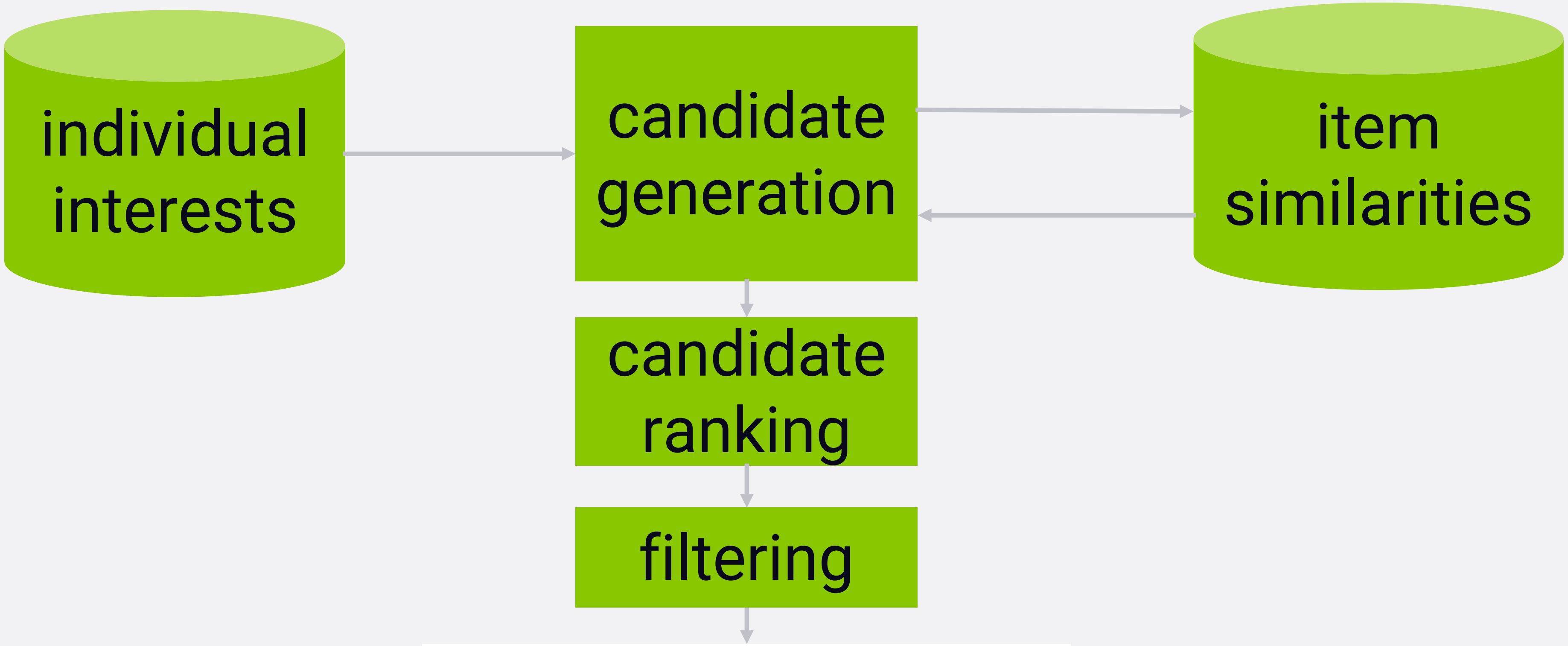
In Evaluator.py's SampleTopNRecs:

```
print ("\nWe recommend:")
           for userID, movieID, actualRating, estimatedRating, _ in predictions:
               intMovieID = int(movieID)
               recommendations.append((intMovieID, estimatedRating, ml.getPopularityRanks()[intMovieID]))

           recommendations.sort(key=lambda x: x[2])
           recommendations.sort(key=lambda x: x[1], reverse=True)
```

```
We recommend:
Clerks (1994) 3.37112480076
Quiz Show (1994) 3.37112480076
Ed Wood (1994) 3.37112480076
Legends of the Fall (1994) 3.37112480076
Crow, The (1994) 3.37112480076
Hoop Dreams (1994) 3.37112480076
Muriel's Wedding (1994) 3.37112480076
Disclosure (1994) 3.37112480076
Adventures of Priscilla, Queen of the Desert, The (1994) 3.37112480076
River Wild, The (1994) 3.37112480076
```

neighborhood-based collaborative filtering

(one) anatomy of a top-N recommender



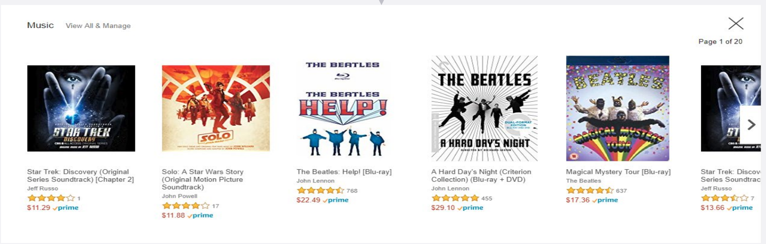
individual interests

candidate generation

item similarities

candidate ranking

filtering



Music View All & Manage

Page 1 of 20

Star Trek: Discovery (Original Series Soundtrack) [Chapter 2]
Jeff Russo
★★★★☆ 1
$11.29 ✓prime

Solo: A Star Wars Story (Original Motion Picture Soundtrack)
John Powell
★★★★☆ 17
$11.88 ✓prime

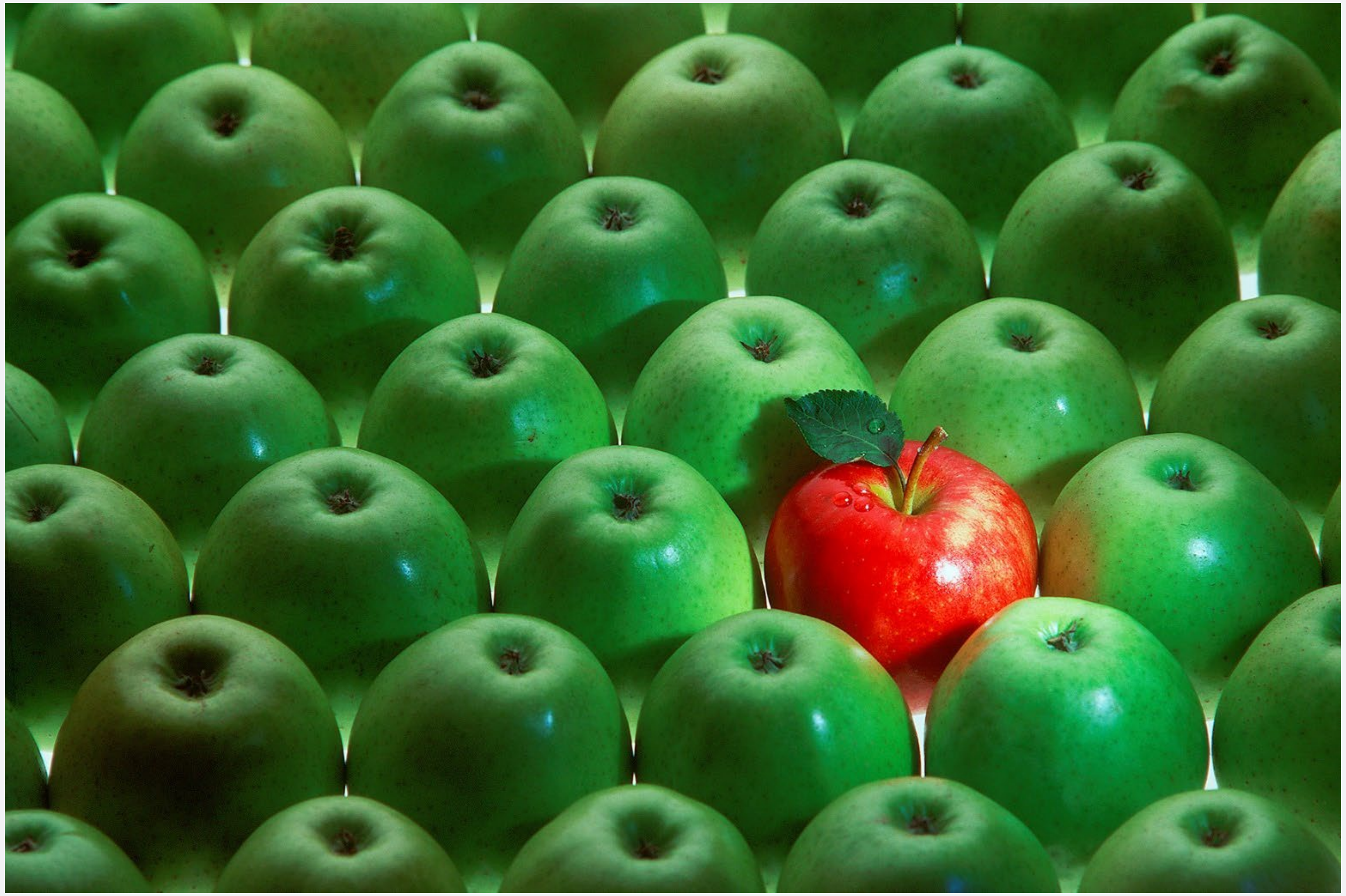
The Beatles: Help! [Blu-ray]
John Lennon
★★★★½ 768
$22.49 ✓prime

A Hard Day's Night (Criterion Collection) (Blu-ray + DVD)
John Lennon
★★★★★ 455
$29.10 ✓prime

Magical Mystery Tour [Blu-ray]
The Beatles
★★★★½ 637
$17.36 ✓prime

Star Trek: Discov
Series Soundtrac
Jeff Russo
★★★½☆ 7
$13.66 ✓prime

ways to measure similarity



cosine similarity

$$CosSim(x,y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

sparsity

	Indiana Jones	Star Wars	Shape of Water	Incredibles	Casablanca
Bob	4				
Ted					
Alice				5	

adjusted cosine

$$CosSim(x,y) = \frac{\sum_i\left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

(item-based) pearson similarity

$$CosSim(x,y) = \frac{\sum_i \left((x_i - \bar{\imath})(y_i - \bar{\imath}) \right)}{\sqrt{\sum_i (x_i - \bar{\imath})^2} \sqrt{\sum_i (y_i - \bar{\imath})^2}}$$

spearman rank correlation

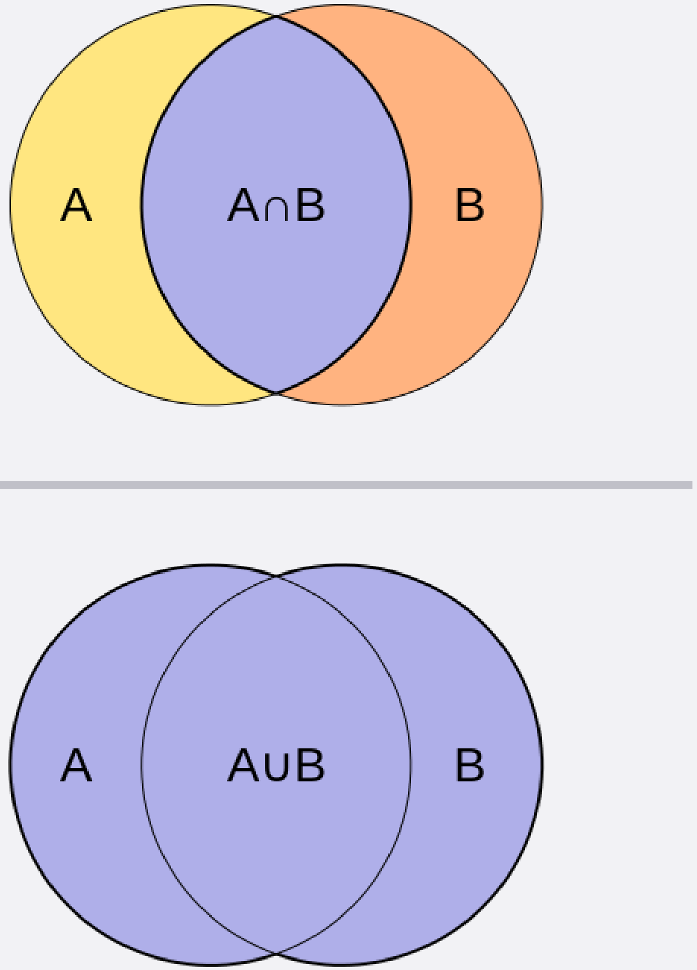
pearson similarity based on ranks, not ratings

mean squared difference

$$MSD(x,y) = \frac{\sum_{i \in I_{xy}} (x_i - y_i)^2}{|I_{xy}|}$$

$$MSDsim(x,y) = \frac{1}{MSD(x,y) + 1}$$

jaccard similarity



recap

THE OTHER WAY
THIS WAY
THAT WAY

cosine

adjusted cosine

pearson

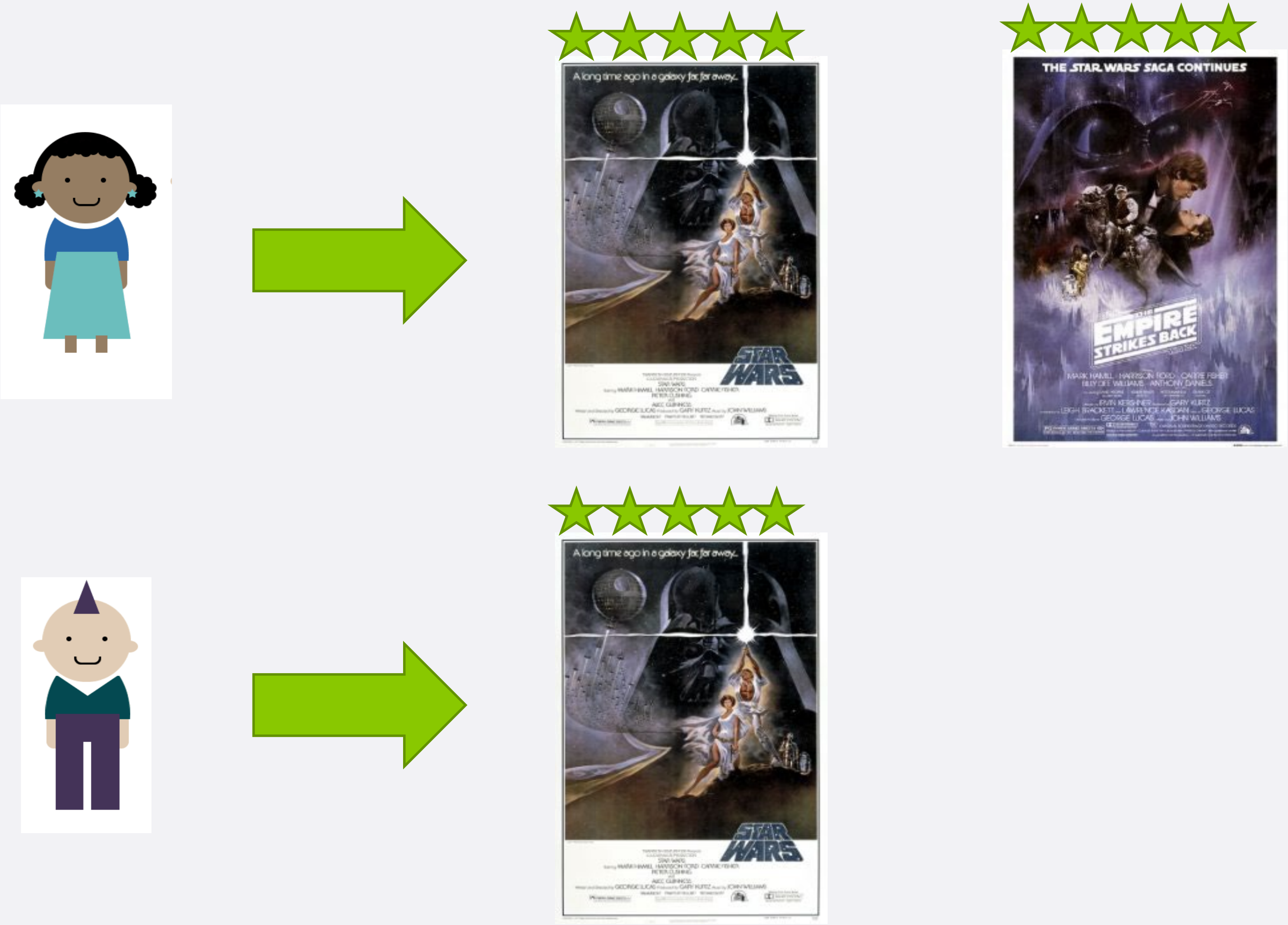
spearman

msd

jaccard

user-based collaborative filtering

user-based
collaborative filtering


A long time ago in a galaxy far, far away....
STAR WARS
THE STAR WARS SAGA CONTINUES
THE EMPIRE STRIKES BACK
A long time ago in a galaxy far, far away....
STAR WARS

user-based collaborative filtering

THE STAR WARS SAGA CONTINUES
EMPIRE STRIKES BACK

user-based collaborative filtering

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Bob	4	5			
Ted					1
Ann		5	5	5	

user-based
collaborative filtering

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Bob	4	5			
Ted					1
Ann		5	5	5	

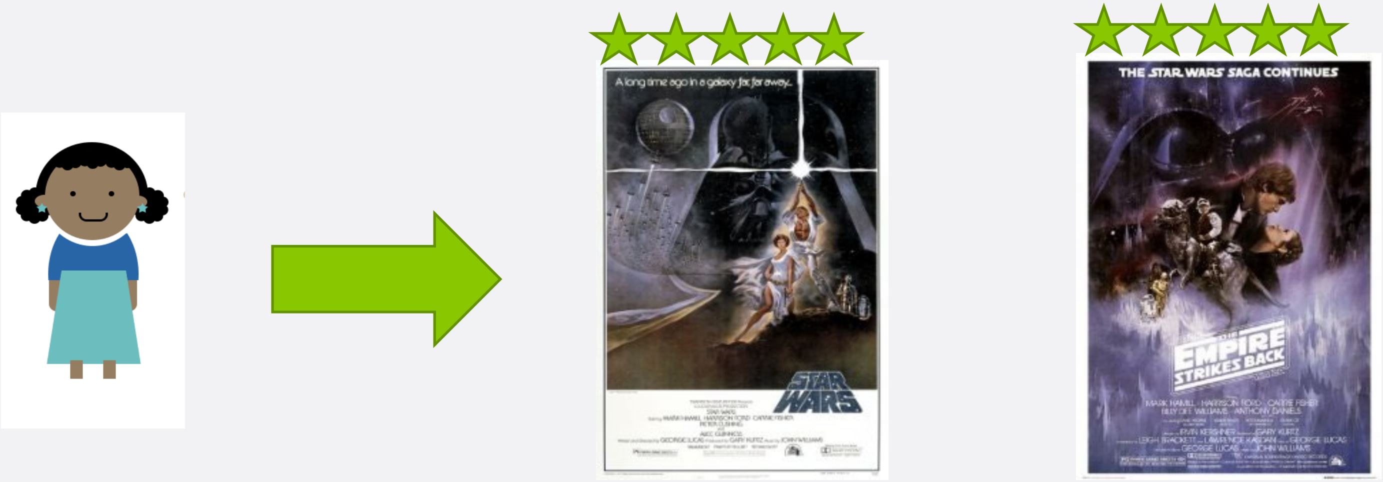
	Bob	Ted	Ann
Bob	1	0	1
Ted	0	1	0
Ann	1	0	1

user-based collaborative filtering

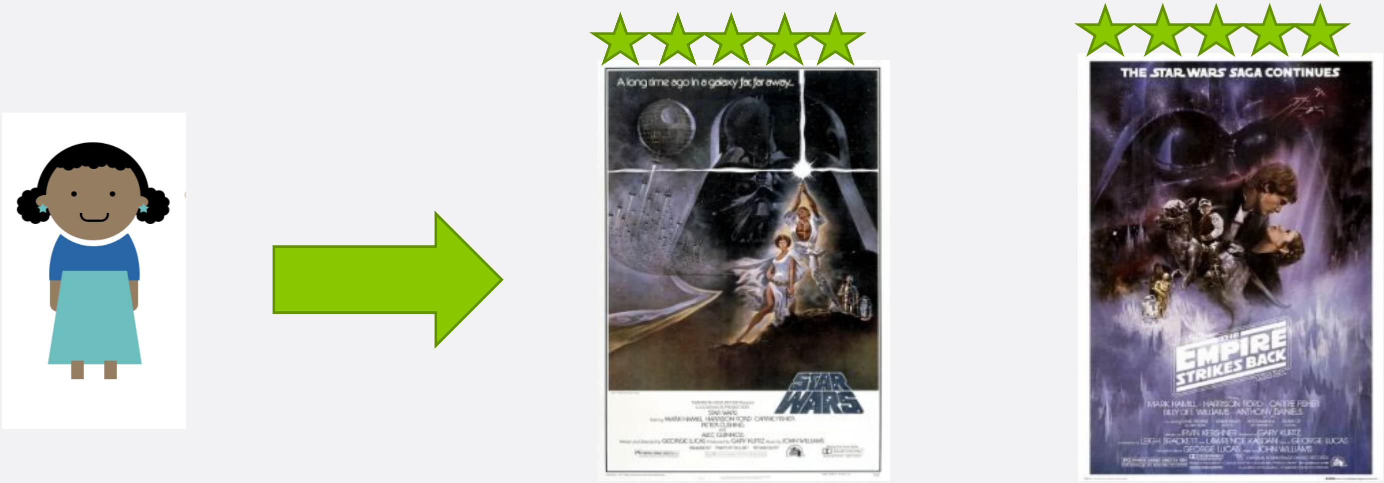
	Bob	Ted	Ann
Bob	1	0	1
Ted	0	1	0
Ann	1	0	1

Bob’s neighbors: Ann: 1.0, Ted: 0

candidate generation


A long time ago in a galaxy far, far away....
STAR WARS
THE STAR WARS SAGA CONTINUES
THE EMPIRE STRIKES BACK

candidate scoring

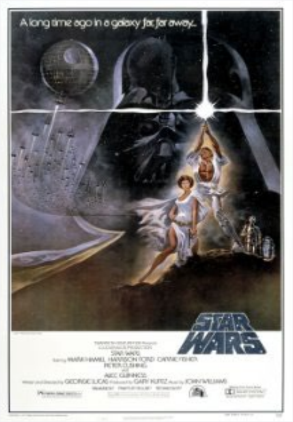

A long time ago in a galaxy far, far away....
STAR WARS
THE STAR WARS SAGA CONTINUES
THE EMPIRE STRIKES BACK

1.0

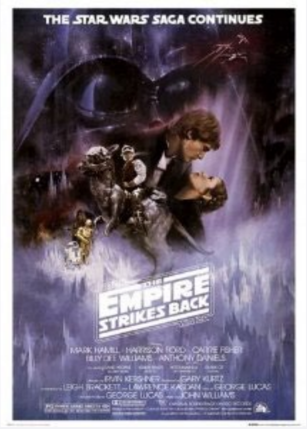
1.0

1.0

candidate sorting

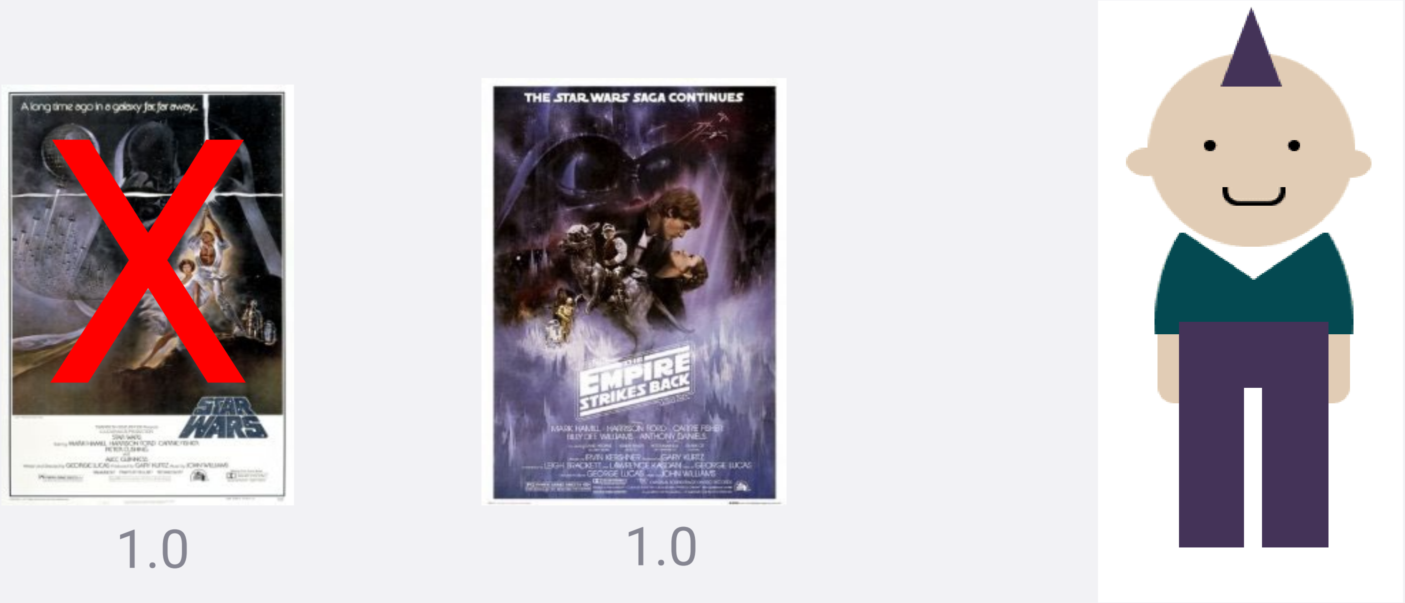


1.0



1.0

candidate filtering



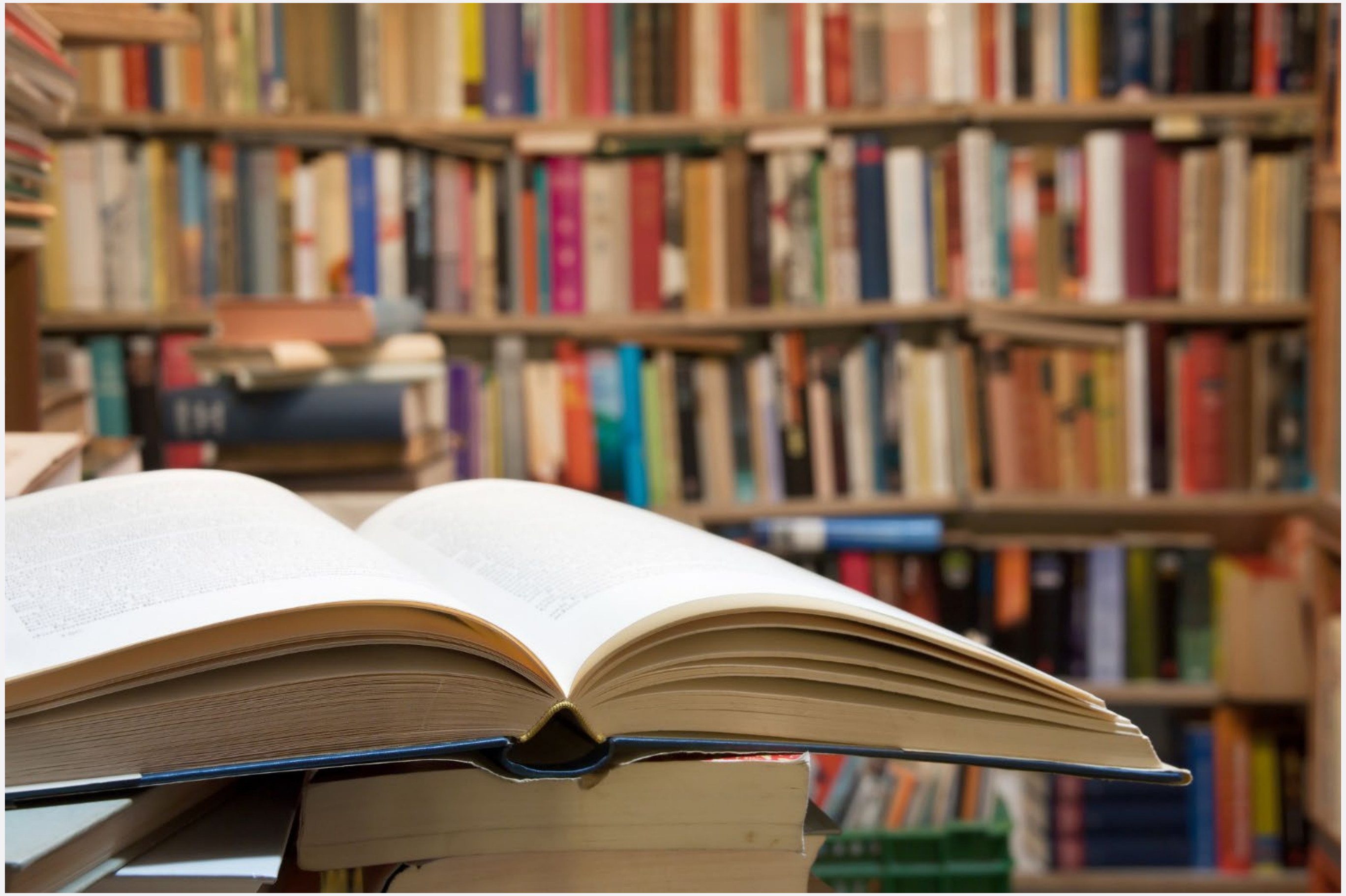
user-based collaborative filtering

- user -> item rating matrix
- user -> user similarity matrix
- look up similar users
- candidate generation
- candidate scoring
- candidate filtering

code walkthrough

item-based collaborative filtering

things, not people



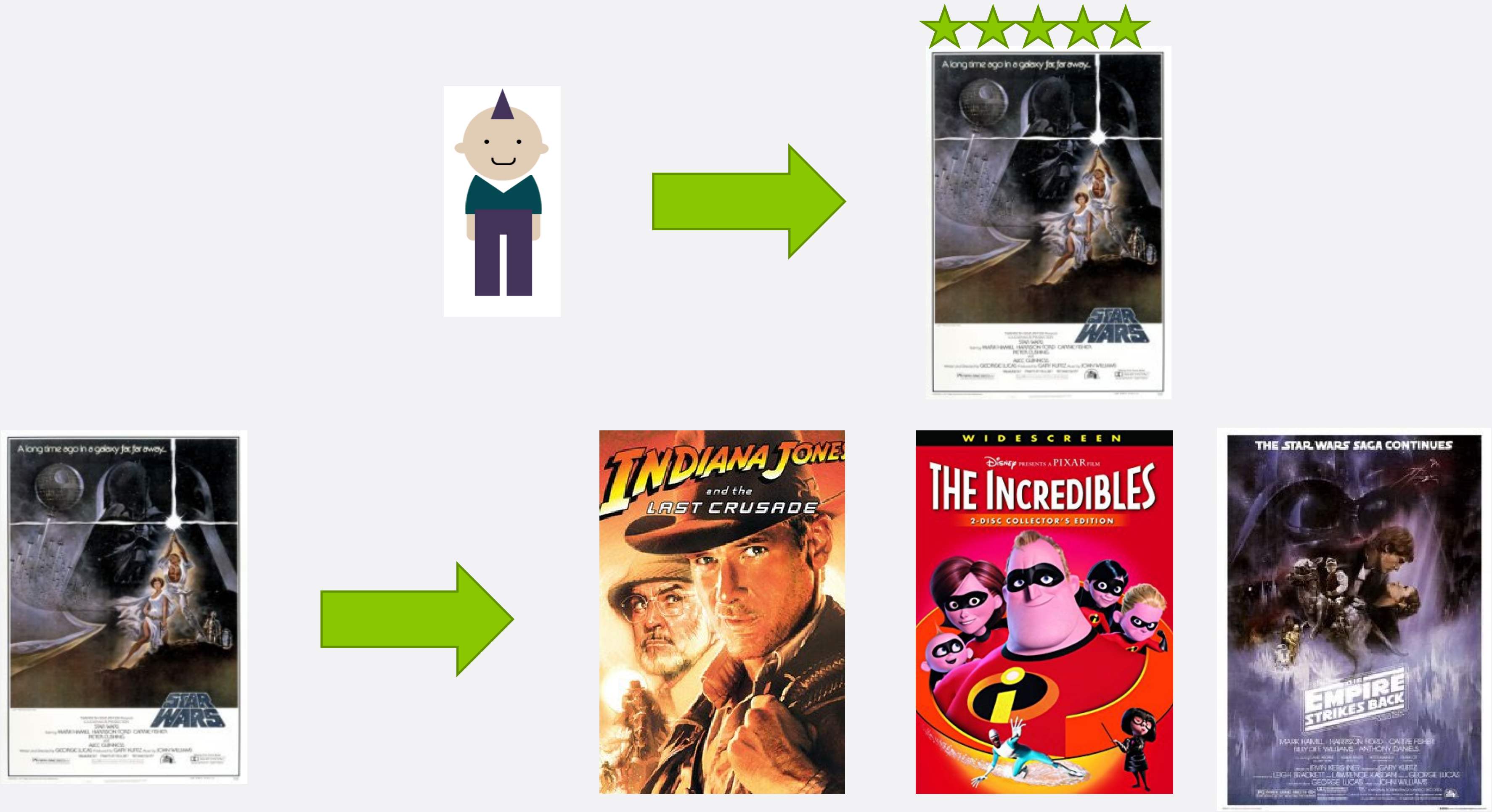
item-based collaborative filtering

	Bob	Ted	Ann
Indiana Jones	4		
Star Wars	5		5
Empire Strikes Back			5
Incredibles			5
Casablanca		1	

item-based collaborative filtering

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Indiana Jones	1	1	0	0	0
Star Wars	1	1	1	1	0
Empire Strikes Back	1	1	1	1	0
Incredibles	1	1	1	1	0
Casablanca	0	0	0	0	1

item-based collaborative filtering


A long time ago in a galaxy far, far away....
STAR WARS
A long time ago in a galaxy far, far away....
STAR WARS
INDIANA JONES and the LAST CRUSADE
WIDESCREEN
Disney presents a PIXAR film
THE INCREDIBLES
2-DISC COLLECTOR'S EDITION
THE STAR WARS SAGA CONTINUES
THE EMPIRE STRIKES BACK

code walkthrough

exercise

Build recommendation candidates from items above a certain rating or similarity threshold, instead of the top 10.

exercise solution: item-based

```
#kNeighbors = heapq.nlargest(k, testUserRatings, key=lambda t: t[1])
kNeighbors = []
for rating in testUserRatings:
    if rating[1] > 4.0:
        kNeighbors.append(rating)
```

before	after
James Dean Story, The (1957)	Kiss of Death (1995)
Get Real (1998)	Amos & Andrew (1993)
Kiss of Death (1995)	Edge of Seventeen (1998)
Set It Off (1996)	Get Real (1998)
How Green Was My Valley (1941)	Grace of My Heart (1996)
Amos & Andrew (1993)	Relax... It's Just Sex (1998)
My Crazy Life (Mi vida loca) (1993)	My Crazy Life (Mi vida loca) (1993)
Grace of My Heart (1996)	Set It Off (1996)
Fanny and Alexander (Fanny och Alexander) (1982)	Bean (1997)
Wild Reeds (Les roseaux sauvages) (1994)	Joe's Apartment (1996)
Edge of Seventeen (1998)	Lost & Found (1999)

exercise solution: user-based

```
#kNeighbors = heapq.nlargest(k, similarUsers, key=lambda t: t[1])
kNeighbors = []
for rating in similarUsers:
    if rating[1] > 0.95:
        kNeighbors.append(rating)
```

before

Inception (2010)
Star Wars: Episode V - The Empire Strikes Back (1980)
Bourne Identity, The (1988)
Crouching Tiger, Hidden Dragon (Wo hu cang long) (2000)
Dark Knight, The (2008)
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)
Departed, The (2006)
Dark Knight Rises, The (2012)
Back to the Future (1985)
Gravity (2013)
Fight Club (1999)

after

Star Wars: Episode IV - A New Hope (1977)
Matrix, The (1999)
Star Wars: Episode V - The Empire Strikes Back (1980)
Fight Club (1999)
Back to the Future (1985)
Raiders of the Lost Ark (1981)
American Beauty (1999)
Toy Story (1995)
Godfather, The (1972)
Star Wars: Episode VI - Return of the Jedi (1983)
Lord of the Rings: The Fellowship of the Ring, The (2001)

evaluating collaborative filtering

exercise

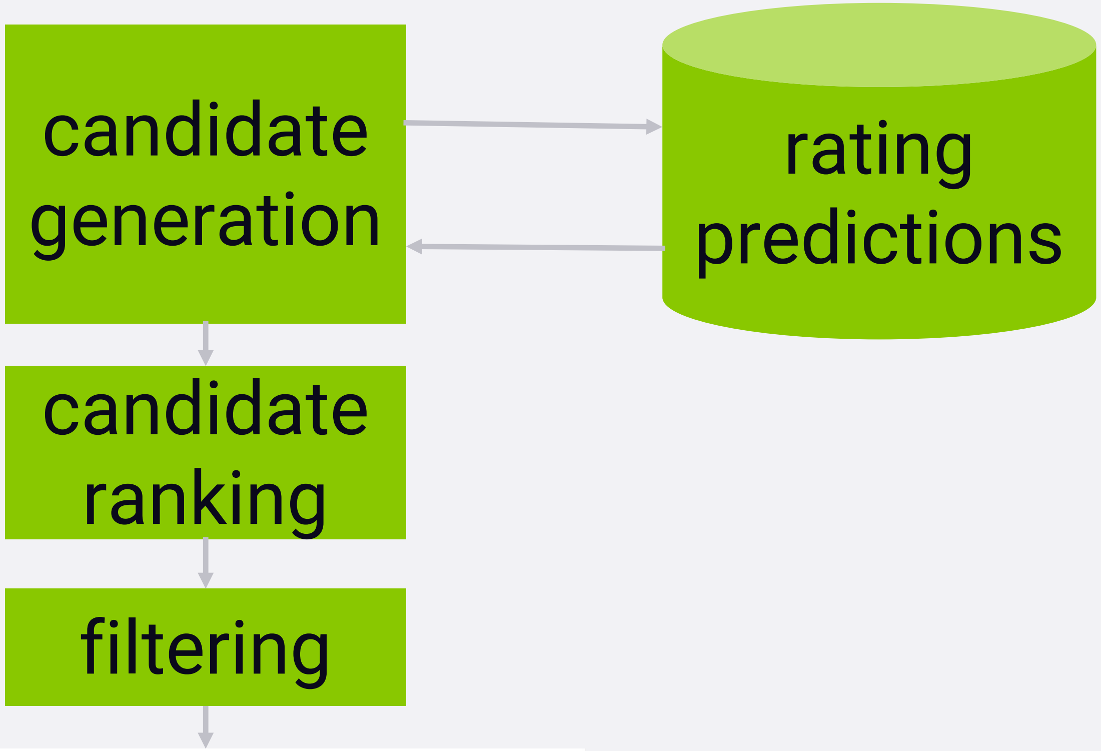
measure the hit-rate of item-based collaborative filtering.

exercise solution

```
sim_options = {'name': 'cosine',
               'user_based': False
               }


for uiid in range(trainSet.n_users):

    userRatings = trainSet.ur[uiid]
    kNeighbors = heapq.nlargest(k, userRatings, key=lambda t: t[1])

    candidates = defaultdict(float)
    for itemID, rating in kNeighbors:
        similarityRow = simsMatrix[itemID]
        for innerID, score in enumerate(similarityRow):
            candidates[innerID] += score * (rating / 5.0)

    # Build a dictionary of stuff the user has already seen
```

k-nearest-neighbors (knn) recommenders

another way to do it

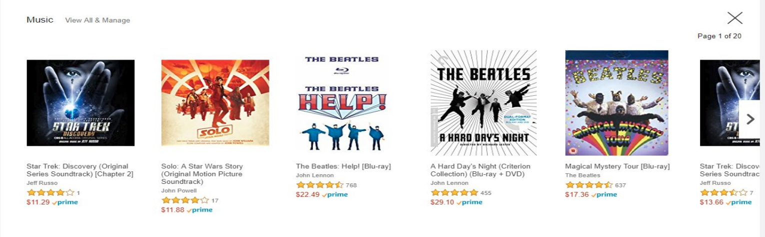


candidate generation

rating predictions

candidate ranking

filtering



Star Trek: Discovery (Original Series Soundtrack) [Chapter 2]
Jeff Russo
1
$11.29 prime

Solo: A Star Wars Story (Original Motion Picture Soundtrack)
John Powell
17
$11.88 prime

The Beatles: Help! [Blu-ray]
John Lennon
768
$22.49 prime

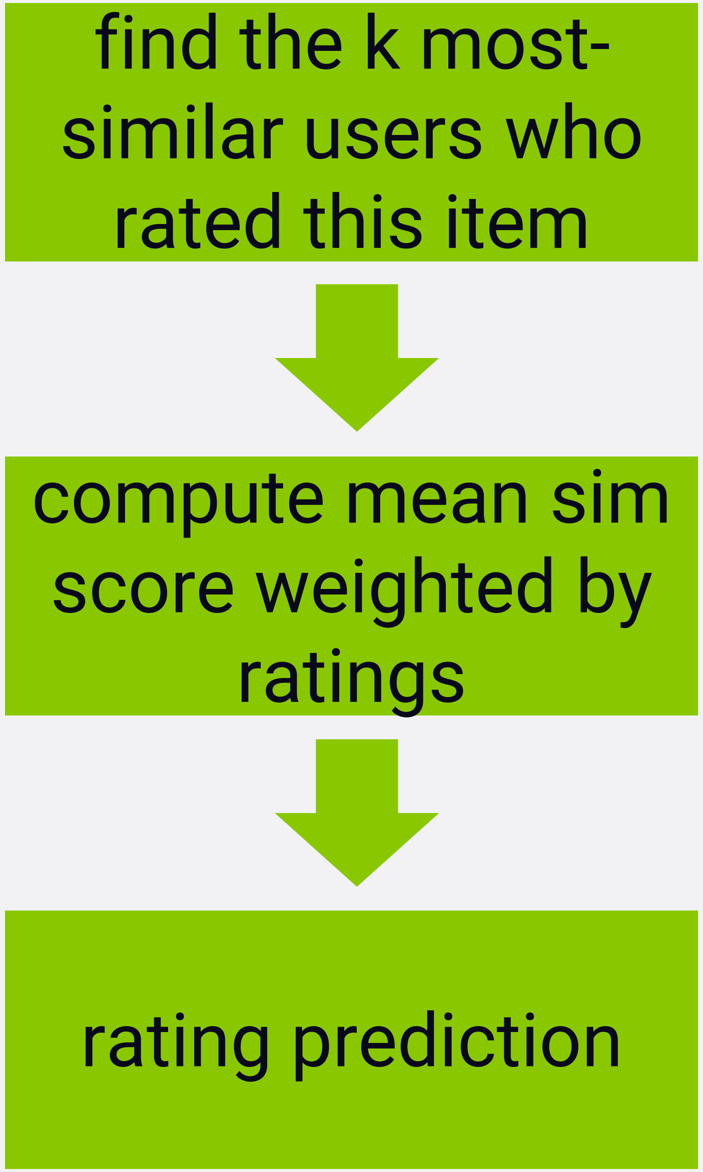
A Hard Day's Night (Criterion Collection) (Blu-ray + DVD)
John Lennon
455
$29.10 prime

Magical Mystery Tour [Blu-ray]
The Beatles
637
$17.36 prime

Star Trek: Discov
Series Soundtrac
Jeff Russo
7
$13.66 prime

user-based KNN

for user u and item i…



find the k most-similar users who rated this item

compute mean sim score weighted by ratings

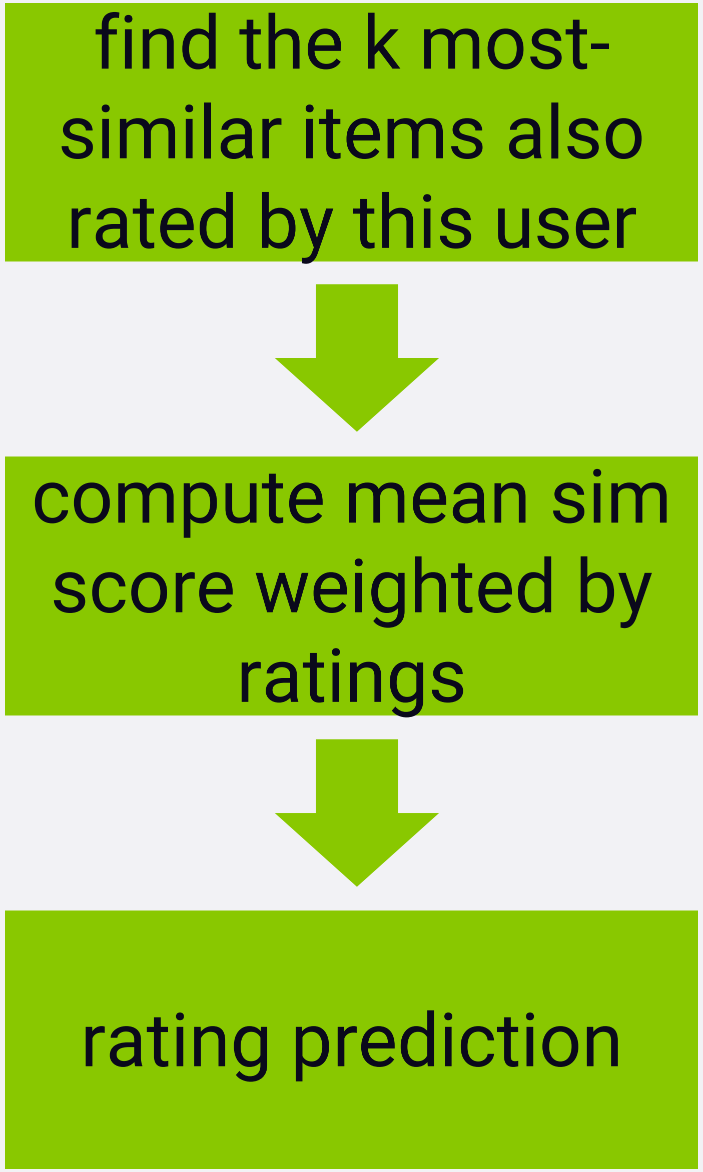
rating prediction

user-based knn

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

item-based KNN

for user u and item i…



find the k most-similar items also rated by this user

compute mean sim score weighted by ratings

rating prediction

user-based knn

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(j)} sim(i,j)}$$

code walkthrough

exercise

try out different similarity metrics: cosine, msd, and pearson.

exercise results: user-based

cosine	msd	pearson
RMSE: 0.9961	RMSE: 0.9713	RMSE: 1.0016
One Magic Christmas (1985)	One Magic Christmas (1985)	Othello (1995)
Step Into Liquid (2002)	Step Into Liquid (2002)	Step Into Liquid (2002)
Art of War, The (2000)	Art of War, The (2000)	Dreamscape (1984)
Taste of Cherry (1997)	Taste of Cherry (1997)	Taste of Cherry (1997)
King Is Alive, The (2000)	King Is Alive, The (2000)	King Is Alive, The (2000)
Innocence (2000)	Innocence (2000)	Innocence (2000)
Maelstrom (2000)	Maelstrom (2000)	Maelstrom (2000)
Faust (1926)	Faust (1926)	Last Seduction, The (1994)
Seconds (1966)	Seconds (1966)	Amazing Grace (2006)
Amazing Grace (2006)	Amazing Grace (2006)	Unvanquished, The (1957)

exercise results: item-based

cosine	msd	pearson
RMSE: 0.9995	RMSE: 0.9424	RMSE: 0.9928
Life in a Day (2011)	Life in a Day (2011)	Hearts and Minds (1996)
Under Suspicion (2000)	Under Suspicion (2000)	Pokemon the Movie 2000 (2000)
Asterix and the Gauls (1967)	Asterix and the Gauls (1967)	Eureka (2000)
Find Me Guilty (2006)	Find Me Guilty (2006)	Silent Running (1972)
Elementary Particles, The (2006)	Elementary Particles (2006)	It Might Get Loud (2008)
Asterix and the Vikings (2006)	Asterix and the Vikings (2006)	Dinner Rush (2000)
From the Sky Down (2011)	From the Sky Down (2011)	Brainstorm (1983)
Vive L'Amour (1994)	Vive L'Amour (1994)	Europa (Zentropa) (1991)
Vagabond (1985)	Vagabond (1985)	Gerry (2002)
Ariel (1988)	Ariel (1988)	Soul Kitchen (2009)

more experiments

KNNWithZScore

RMSE: 0.9347

One Magic Christmas (1985)
Taste of Cherry (1997)
King Is Alive, The (2000)
Innocence (2000)
MaelstrÃ¶m (2000)
Amazing Grace (2006)
Unvanquished, The (1957)
Undertow (2004)
Big Town, The (1987)
Masquerade (1988)

KNNWithMeans

RMSE: 0.9306

One Magic Christmas (1985)
Taste of Cherry (1997)
King Is Alive, The (2000)
Innocence (2000)
Maelstrom (2000)
Amazing Grace (2006)
Unvanquished, The (1957)
Undertow (2004)
Soul Kitchen (2009)
Big Town, The (1987)

KNNBaseline

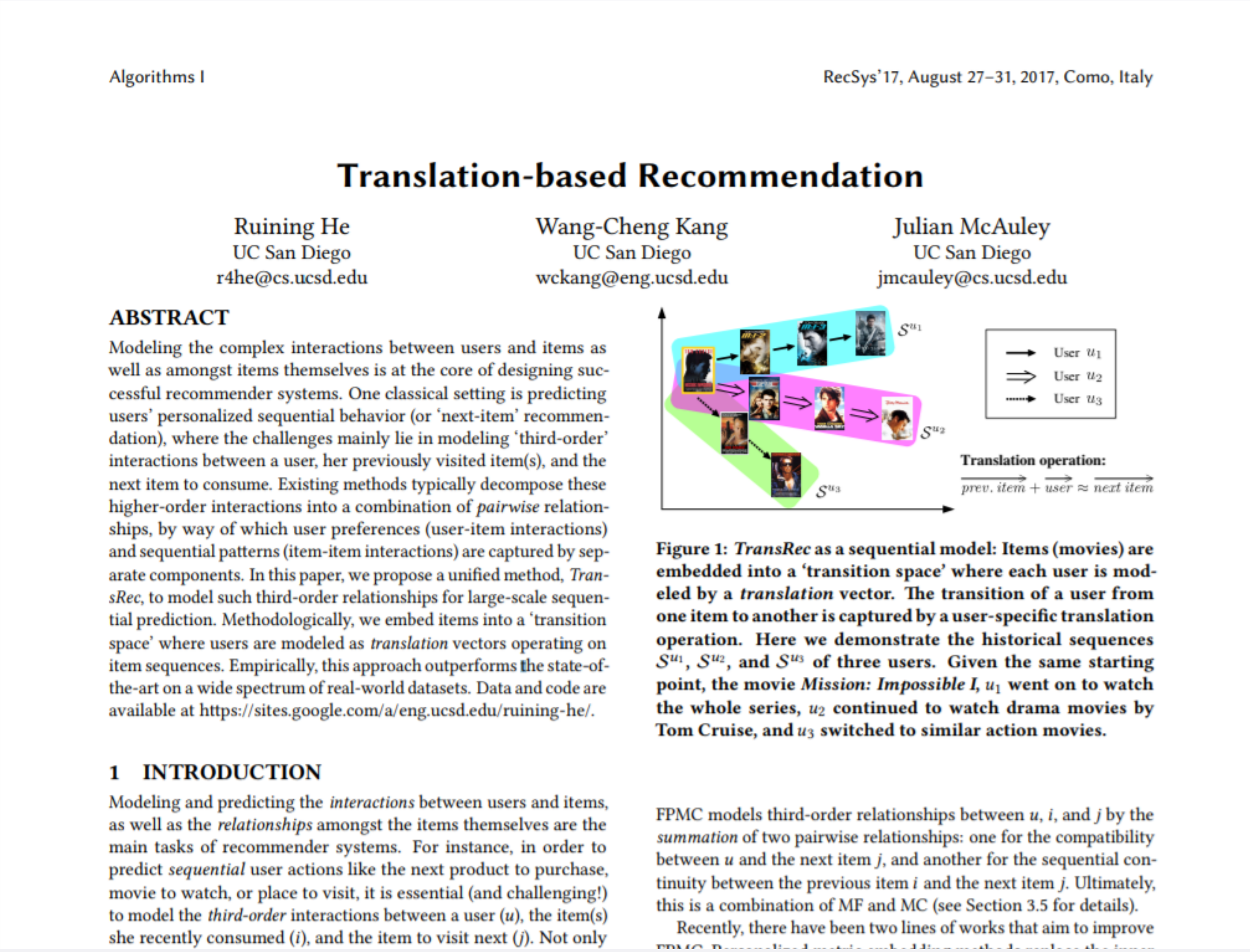
RMSE: 0.9129

Digimon: The Movie (2000)
Pokemon 3: The Movie (2001)
City of Industry (1997) Amityville
Curse, The (1990)
Grand, The (2007)
Tracey Fragments, The (2007)
T-Rex: Back to the Cretaceous (1998)
Above the Law (1988)
Enforcer, The (1976)
Kirikou and the Sorceress (1998)

why is knn so bad?

bleeding edge alert!

translation-based recommendations



Algorithms I RecSys'17, August 27–31, 2017, Como, Italy

Translation-based Recommendation

Ruining He
UC San Diego
r4he@cs.ucsd.edu

Wang-Cheng Kang
UC San Diego
wckang@eng.ucsd.edu

Julian McAuley
UC San Diego
jmcauley@cs.ucsd.edu

ABSTRACT

Modeling the complex interactions between users and items as well as amongst items themselves is at the core of designing successful recommender systems. One classical setting is predicting users' personalized sequential behavior (or 'next-item' recommendation), where the challenges mainly lie in modeling 'third-order' interactions between a user, her previously visited item(s), and the next item to consume. Existing methods typically decompose these higher-order interactions into a combination of *pairwise* relationships, by way of which user preferences (user-item interactions) and sequential patterns (item-item interactions) are captured by separate components. In this paper, we propose a unified method, *TransRec*, to model such third-order relationships for large-scale sequential prediction. Methodologically, we embed items into a 'transition space' where users are modeled as *translation* vectors operating on item sequences. Empirically, this approach outperforms the state-of-the-art on a wide spectrum of real-world datasets. Data and code are available at https://sites.google.com/a/eng.ucsd.edu/ruining-he/.

1 INTRODUCTION

Modeling and predicting the *interactions* between users and items, as well as the *relationships* amongst the items themselves are the main tasks of recommender systems. For instance, in order to predict *sequential* user actions like the next product to purchase, movie to watch, or place to visit, it is essential (and challenging!) to model the *third-order* interactions between a user (u), the item(s) she recently consumed (i), and the item to visit next (j). Not only

Figure 1: *TransRec* as a sequential model: Items (movies) are embedded into a 'transition space' where each user is modeled by a *translation* vector. The transition of a user from one item to another is captured by a user-specific translation operation. Here we demonstrate the historical sequences $\mathcal{S}^{u_1}$, $\mathcal{S}^{u_2}$, and $\mathcal{S}^{u_3}$ of three users. Given the same starting point, the movie *Mission: Impossible I*, u_1 went on to watch the whole series, u_2 continued to watch drama movies by Tom Cruise, and u_3 switched to similar action movies.

FPMC models third-order relationships between u, i, and j by the *summation* of two pairwise relationships: one for the compatibility between u and the next item j, and another for the sequential continuity between the previous item i and the next item j. Ultimately, this is a combination of MF and MC (see Section 3.5 for details).

Recently, there have been two lines of works that aim to improve

translation-based recommendations

https://sites.google.com/view/ruining-he/

translation-based recommendations

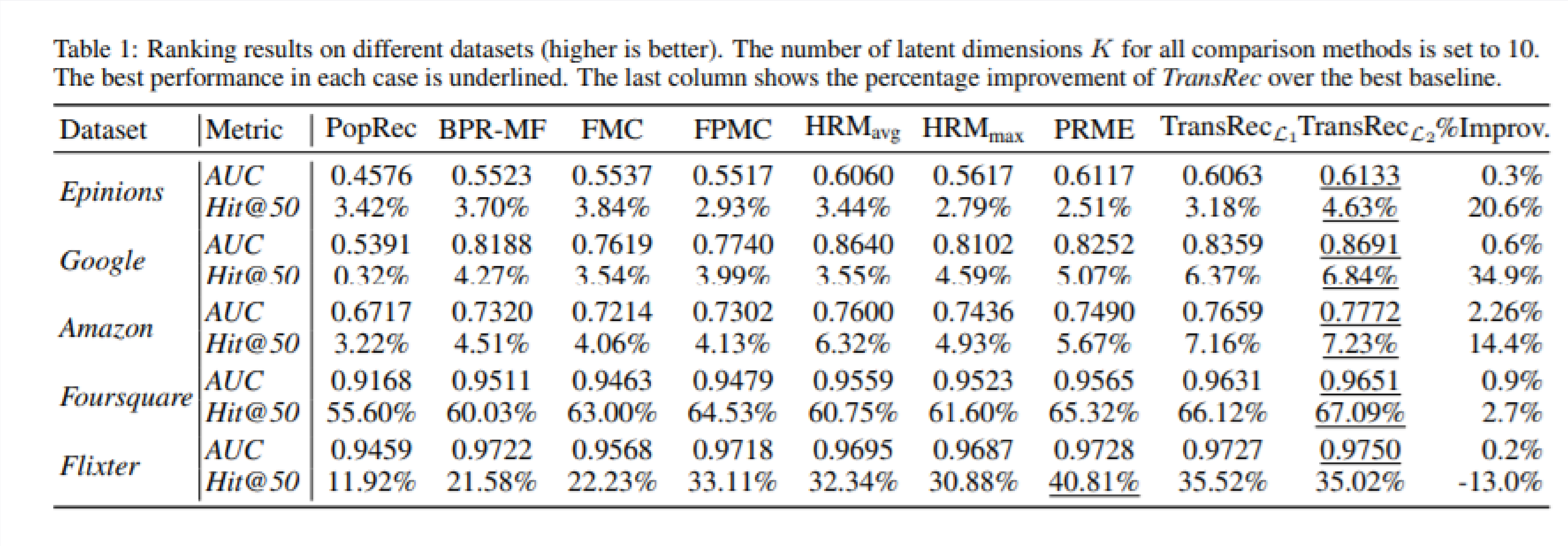
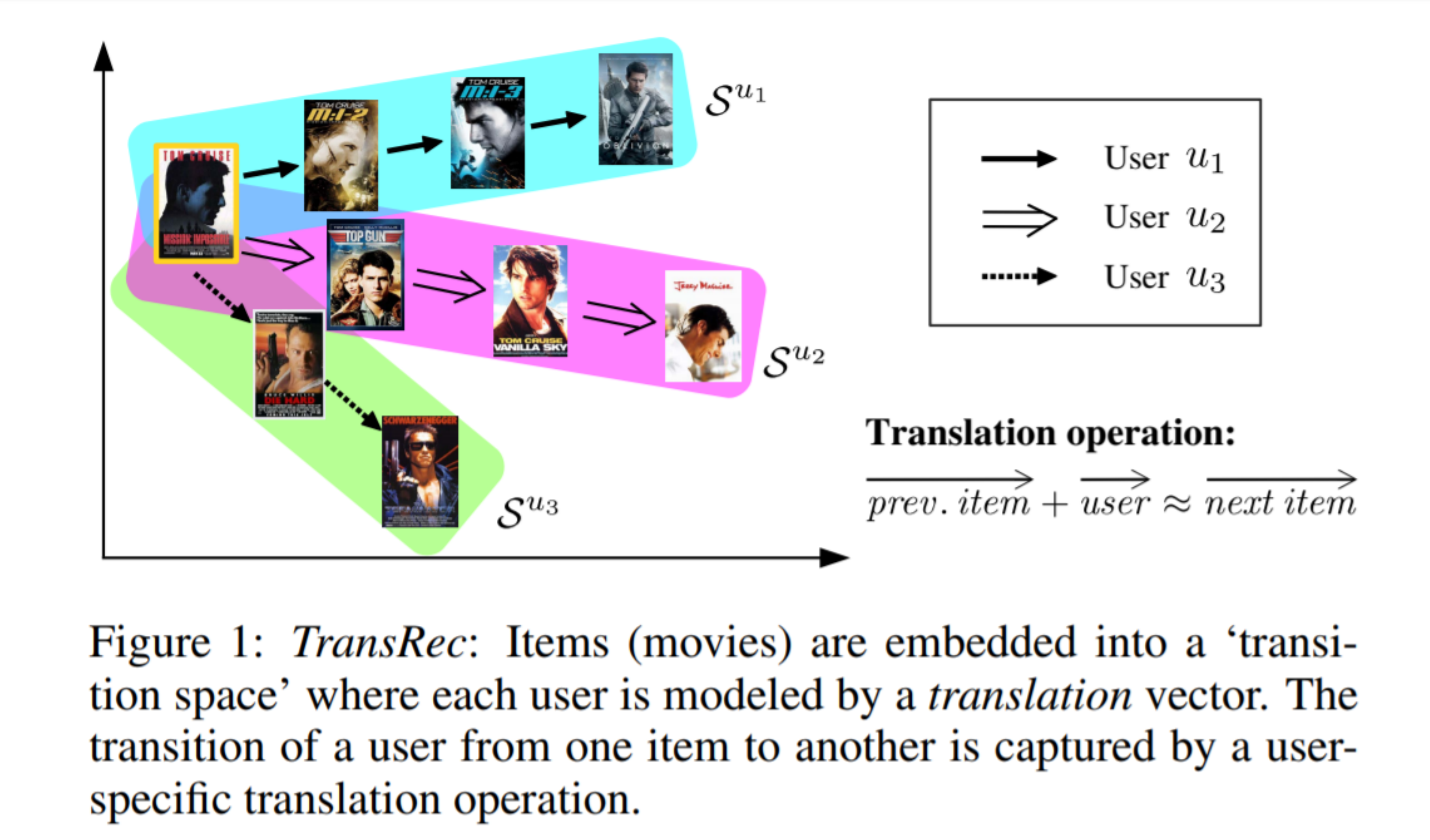


Table 1: Ranking results on different datasets (higher is better). The number of latent dimensions K for all comparison methods is set to 10. The best performance in each case is underlined. The last column shows the percentage improvement of *TransRec* over the best baseline.

Dataset	Metric	PopRec	BPR-MF	FMC	FPMC	HRM$_{avg}$	HRM$_{max}$	PRME	TransRec$_{\mathcal{L}_1}$	TransRec$_{\mathcal{L}_2}$	%Improv.
Epinions	*AUC*	0.4576	0.5523	0.5537	0.5517	0.6060	0.5617	0.6117	0.6063	0.6133	0.3%
	Hit@50	3.42%	3.70%	3.84%	2.93%	3.44%	2.79%	2.51%	3.18%	4.63%	20.6%
Google	*AUC*	0.5391	0.8188	0.7619	0.7740	0.8640	0.8102	0.8252	0.8359	0.8691	0.6%
	Hit@50	0.32%	4.27%	3.54%	3.99%	3.55%	4.59%	5.07%	6.37%	6.84%	34.9%
Amazon	*AUC*	0.6717	0.7320	0.7214	0.7302	0.7600	0.7436	0.7490	0.7659	0.7772	2.26%
	Hit@50	3.22%	4.51%	4.06%	4.13%	6.32%	4.93%	5.67%	7.16%	7.23%	14.4%
Foursquare	*AUC*	0.9168	0.9511	0.9463	0.9479	0.9559	0.9523	0.9565	0.9631	0.9651	0.9%
	Hit@50	55.60%	60.03%	63.00%	64.53%	60.75%	61.60%	65.32%	66.12%	67.09%	2.7%
Flixter	*AUC*	0.9459	0.9722	0.9568	0.9718	0.9695	0.9687	0.9728	0.9727	0.9750	0.2%
	Hit@50	11.92%	21.58%	22.23%	33.11%	32.34%	30.88%	40.81%	35.52%	35.02%	-13.0%

translation-based recommendations


$\mathcal{S}^{u_1}$
$\mathcal{S}^{u_2}$
$\mathcal{S}^{u_3}$
→ User u_1
⇒ User u_2
⇢ User u_3

Translation operation:

$$\overrightarrow{prev.\ item} + \overrightarrow{user} \approx \overrightarrow{next\ item}$$

Figure 1: *TransRec*: Items (movies) are embedded into a 'transition space' where each user is modeled by a *translation* vector. The transition of a user from one item to another is captured by a user-specific translation operation.

model-based methods

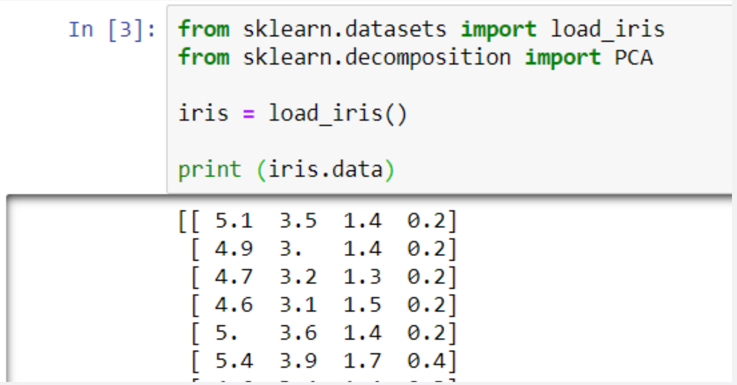
matrix factorization

the problem

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Bob	4	5	?	?	?
Ted	?	?	?	?	1
Ann	?	5	5	5	?

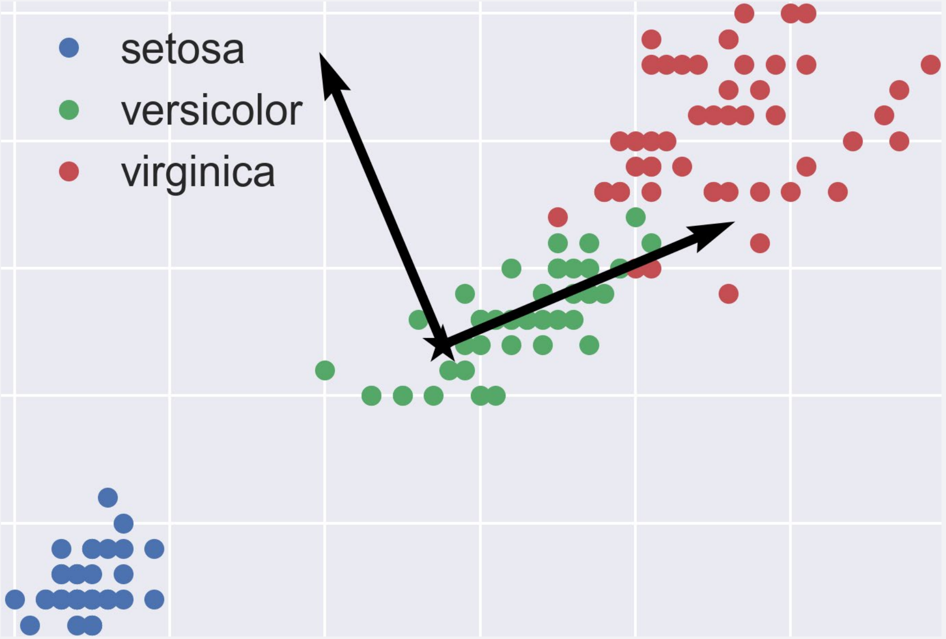
principal component analysis

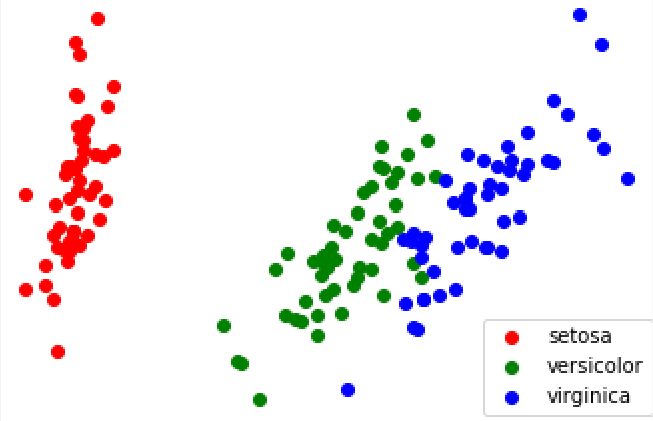
eigenvectors are principal components



```
In [3]: from sklearn.datasets import load_iris
        from sklearn.decomposition import PCA

        iris = load_iris()

        print (iris.data)
```

```
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
```





pca on movie ratings

R

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Bob	4	5	5	4	4
Ted	3	3	3	5	4
Ann	4	5	5	5	2

PCA

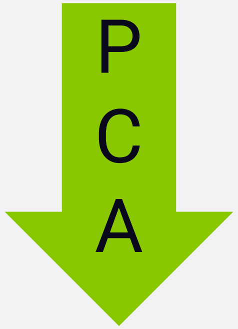
	"Action"	"Sci-Fi"	"Classic"
Bob	0.3	0.5	0.2
Ted	0.1	0.1	0.8
Ann	0.3	0.6	0.1

U

pca on movie ratings

R^T

	Bob	Ted	Ann
Indiana Jones	4	3	4
Star Wars	5	3	5
Empire Strikes Back	5	3	5
Incredibles	4	5	5
Casablanca	4	4	2



	"Action"	"Sci-Fi"	"Classic"
Indiana Jones	0.6	0.3	0.1
Star Wars	0.4	0.6	0
Empire Strikes Back	0.4	0.6	0
Incredibles	0.8	0.2	0
Casablanca	0.2	0	0.8

M

matrix factorization

$$R = U\Sigma M^T$$

singular value decomposition (svd)

but wait

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Bob	4	5	?	?	?
Ted	?	?	?	?	1
Ann	?	5	5	5	?

$$R = U \Sigma M^T$$

$$R_{Bob, Empire\ Strikes\ Back} = U_{Bob} \cdot M^T_{Empire\ Strikes\ Back}$$

stochastic gradient descent (sgd)

enough talk

code walkthrough

a matrix factorization bestiary

Non-Negative Matrix Factorization (NMF)

Probabilistic Matrix Factorization (PMF)

Probabilistic Latent Semantic Analysis (PLSA)

PureSVD

UV Decomposition

Weighted Regularized Matrix Factorization (WRMF)

SVD++

timeSVD++

HOSVD

CUR

Factorization Machines

Factorized Personalized Markov Chains

tuning svd

```
print("Searching for best parameters...")
param_grid = {'n_epochs': [20, 30], 'lr_all': [0.005, 0.010],
              'n_factors': [50, 100]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)

gs.fit(evaluationData)

# best RMSE score
print("Best RMSE score attained: ", gs.best_score['rmse'])

params = gs.best_params['rmse']
SVDtuned = SVD(n_epochs = params['n_epochs'], lr_all = params['lr_all'], n_factors
= params['n_factors'])
```

exercise

tune the hyperparameters for SVD with the MovieLens data set.

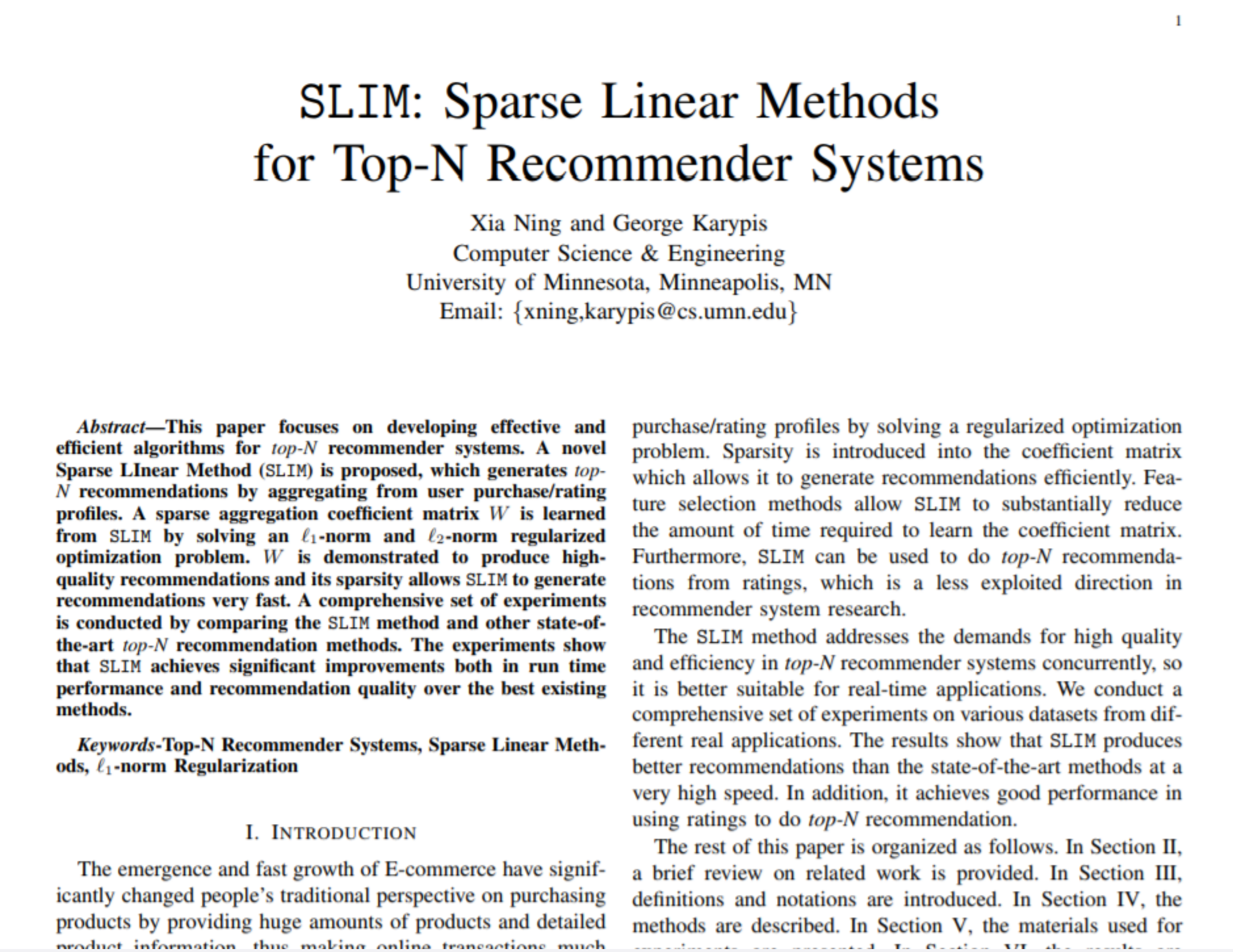
svd tuning results

{'n_epochs': 20, 'lr_all': 0.005, 'n_factors': 50}

Untuned	Tuned
RMSE: 0.9033	RMSE: 0.9002
Sixth Sense, The (1999)	Lord of the Rings: The Return of the King, The (2003)
Casablanca (1942)	Modern Times (1936)
Hamlet (1996)	Lord of the Rings: The Two Towers, The (2002)
Monty Python and the Holy Grail (1975)	Lord of the Rings: The Fellowship of the Ring, The (2001)
When We Were Kings (1996)	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)
It Happened One Night (1934)	Lawrence of Arabia (1962)
Bridge on the River Kwai, The (1957)	Departed, The (2006)
Smoke (1995)	Raging Bull (1980)
Big Night (1996)	Matrix, The (1999)
Seven Samurai (1954)	Singin' in the Rain (1952)

bleeding edge alert!

sparse linear methods (SLIM)



1

SLIM: Sparse Linear Methods for Top-N Recommender Systems

Xia Ning and George Karypis
Computer Science & Engineering
University of Minnesota, Minneapolis, MN
Email: {xning,karypis@cs.umn.edu}

***Abstract*—This paper focuses on developing effective and efficient algorithms for *top-N* recommender systems. A novel Sparse LInear Method (`SLIM`) is proposed, which generates *top-N* recommendations by aggregating from user purchase/rating profiles. A sparse aggregation coefficient matrix W is learned from `SLIM` by solving an ℓ_1-norm and ℓ_2-norm regularized optimization problem. W is demonstrated to produce high-quality recommendations and its sparsity allows `SLIM` to generate recommendations very fast. A comprehensive set of experiments is conducted by comparing the `SLIM` method and other state-of-the-art *top-N* recommendation methods. The experiments show that `SLIM` achieves significant improvements both in run time performance and recommendation quality over the best existing methods.**

***Keywords*-Top-N Recommender Systems, Sparse Linear Methods, ℓ_1-norm Regularization**

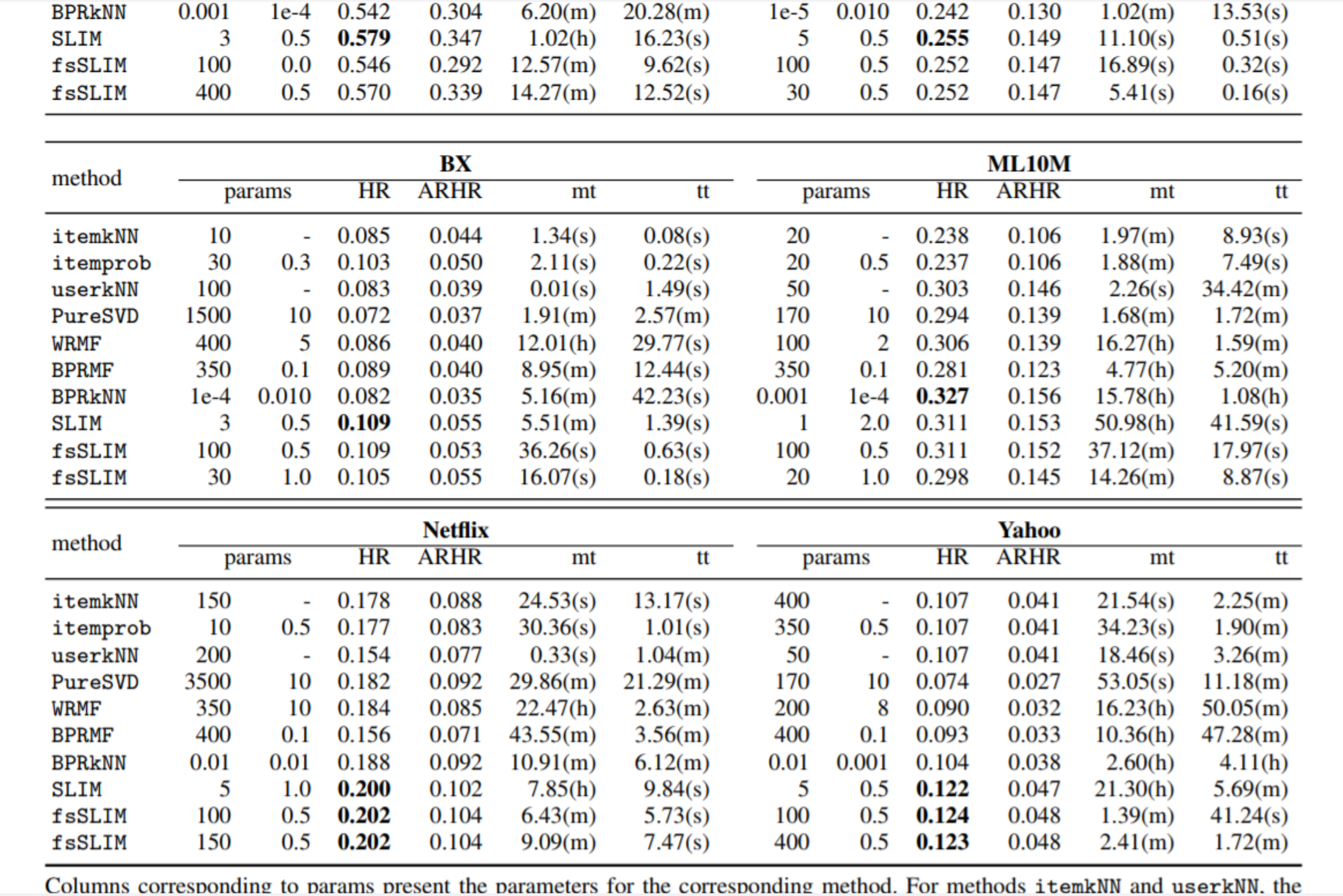
I. INTRODUCTION

The emergence and fast growth of E-commerce have significantly changed people's traditional perspective on purchasing products by providing huge amounts of products and detailed purchase/rating profiles by solving a regularized optimization problem. Sparsity is introduced into the coefficient matrix which allows it to generate recommendations efficiently. Feature selection methods allow `SLIM` to substantially reduce the amount of time required to learn the coefficient matrix. Furthermore, `SLIM` can be used to do *top-N* recommendations from ratings, which is a less exploited direction in recommender system research.

The `SLIM` method addresses the demands for high quality and efficiency in *top-N* recommender systems concurrently, so it is better suitable for real-time applications. We conduct a comprehensive set of experiments on various datasets from different real applications. The results show that `SLIM` produces better recommendations than the state-of-the-art methods at a very high speed. In addition, it achieves good performance in using ratings to do *top-N* recommendation.

The rest of this paper is organized as follows. In Section II, a brief review on related work is provided. In Section III, definitions and notations are introduced. In Section IV, the methods are described. In Section V, the materials used for

SLIM results



`BPRkNN`	0.001	1e-4	0.542	0.304	6.20(m)	20.28(m)	1e-5	0.010	0.242	0.130	1.02(m)	13.53(s)
`SLIM`	3	0.5	**0.579**	0.347	1.02(h)	16.23(s)	5	0.5	**0.255**	0.149	11.10(s)	0.51(s)
`fsSLIM`	100	0.0	0.546	0.292	12.57(m)	9.62(s)	100	0.5	0.252	0.147	16.89(s)	0.32(s)
`fsSLIM`	400	0.5	0.570	0.339	14.27(m)	12.52(s)	30	0.5	0.252	0.147	5.41(s)	0.16(s)

method	BX						ML10M					
	params		HR	ARHR	mt	tt	params		HR	ARHR	mt	tt
`itemkNN`	10	-	0.085	0.044	1.34(s)	0.08(s)	20	-	0.238	0.106	1.97(m)	8.93(s)
`itemprob`	30	0.3	0.103	0.050	2.11(s)	0.22(s)	20	0.5	0.237	0.106	1.88(m)	7.49(s)
`userkNN`	100	-	0.083	0.039	0.01(s)	1.49(s)	50	-	0.303	0.146	2.26(s)	34.42(m)
`PureSVD`	1500	10	0.072	0.037	1.91(m)	2.57(m)	170	10	0.294	0.139	1.68(m)	1.72(m)
`WRMF`	400	5	0.086	0.040	12.01(h)	29.77(s)	100	2	0.306	0.139	16.27(h)	1.59(m)
`BPRMF`	350	0.1	0.089	0.040	8.95(m)	12.44(s)	350	0.1	0.281	0.123	4.77(h)	5.20(m)
`BPRkNN`	1e-4	0.010	0.082	0.035	5.16(m)	42.23(s)	0.001	1e-4	**0.327**	0.156	15.78(h)	1.08(h)
`SLIM`	3	0.5	**0.109**	0.055	5.51(m)	1.39(s)	1	2.0	0.311	0.153	50.98(h)	41.59(s)
`fsSLIM`	100	0.5	0.109	0.053	36.26(s)	0.63(s)	100	0.5	0.311	0.152	37.12(m)	17.97(s)
`fsSLIM`	30	1.0	0.105	0.055	16.07(s)	0.18(s)	20	1.0	0.298	0.145	14.26(m)	8.87(s)

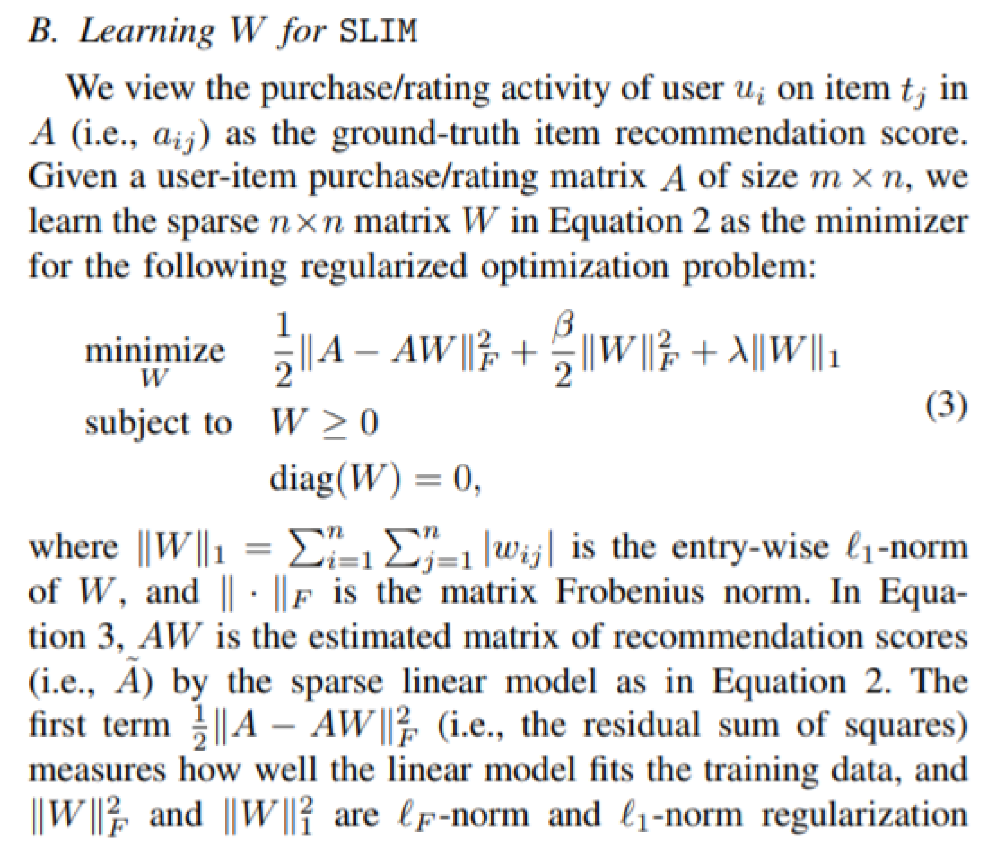
method	Netflix						Yahoo					
	params		HR	ARHR	mt	tt	params		HR	ARHR	mt	tt
`itemkNN`	150	-	0.178	0.088	24.53(s)	13.17(s)	400	-	0.107	0.041	21.54(s)	2.25(m)
`itemprob`	10	0.5	0.177	0.083	30.36(s)	1.01(s)	350	0.5	0.107	0.041	34.23(s)	1.90(m)
`userkNN`	200	-	0.154	0.077	0.33(s)	1.04(m)	50	-	0.107	0.041	18.46(s)	3.26(m)
`PureSVD`	3500	10	0.182	0.092	29.86(m)	21.29(m)	170	10	0.074	0.027	53.05(s)	11.18(m)
`WRMF`	350	10	0.184	0.085	22.47(h)	2.63(m)	200	8	0.090	0.032	16.23(h)	50.05(m)
`BPRMF`	400	0.1	0.156	0.071	43.55(m)	3.56(m)	400	0.1	0.093	0.033	10.36(h)	47.28(m)
`BPRkNN`	0.01	0.01	0.188	0.092	10.91(m)	6.12(m)	0.01	0.001	0.104	0.038	2.60(h)	4.11(h)
`SLIM`	5	1.0	**0.200**	0.102	7.85(h)	9.84(s)	5	0.5	**0.122**	0.047	21.30(h)	5.69(m)
`fsSLIM`	100	0.5	**0.202**	0.104	6.43(m)	5.73(s)	100	0.5	**0.124**	0.048	1.39(m)	41.24(s)
`fsSLIM`	150	0.5	**0.202**	0.104	9.09(m)	7.47(s)	400	0.5	**0.123**	0.048	2.41(m)	1.72(m)

Columns corresponding to params present the parameters for the corresponding method. For methods `itemkNN` and `userkNN`, the

how SLIM works

$$\tilde{a}_{ij} = a_i^T w_j$$

$$\tilde{A} = AW$$



B. *Learning W for* SLIM

We view the purchase/rating activity of user u_i on item t_j in A (i.e., a_{ij}) as the ground-truth item recommendation score. Given a user-item purchase/rating matrix A of size $m \times n$, we learn the sparse $n \times n$ matrix W in Equation 2 as the minimizer for the following regularized optimization problem:

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0 \\ & \operatorname{diag}(W) = 0, \end{aligned} \tag{3}$$

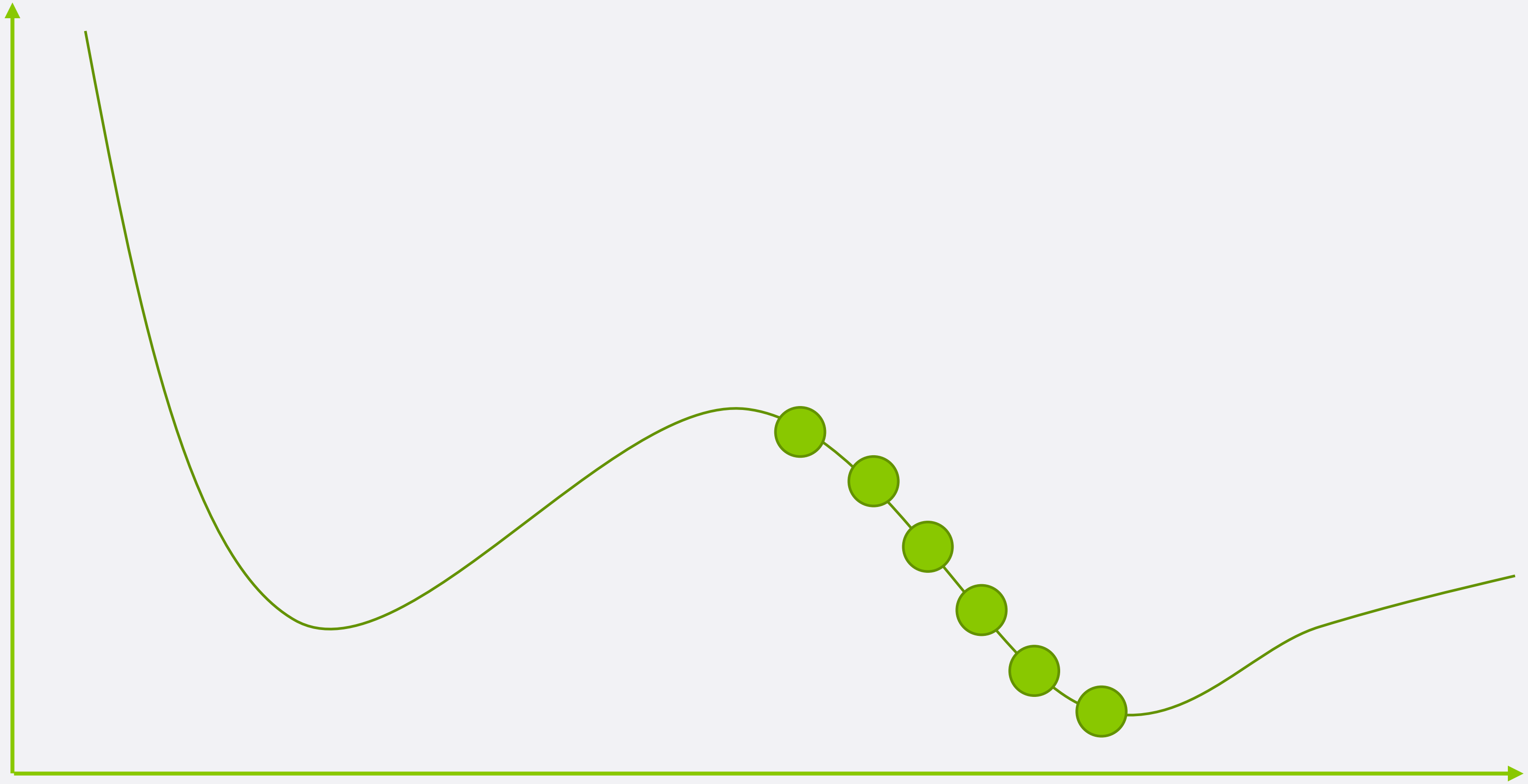
where $\|W\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}|$ is the entry-wise ℓ_1-norm of W, and $\|\cdot\|_F$ is the matrix Frobenius norm. In Equation 3, AW is the estimated matrix of recommendation scores (i.e., $\tilde{A}$) by the sparse linear model as in Equation 2. The first term $\frac{1}{2}\|A - AW\|_F^2$ (i.e., the residual sum of squares) measures how well the linear model fits the training data, and $\|W\|_F^2$ and $\|W\|_1^2$ are ℓ_F-norm and ℓ_1-norm regularization

recommendations with deep learning

intro to deep learning

deep learning pre-requisites

gradient descent



autodiff

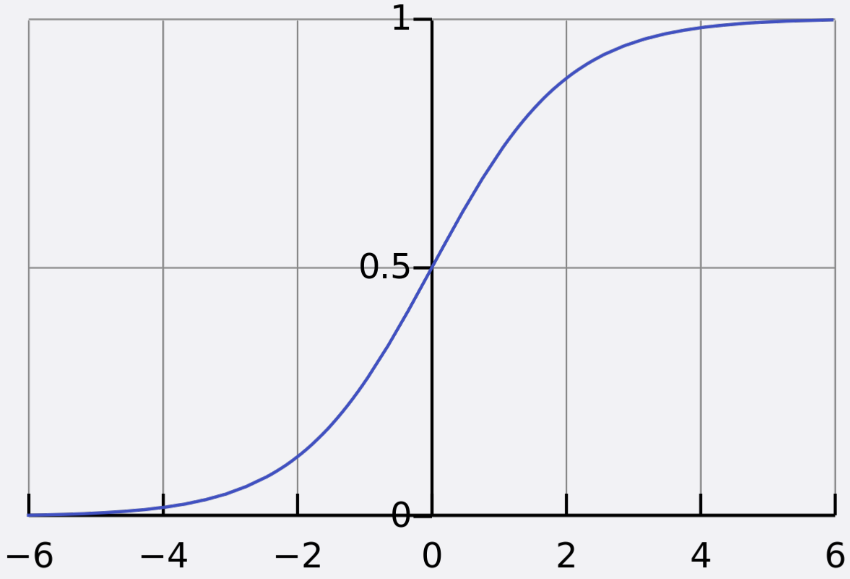
- Gradient descent requires knowledge of, well, the gradient from your cost function (MSE)
- Mathematically we need the first partial derivatives of all the inputs
  - This is hard and inefficient if you just throw calculus at the problem
- Reverse-mode autodiff to the rescue!
  - Optimized for many inputs + few outputs (like a neuron)
  - Computes all partial derivatives in # of outputs + 1 graph traversals
  - Still fundamentally a calculus trick – it's complicated but it works
  - This is what Tensorflow uses

softmax

- Used for classification
  - Given a score for each class
  - It produces a probability of each class
  - The class with the highest probability is the “answer” you get

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)},$$

x is a vector of input values
theta is a vector of weights



in review

- Gradient descent is an algorithm for minimizing error over multiple steps
- Autodiff is a calculus trick for finding the gradients in gradient descent
- Softmax is a function for choosing the most probable classification given several input values

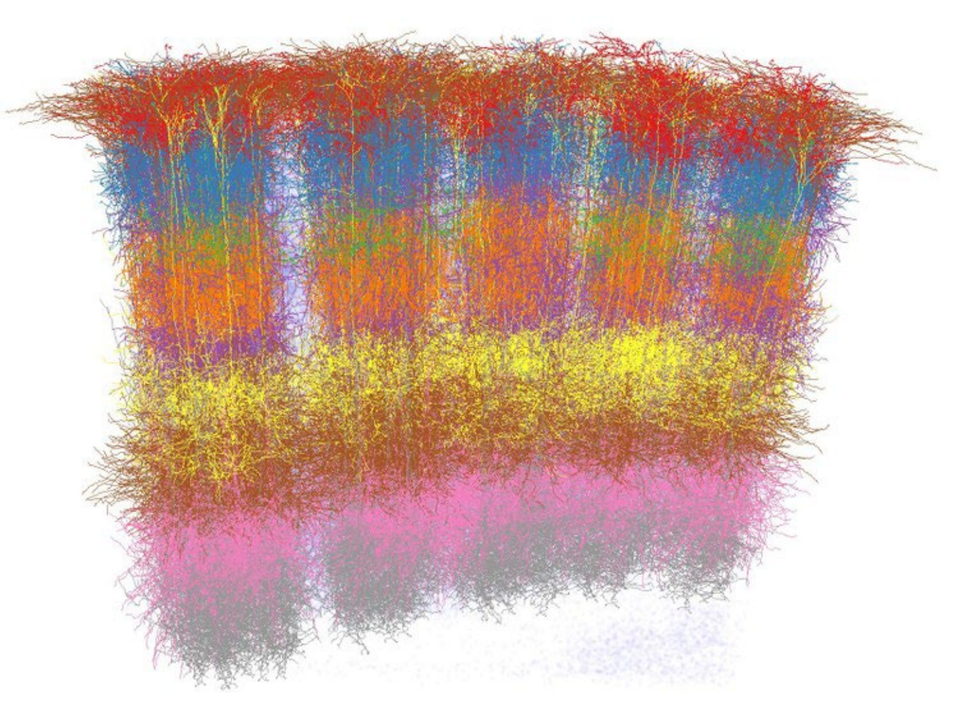
introducing artificial neural networks

the biological inspiration

- Neurons in your cerebral cortex are connected via axons
- A neuron “fires” to the neurons it’s connected to, when enough of its input signals are activated.
- Very simple at the individual neuron level – but layers of neurons connected in this way can yield learning behavior.
- Billions of neurons, each with thousands of connections, yields a mind

cortical columns

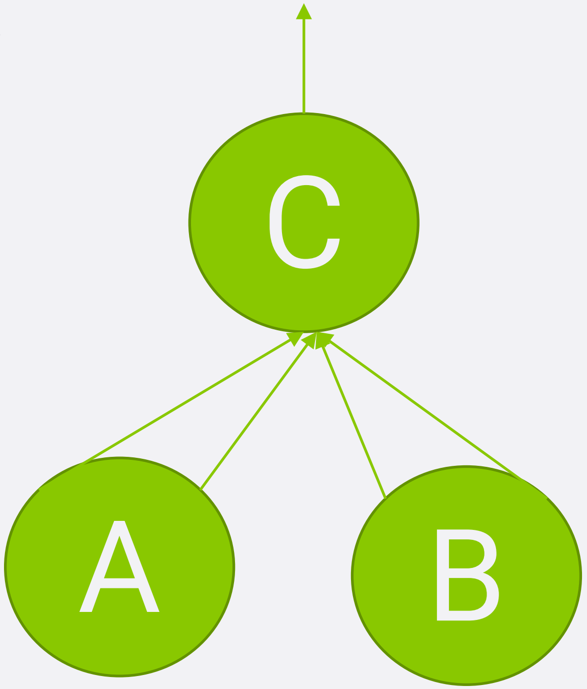
- Neurons in your cortex seem to be arranged into many stacks, or “columns” that process information in parallel
- “mini-columns” of around 100 neurons are organized into larger “hyper-columns”. There are 100 million mini-columns in your cortex
- This is coincidentally similar to how GPU’s work…



(credit: Marcel Oberlaender et al.)

the first artificial neurons

- 1943!!



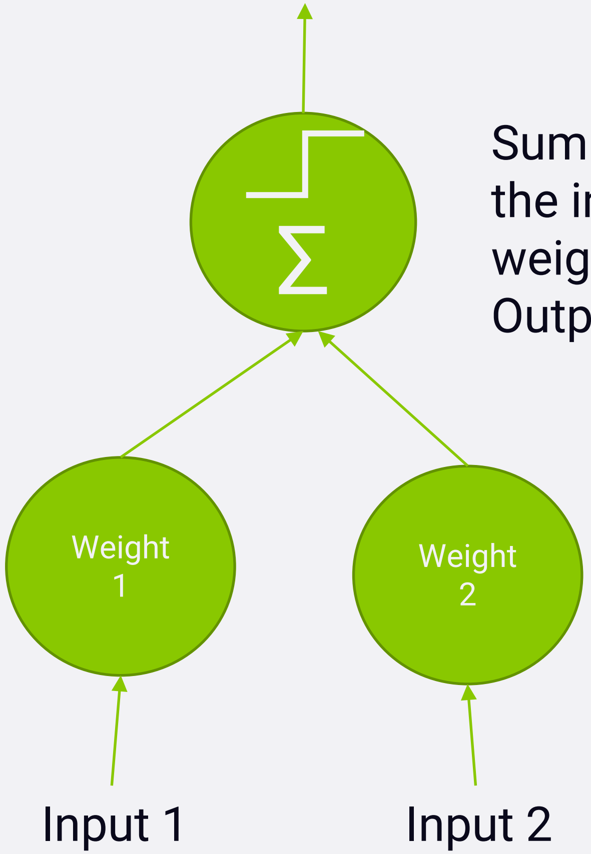
An artificial neuron "fires" if more than N input connections are active.

Depending on the number of connections from each input neuron, and whether a connection activates or suppresses a neuron, you can construct AND, OR, and NOT logical constructs this way.

This example would implement C = A OR B if the threshold is 2 inputs being active.

the linear threshold unit (ltu)

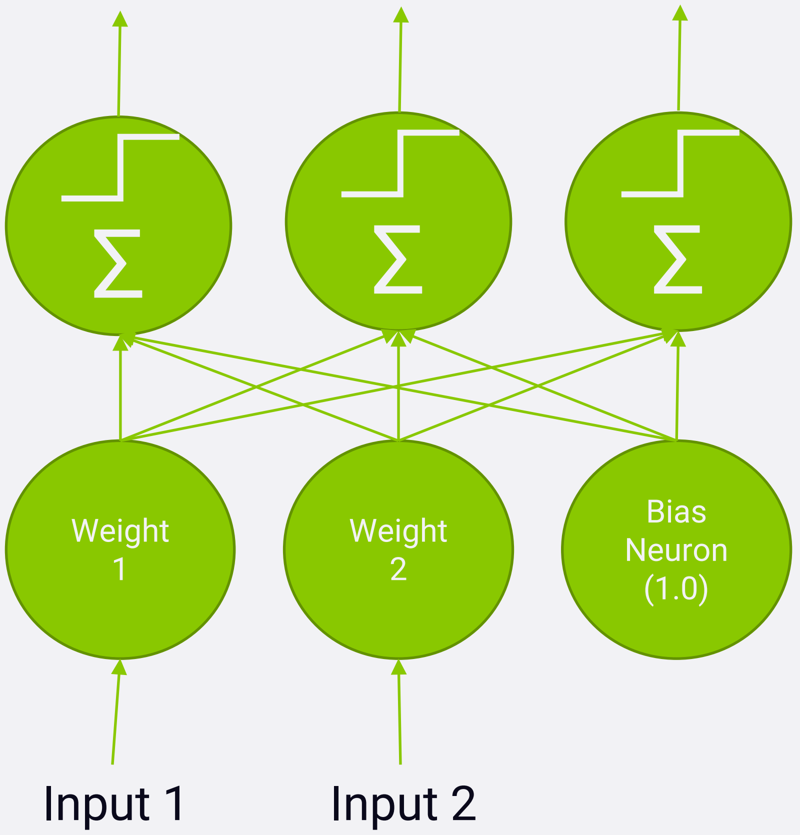
- 1957!
- Adds weights to the inputs; output is given by a step function


Σ
Weight 1
Weight 2
Input 1
Input 2

Sum up the products of the inputs and their weights
Output 1 if sum is >= 0

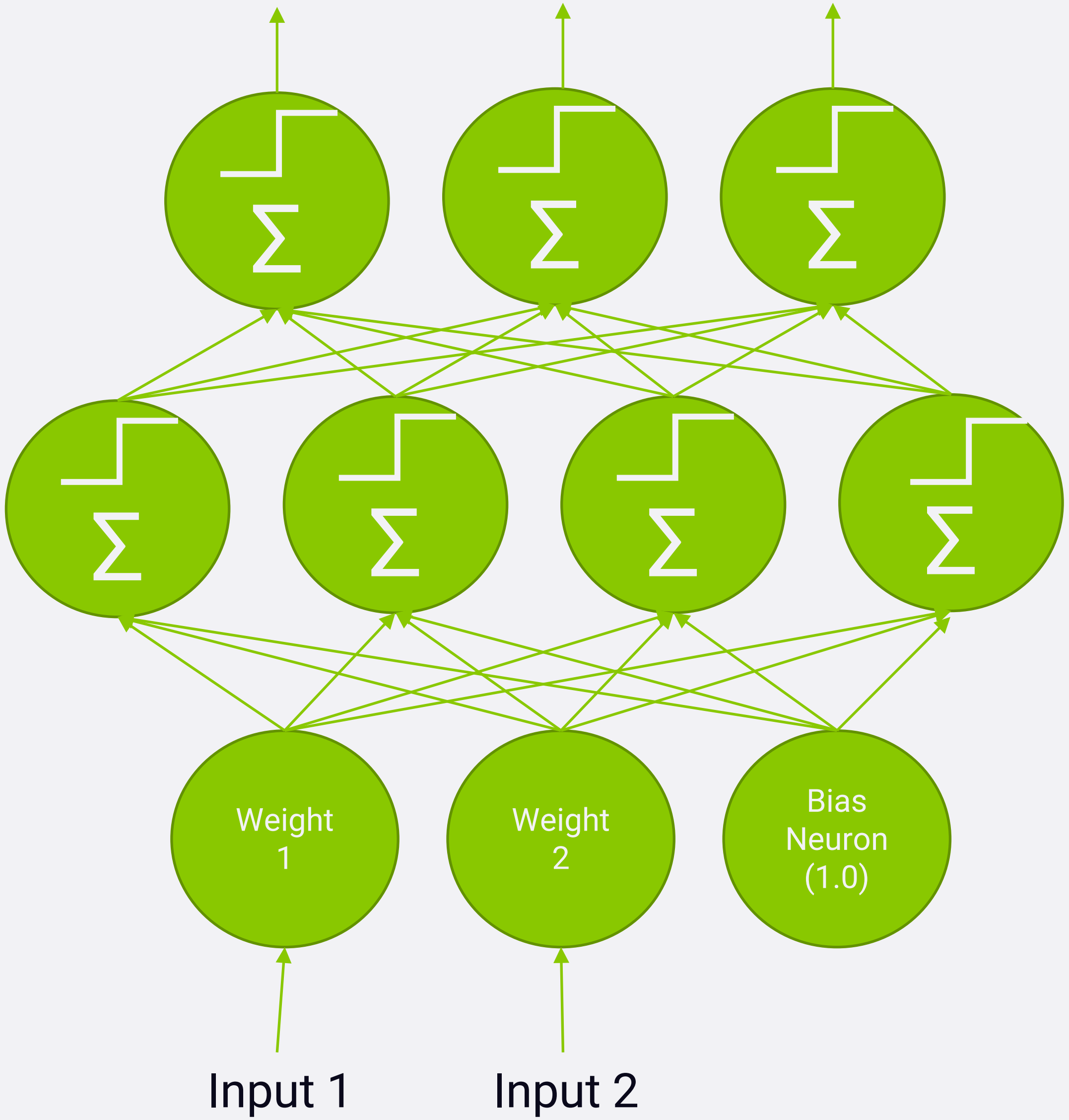
the perceptron

- A layer of LTU’s
- A perceptron can learn by reinforcing weights that lead to correct behavior during training
- This too has a biological basis (“cells that fire together, wire together”)


Σ
Σ
Σ
Weight 1
Weight 2
Bias Neuron (1.0)
Input 1
Input 2

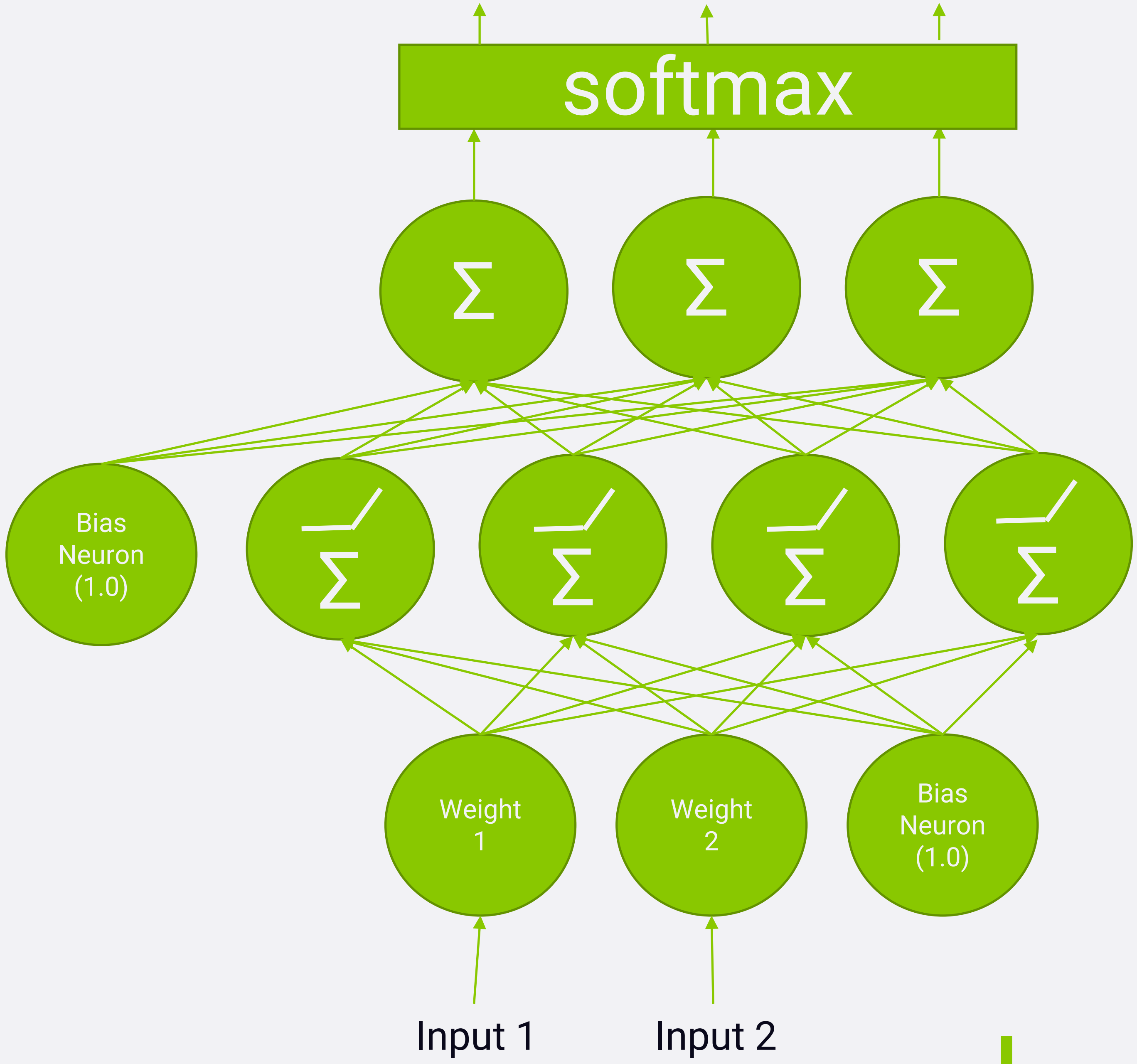
multi-layer perceptrons

- Addition of “hidden layers”
- This is a Deep Neural Network
- Training them is trickier – but we’ll talk about that.


Σ
Weight 1
Weight 2
Bias Neuron (1.0)
Input 1
Input 2

a modern deep neural network

- Replace step activation function with something better
- Apply softmax to the output
- Training using gradient descent

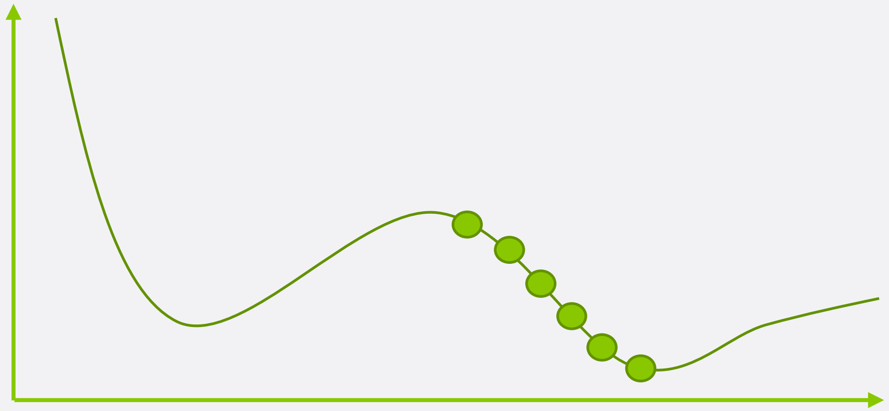

softmax
Σ
Σ
Σ
Bias Neuron (1.0)
Σ
Σ
Σ
Σ
Weight 1
Weight 2
Bias Neuron (1.0)
Input 1
Input 2

let’s play

deep learning

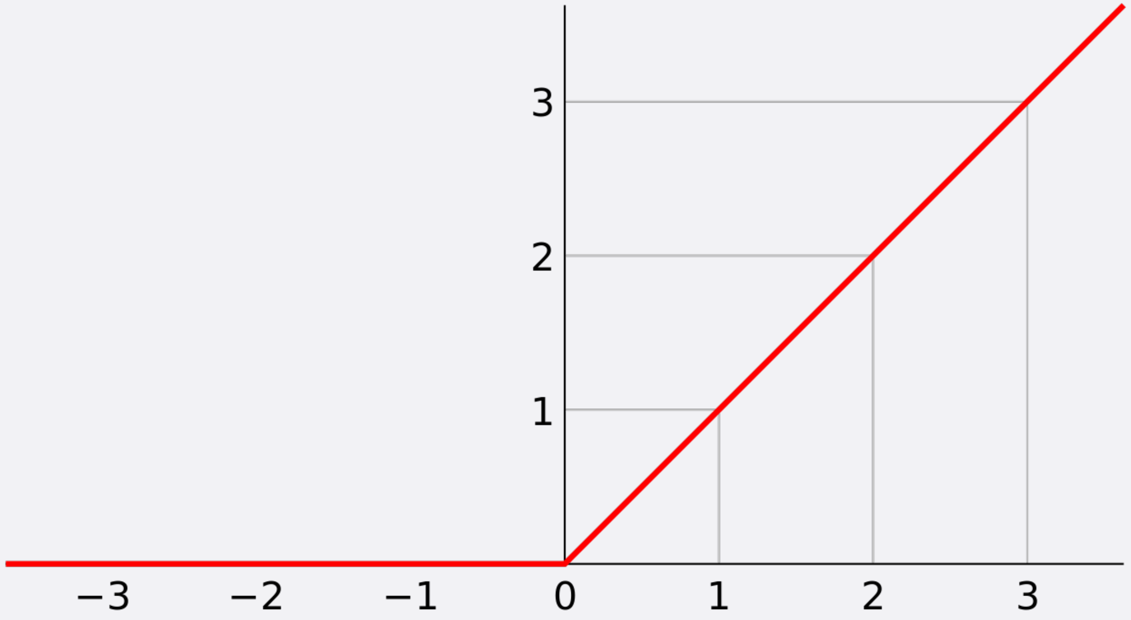
backpropagation

- How do you train a MLP’s weights? How does it **learn**?
- Backpropagation… or more specifically: Gradient Descent using reverse-mode autodiff!
- For each training step:
  - Compute the output error
  - Compute how much each neuron in the previous hidden layer contributed
  - Back-propagate that error in a reverse pass
  - Tweak weights to reduce the error using gradient descent



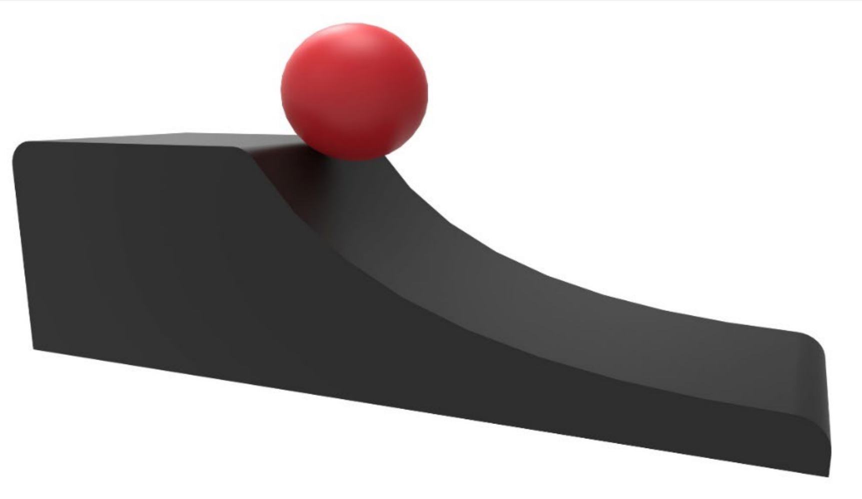
activation functions
(aka rectifier)

- Step functions don’t work with gradient descent – there is no gradient!
  - Mathematically, they have no useful derivative.
- Alternatives:
  - Logistic function
  - Hyperbolic tangent function
  - Exponential linear unit (ELU)
  - ReLU function (Rectified Linear Unit)
- ReLU is common. Fast to compute and works well.
  - Also: “Leaky ReLU”, “Noisy ReLU”
  - ELU can sometimes lead to faster learning though.



ReLU function

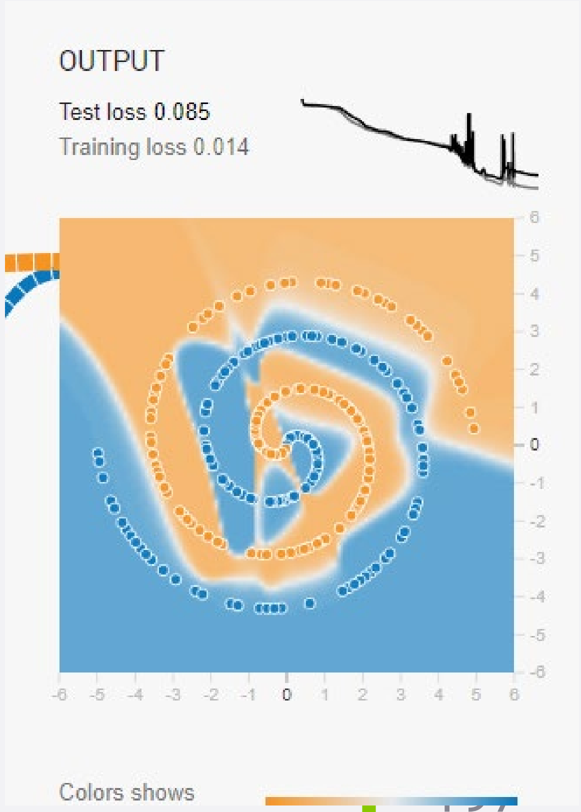
optimization functions



- There are faster (as in faster learning) optimizers than gradient descent
  - Momentum Optimization
    - Introduces a momentum term to the descent, so it slows down as things start to flatten and speeds up as the slope is steep
  - Nesterov Accelerated Gradient
    - A small tweak on momentum optimization – computes momentum based on the gradient slightly ahead of you, not where you are
  - RMSProp
    - Adaptive learning rate to help point toward the minimum
  - Adam
    - Adaptive moment estimation – momentum + RMSProp combined
    - Popular choice today, easy to use

avoiding **overfitting**

- With thousands of weights to tune, overfitting is a problem
- Early stopping (when performance starts dropping)
- Regularization terms added to cost function during training
- Dropout – ignore say 50% of all neurons randomly at each training step
  - Works surprisingly well!
  - Forces your model to spread out its learning



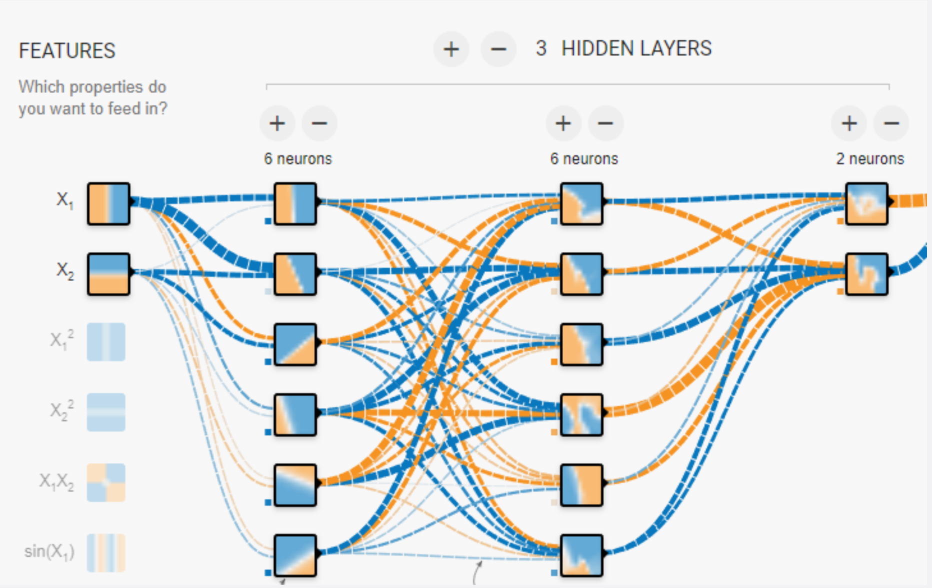
OUTPUT

Test loss 0.085

Training loss 0.014

tuning your topology

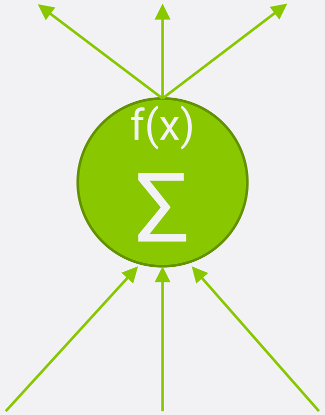
- Trial & error is one way
  - Evaluate a smaller network with less neurons in the hidden layers
  - Evaluate a larger network with more layers
    - Try reducing the size of each layer as you progress – form a funnel
- More layers can yield faster learning
- Or just use more layers and neurons than you need, and don't care because you use early stopping.
- Use "model zoos"


FEATURES
Which properties do you want to feed in?
+ − 3 HIDDEN LAYERS
6 neurons
6 neurons
2 neurons
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$

activation functions

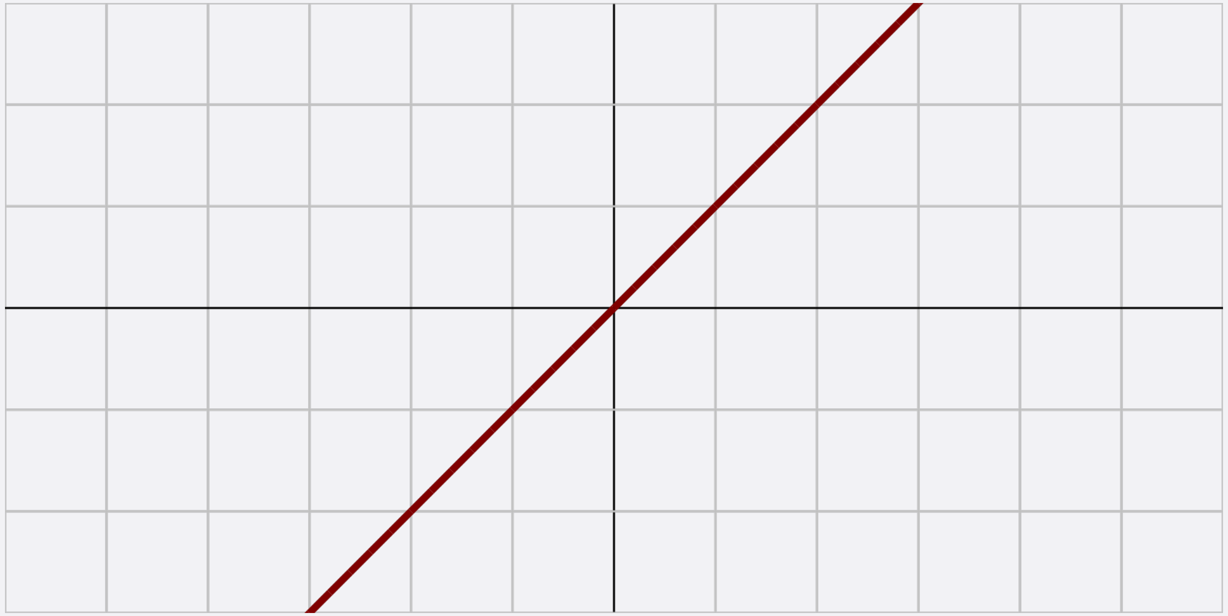
activation functions

- Define the output of a node / neuron given its input signals



linear activation function

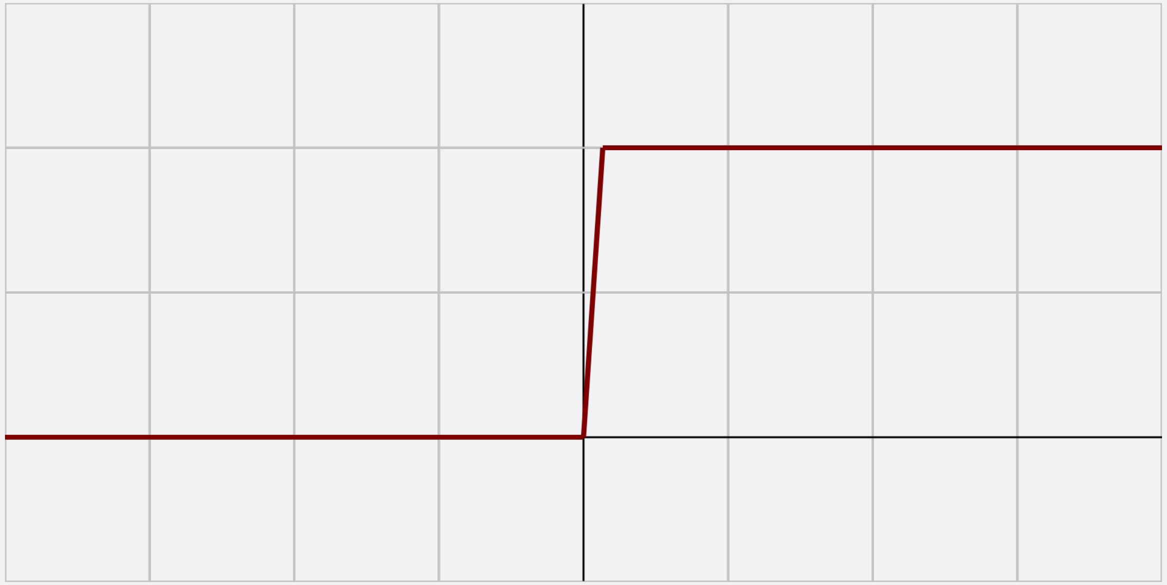
- It doesn't really *do* anything
- Can't do backpropagation



By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920411

binary step function

- It's on or off
- Can't handle multiple classification – it's binary after all
- Vertical slopes don't work well with calculus!



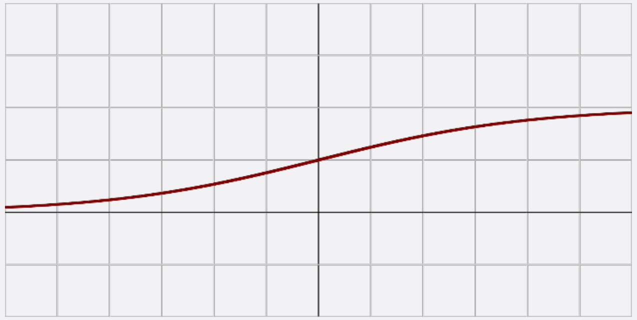
By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920435

instead we need non-linear activation functions

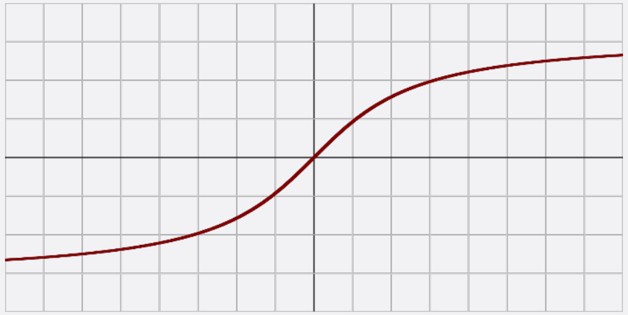
- These can create complex mappings between inputs and outputs
- Allow backpropagation (because they have a useful derivative)
- Allow for multiple layers (linear functions degenerate to a single layer)

Sigmoid / Logistic / TanH

- Nice & smooth
- Scales everything from 0-1 (Sigmoid / Logistic) or -1 to 1 (tanh / hyperbolic tangent)
- But: changes slowly for high or low values
  - The "Vanishing Gradient" problem
- Computationally expensive
- Tanh generally preferred over sigmoid



Sigmoid AKA Logistic

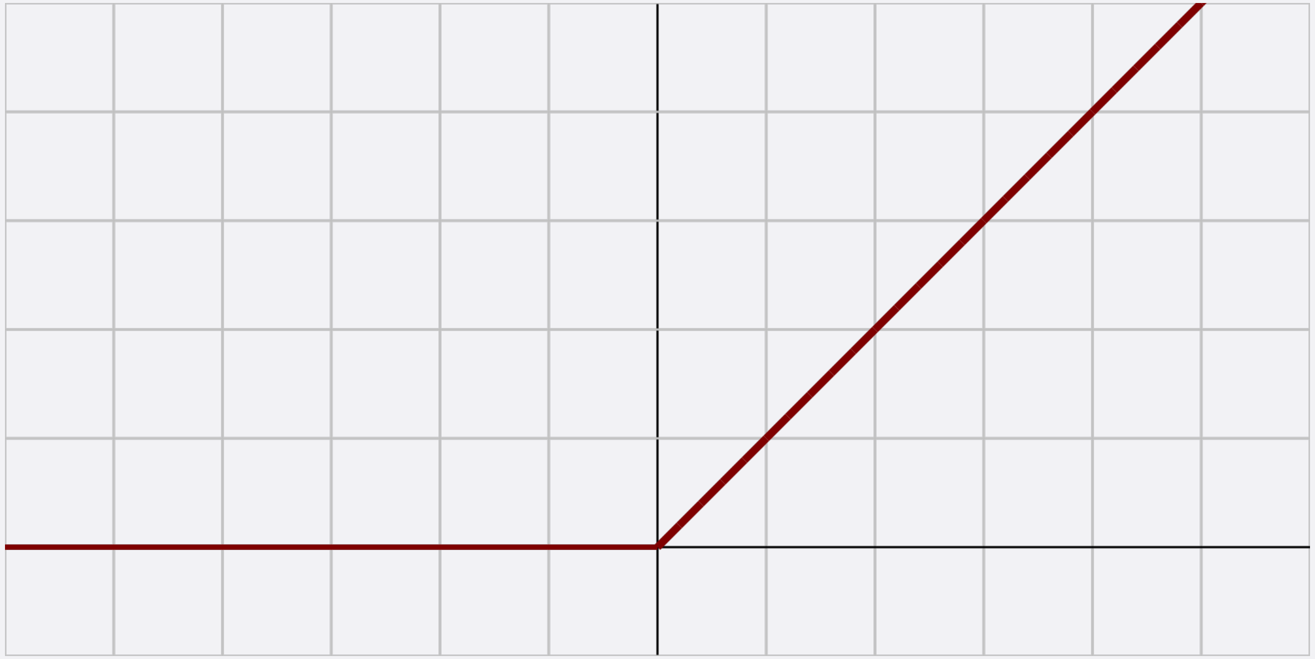


TanH AKA Hyperbolic Tangent

By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920533

Rectified Linear Unit (ReLU)

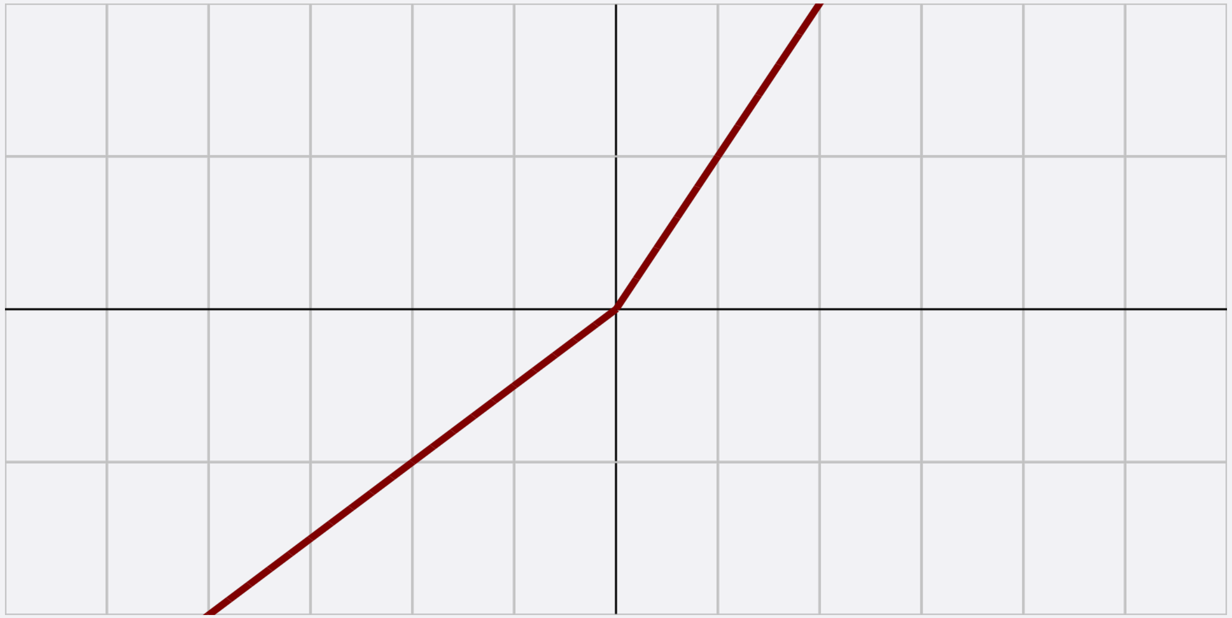
- Now we’re talking
- Very popular choice
- Easy & fast to compute
- But, when inputs are zero or negative, we have a linear function and all of its problems
  - The “Dying ReLU problem”



By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920600

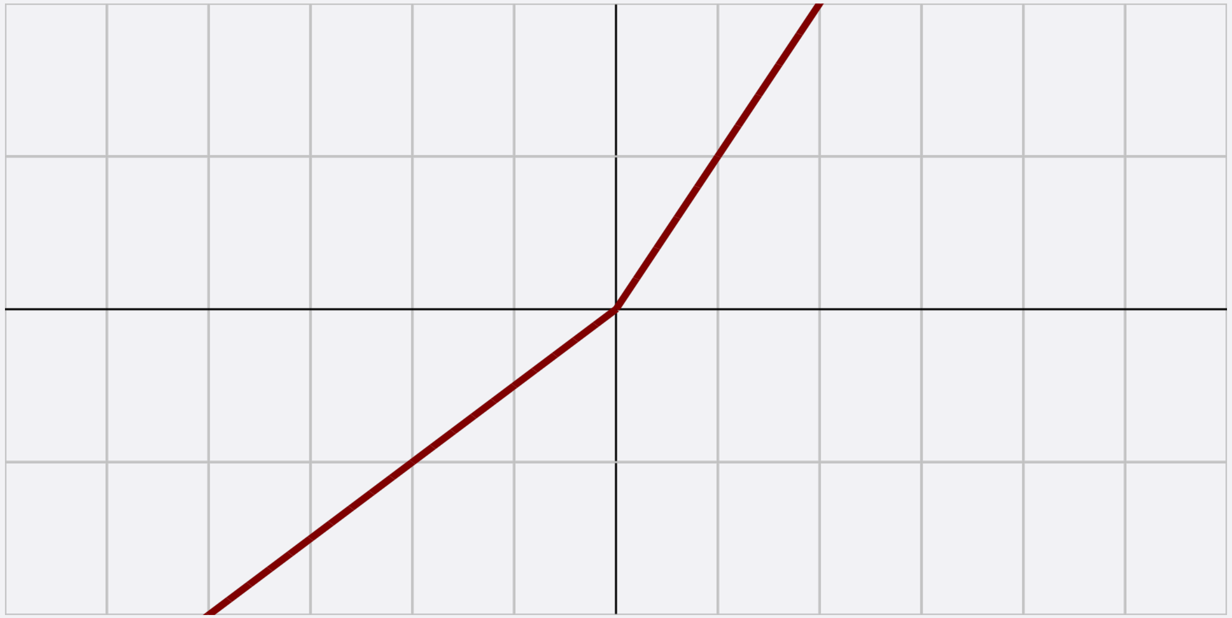
Leaky ReLU

- Solves "dying ReLU" by introducing a negative slope below 0 (usually not as steep as this)



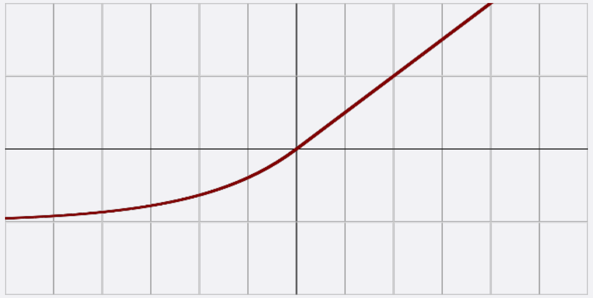
Parametric ReLU (PReLU)

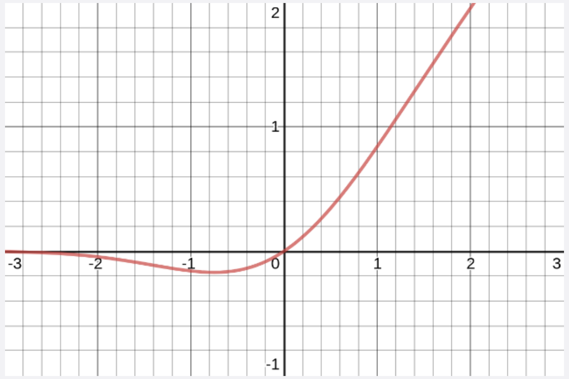
- ReLU, but the slope in the negative part is learned via backpropagation
- Complicated and YMMV



Other ReLU variants

- Exponential Linear Unit (ELU)
- Swish
  - From Google, performs really well
  - Mostly a benefit with very deep networks (40+ layers)
- Maxout
  - Outputs the max of the inputs
  - Technically ReLU is a special case of maxout
  - But doubles parameters that need to be trained, not often practical.

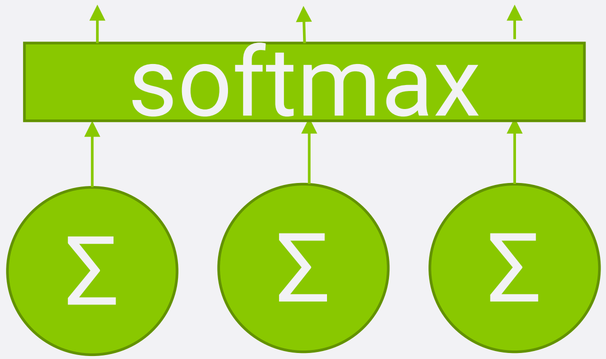




By Ringdongling - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=85402414

Softmax

- Used on the final output layer of a multiple classification problem
- Basically converts outputs to probabilities of each classification
- Can't produce more than one label for something (sigmoid can)
- Don't worry about the actual function for the exam, just know what it's used for.



Choosing an activation function

- For multiple classification, use softmax on the output layer
- RNN's do well with Tanh
- For everything else
  - Start with ReLU
  - If you need to do better, try Leaky ReLU
  - Last resort: PReLU, Maxout
  - Swish for really deep networks

tensorflow

why tensorflow?

- It’s not specifically for neural networks– it’s more generally an architecture for executing a graph of numerical operations
- Tensorflow can optimize the processing of that graph, and distribute its processing across a network
  - Sounds a lot like Apache Spark, eh?
- It can also distribute work across GPU’s!
  - Can handle massive scale – it was made by Google
- Runs on about anything
- Highly efficient C++ code with easy to use Python API’s

tensorflow basics

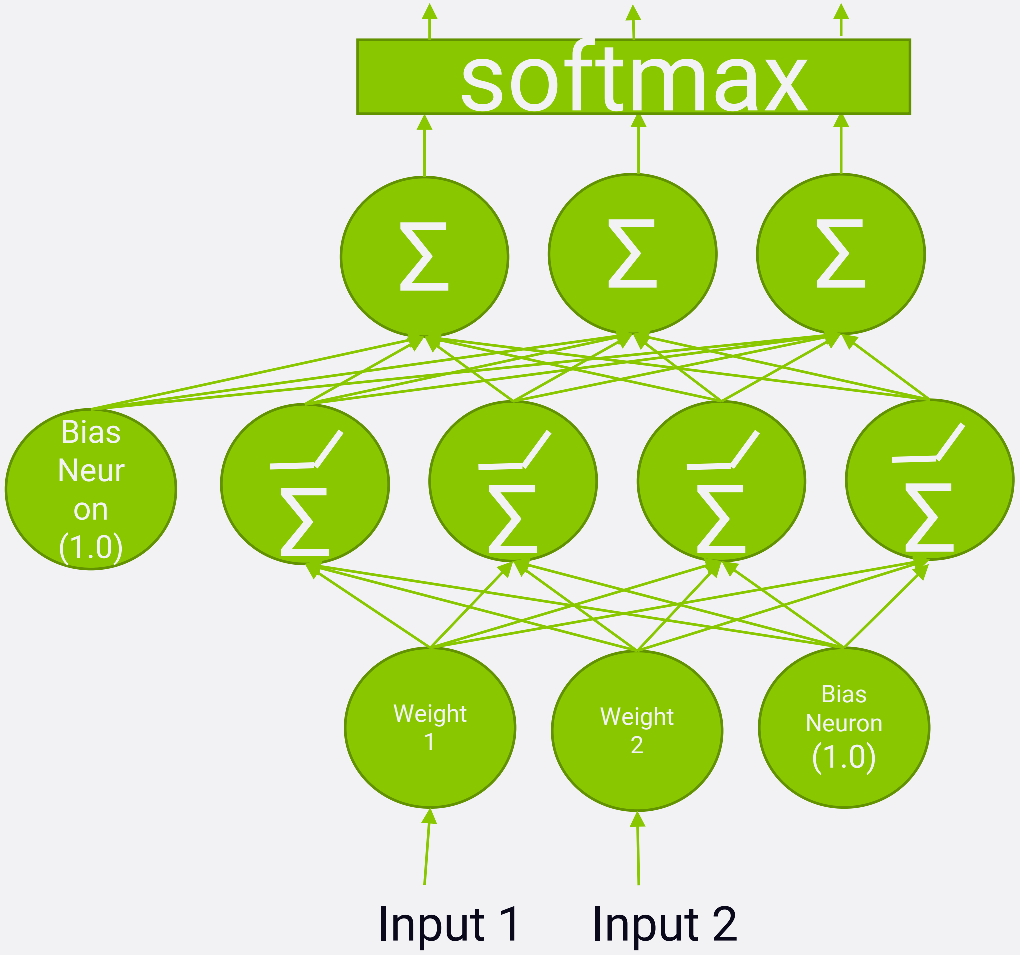
- Install with conda install tensorflow or conda install tensorflow-gpu
- A tensor is just a fancy name for an array or matrix of values
- To use Tensorflow, you:
  - Construct a graph to compute your tensors
  - Initialize your variables
  - Execute that graph – nothing actually happens until then

World's simplest Tensorflow app:

```
import tensorflow as tf

a = tf.Variable(1, name="a")
b = tf.Variable(2, name="b")
f = a + b

tf.print(f)
```

creating a neural network with tensorflow

- Mathematical insights:
  - All those interconnected arrows multiplying weights can be thought of as a big matrix multiplication
  - The bias term can just be added onto the result of that matrix multiplication
- So in Tensorflow, we can define a layer of a neural network as:
  `output = tf.matmul(previous_layer, layer_weights) + layer_biases`
- By using Tensorflow directly we're kinda doing this the "hard way."


softmax
Σ
Σ
Σ
Bias Neur on (1.0)
Σ
Σ
Σ
Σ
Weight 1
Weight 2
Bias Neuron (1.0)
Input 1
Input 2

creating a neural network with tensorflow

- Load up our training and testing data
- Construct a graph describing our neural network
  - Use **placeholders** for the input data and target labels
    - This way we can use the same graph for training and testing!
  - Use **variables** for the learned weights for each connection and learned biases for each neuron
    - Variables are preserved across runs within a Tensorflow session
- Associate an optimizer (ie gradient descent) to the network
- Run the optimizer with your training data
- Evaluate your trained network with your testing data

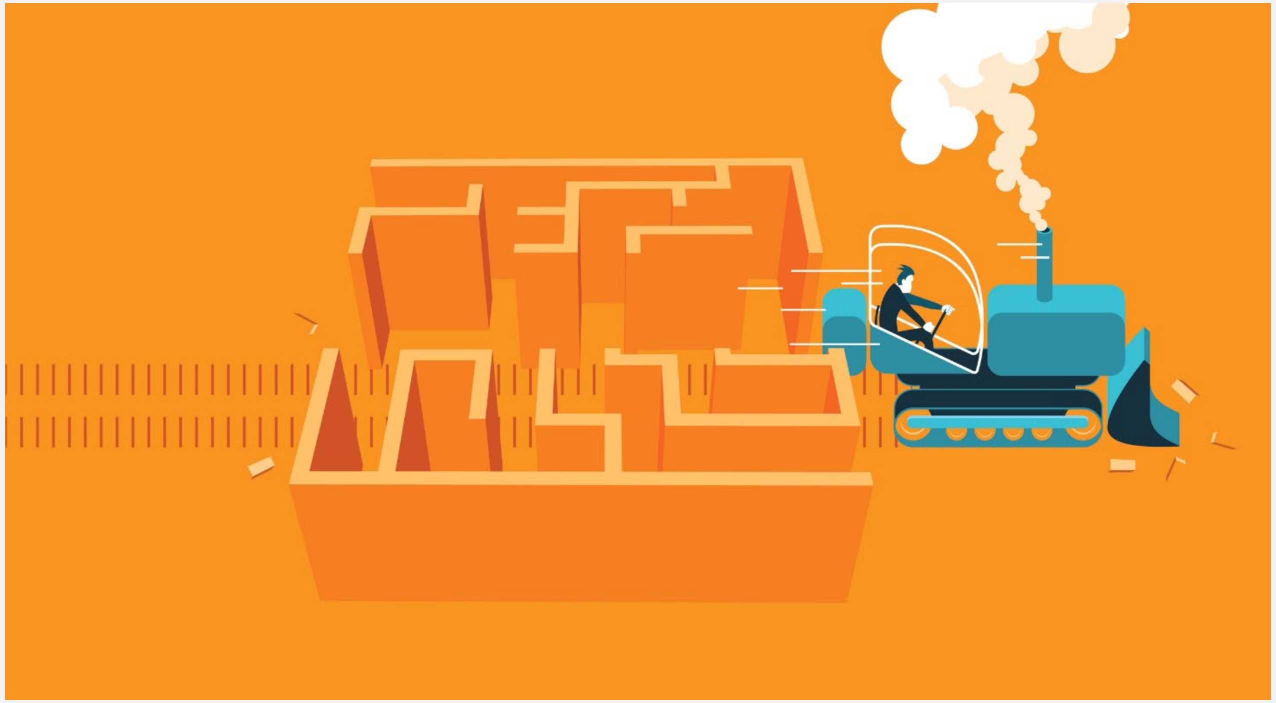
make sure your features are normalized

- Neural networks usually work best if your input data is normalized.
  - That is, 0 mean and unit variance
  - The real goal is that every input feature is comparable in terms of magnitude
- scikit_learn's StandardScaler can do this for you
- Many data sets are normalized to begin with – such as the one we're about to use.

let’s try it out

keras

why keras?



- Easy and fast prototyping
  - Runs on top of TensorFlow (or CNTK, or Theano)
  - scikit_learn integration
  - Less to think about – which often yields better results without even trying
  - This is really important! The faster you can experiment, the better your results.

let’s dive in

example: multi-class classification

- MNIST is an example of multi-class classification.

```
model = Sequential()

model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,
       nesterov=True)
model.compile(loss='categorical_crossentropy',
       optimizer=sgd, metrics=['accuracy'])
```

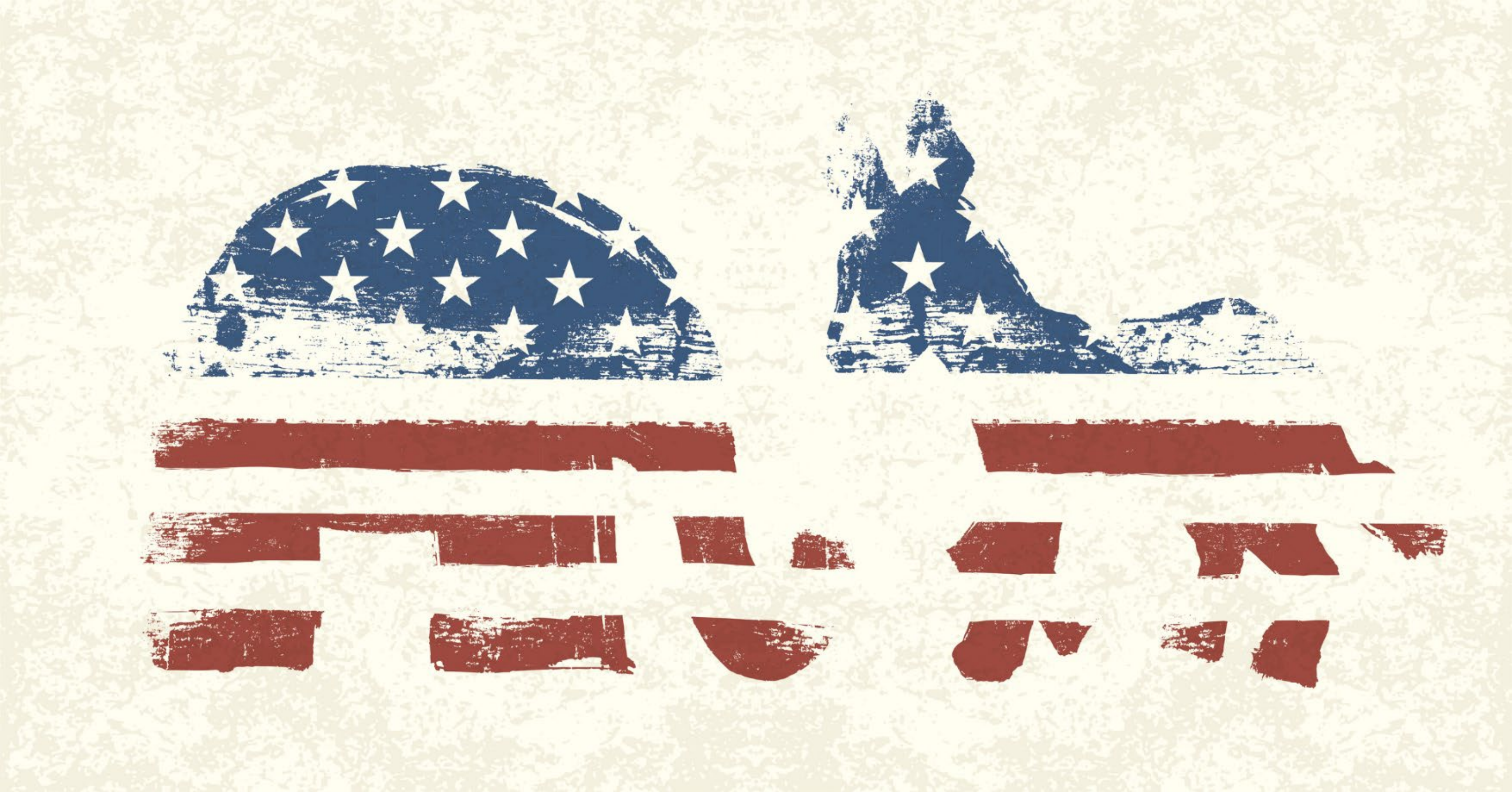
example: binary classification

```
model = Sequential()
model.add(Dense(64, input_dim=20,
activation='relu')) model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
optimizer='rmsprop',  metrics=['accuracy'])
```

integrating keras with scikit-learn

```
from keras.wrappers.scikit_learn import KerasClassifier

def create_model():
    model = Sequential()
    model.add(Dense(6, input_dim=4, kernel_initializer='normal', activation='relu'))
    model.add(Dense(4, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    return model

estimator = KerasClassifier(build_fn=create_model, nb_epoch=100, verbose=0)

cv_scores = cross_val_score(estimator, features, labels, cv=10)
print(cv_scores.mean())
```

let’s try it out

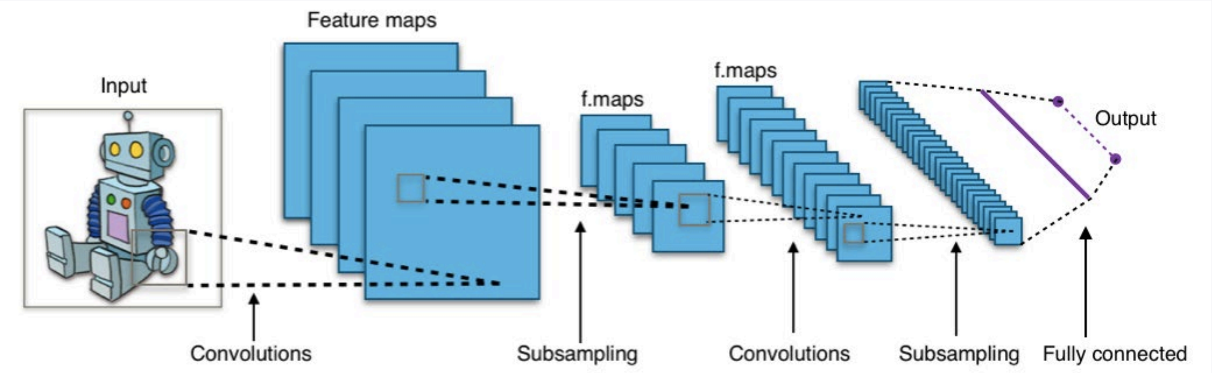


convolutional neural networks

cnn's: what are they for?

- When you have data that doesn't neatly align into columns
  - Images that you want to find features within
  - Machine translation
  - Sentence classification
  - Sentiment analysis
- They can find features that aren't in a specific spot
  - Like a stop sign in a picture
  - Or words within a sentence
- They are "feature-location invariant"

cnn's: how do they work?


Feature maps
Input
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected

- Inspired by the biology of the visual cortex
  - Local receptive fields are groups of neurons that only respond to a part of what your eyes see (subsampling)
  - They overlap each other to cover the entire visual field (convolutions)
  - They feed into higher layers that identify increasingly complex images
    - Some receptive fields identify horizontal lines, lines at different angles, etc. (filters)
    - These would feed into a layer that identifies shapes
    - Which might feed into a layer that identifies objects
  - For color images, extra layers for red, green, and blue

how do we “know” that’s a stop sign?

- Individual local receptive fields scan the image looking for edges, and pick up the edges of the stop sign in a layer
- Those edges in turn get picked up by a higher level convolution that identifies the stop sign’s shape (and letters, too)
- This shape then gets matched against your pattern of what a stop sign looks like, also using the strong red signal coming from your red layers
- That information keeps getting processed upward until your foot hits the brake!
- A CNN works the same way

cnn's with keras

- Source data must be of appropriate dimensions
  - ie width x length x color channels
- Conv2D layer type does the actual convolution on a 2D image
  - Conv1D and Conv3D also available – doesn't have to be image data
- MaxPooling2D layers can be used to reduce a 2D layer down by taking the maximum value in a given block
- Flatten layers will convert the 2D layer to a 1D layer for passing into a flat hidden layer of neurons
- Typical usage:
  - Conv2D -> MaxPooling2D -> Dropout -> Flatten -> Dense -> Dropout -> Softmax

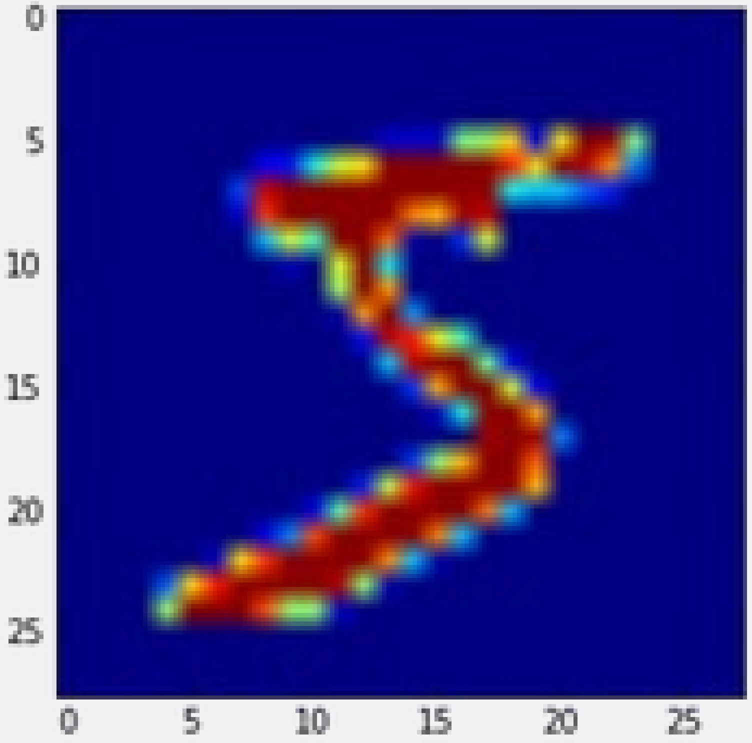
cnn's are hard

- Very resource-intensive (CPU, GPU, and RAM)
- Lots of hyperparameters
  - Kernel sizes, many layers with different numbers of units, amount of pooling… in addition to the usual stuff like number of layers, choice of optimizer
- Getting the training data is often the hardest part! (As well as storing and accessing it)

specialized cnn architectures

- Defines specific arrangement of layers, padding, and hyperparameters
- LeNet-5
  - Good for handwriting recognition
- AlexNet
  - Image classification, deeper than LeNet
- GoogLeNet
  - Even deeper, but with better performance
  - Introduces *inception modules* (groups of convolution layers)
- ResNet (Residual Network)
  - Even deeper – maintains performance via *skip connections*.

let's try it out

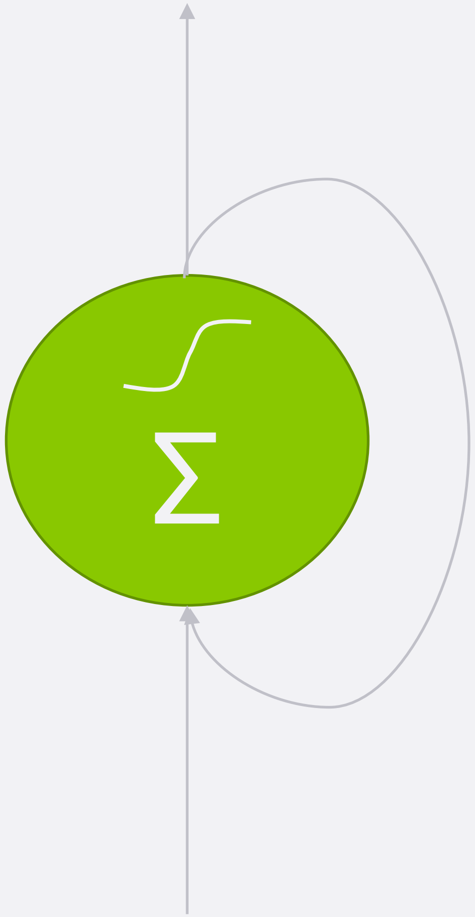


recurrent neural networks

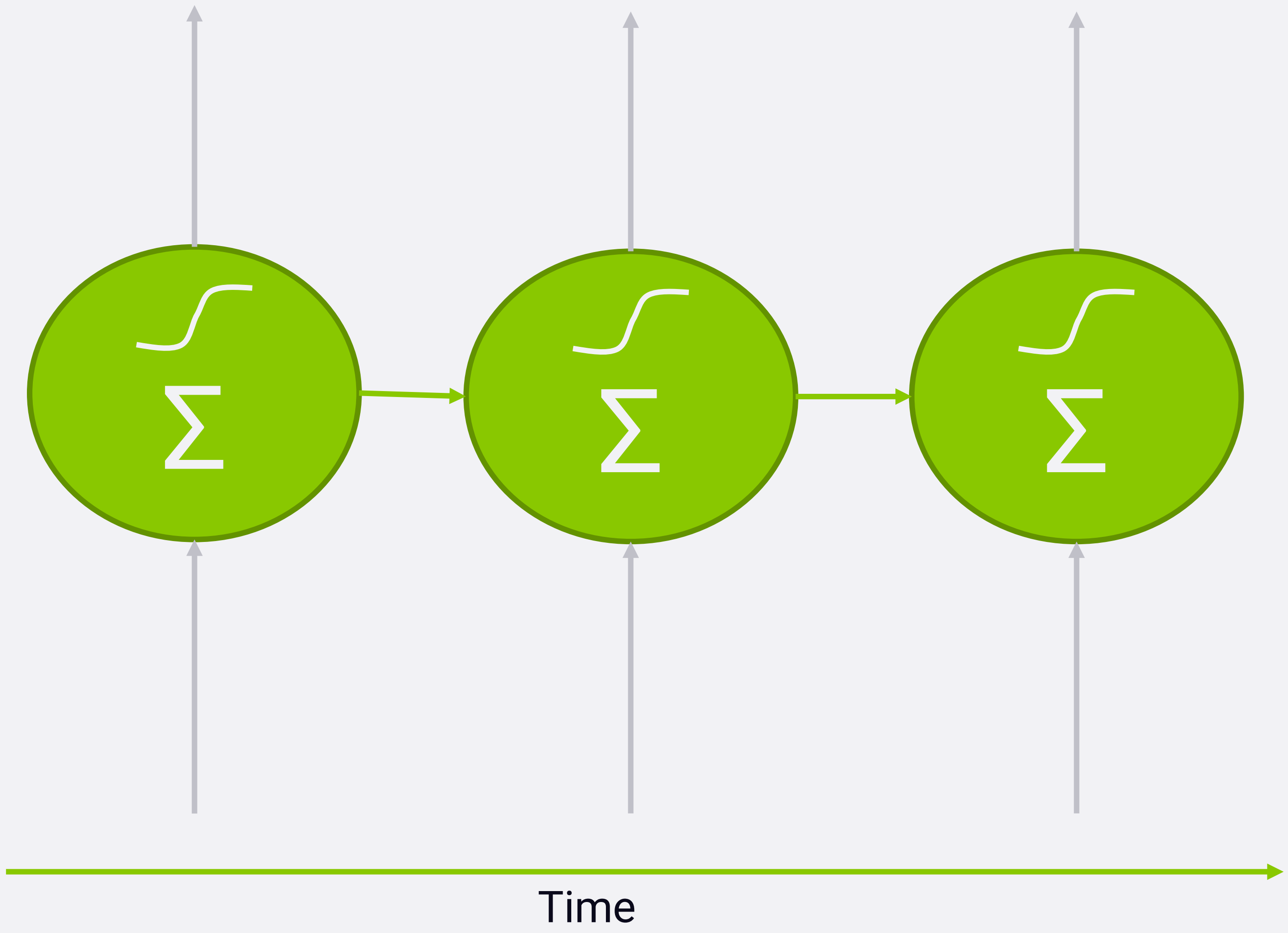
rnn's: what are they for?

- Time-series data
  - When you want to predict future behavior based on past behavior
  - Web logs, sensor logs, stock trades
  - Where to drive your self-driving car based on past trajectories
- Data that consists of sequences of arbitrary length
  - Machine translation
  - Image captions
  - Machine-generated music

a recurrent neuron

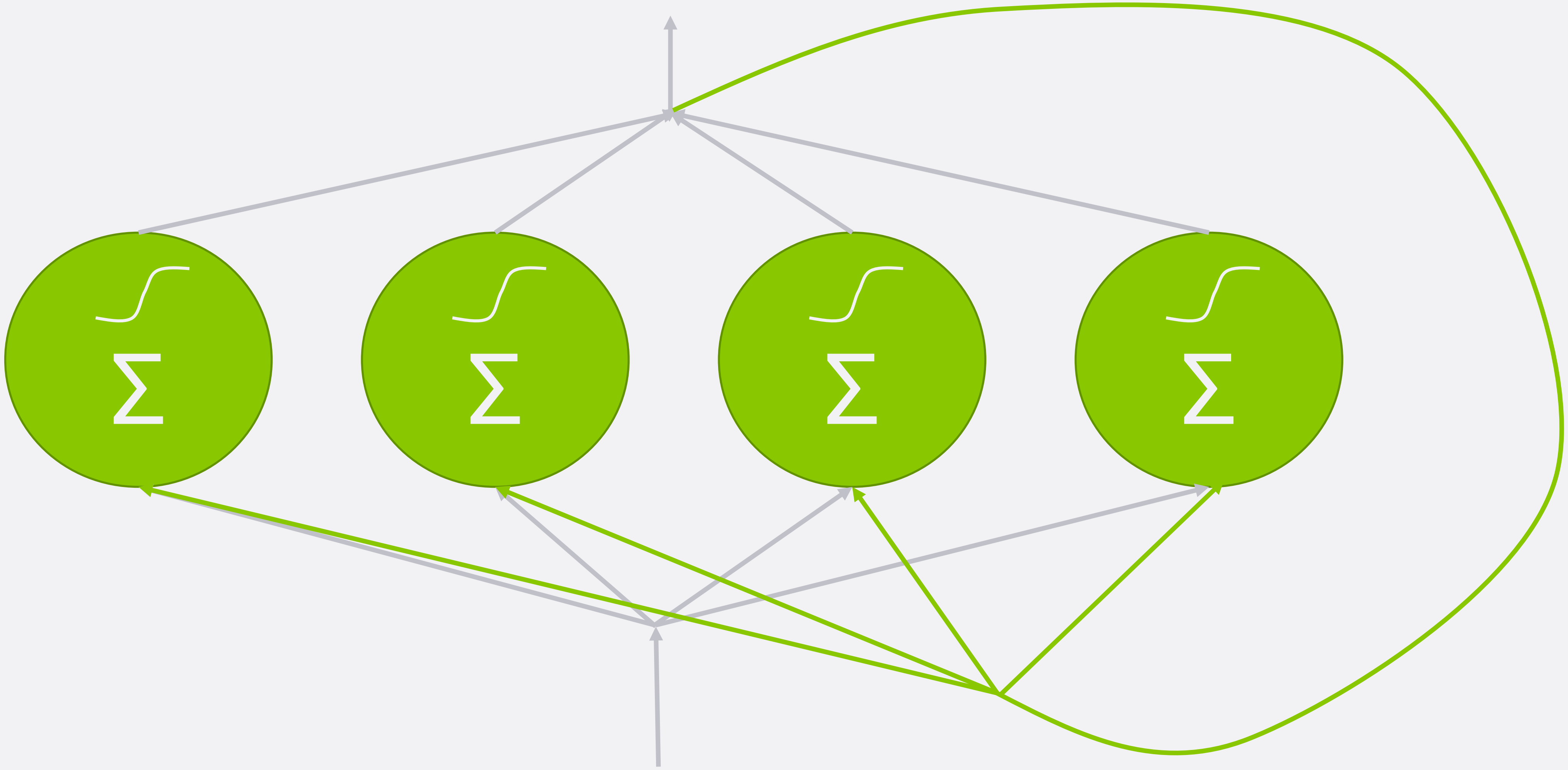


another way to look at it



A “Memory Cell”

a layer of recurrent neurons



rnn topologies

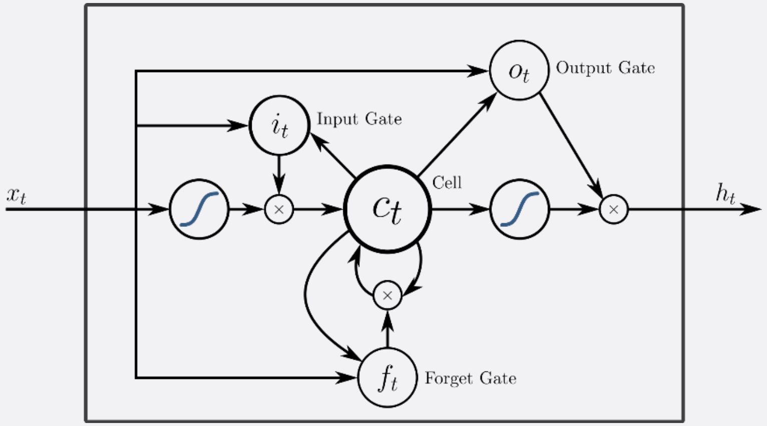
- Sequence to sequence
  - i.e., predict stock prices based on series of historical data
- Sequence to vector
  - i.e., words in a sentence to sentiment
- Vector to sequence
  - i.e., create captions from an image
- Encoder -> Decoder
  - Sequence -> vector -> sequence
  - i.e., machine translation

training rnn's

- Backpropagation through time
  - Just like backpropagation on MLP's, but applied to each time step.
- All those time steps add up fast
  - Ends up looking like a really, really deep neural network.
  - Can limit backpropagation to a limited number of time steps (truncated backpropagation through time)

training rnn's

- State from earlier time steps get diluted over time
    - This can be a problem, for example when learning sentence structures
- LSTM Cell
    - Long Short-Term Memory Cell
    - Maintains separate short-term and long-term states
- GRU Cell
    - Gated Recurrent Unit
    - Simplified LSTM Cell that performs about as well


x_t
o_t
Output Gate
i_t
Input Gate
c_t
Cell
h_t
f_t
Forget Gate

training rnn’s

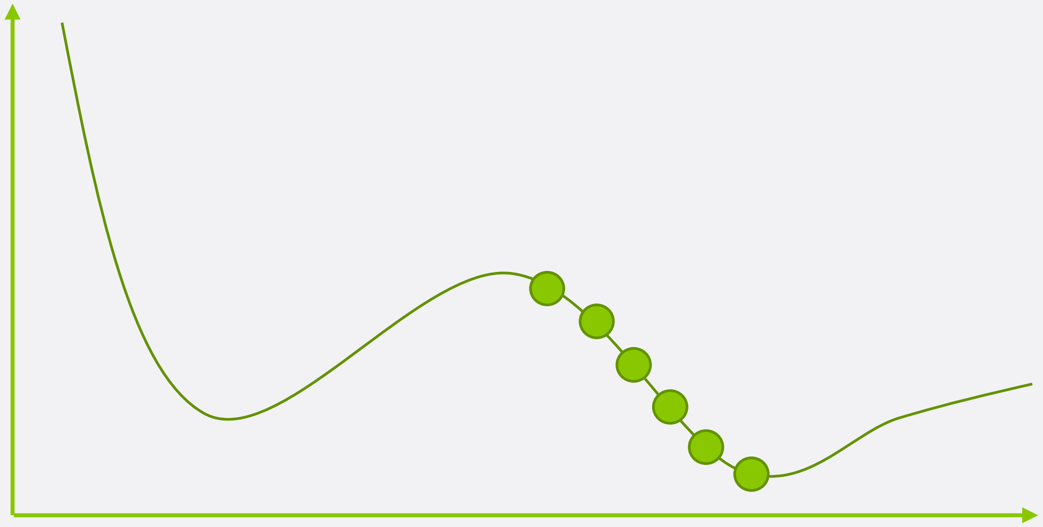
- It’s really hard
  - Very sensitive to topologies, choice of hyperparameters
  - Very resource intensive
  - A wrong choice can lead to a RNN that doesn’t converge at all.

let’s run an example

tuning neural networks

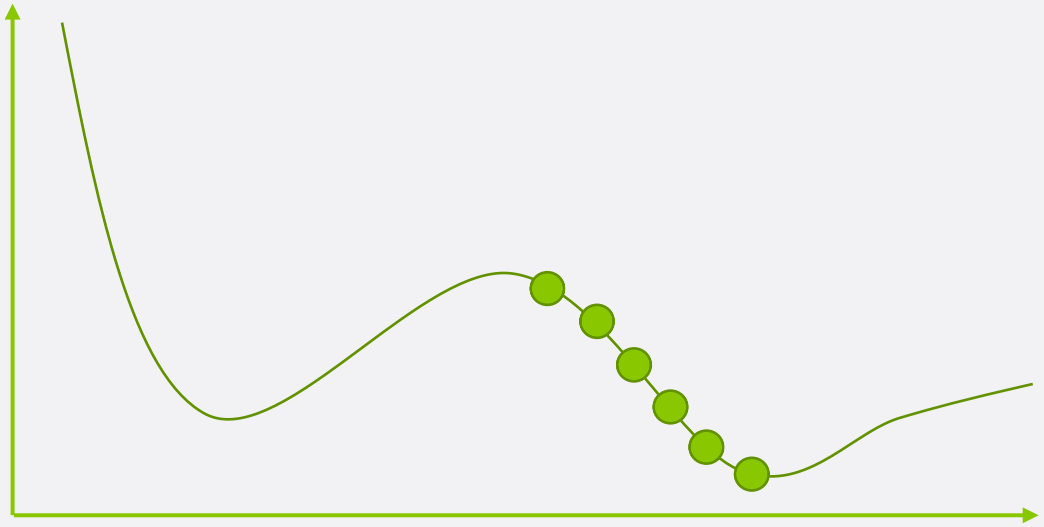
Learning Rate

- Neural networks are trained by gradient descent (or similar means)
- We start at some random point, and sample different solutions (weights) seeking to minimize some cost function, over many *epochs*
- How far apart these samples are is the *learning rate*



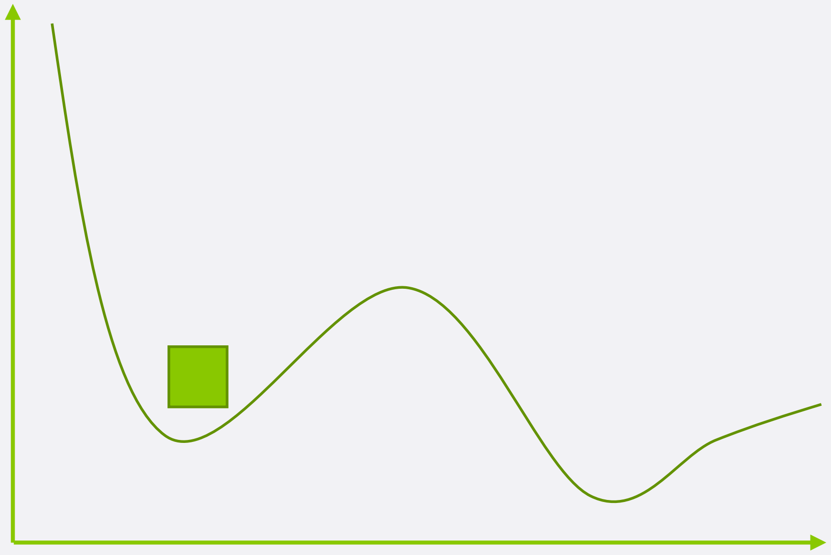
Effect of learning rate

- Too high a learning rate means you might overshoot the optimal solution!
- Too small a learning rate will take too long to find the optimal solution
- Learning rate is an example of a *hyperparameter*



Batch Size

- How many training samples are used within each epoch
- Somewhat counter-intuitively:
  - Smaller batch sizes can work their way out of "local minima" more easily
  - Batch sizes that are too large can end up getting stuck in the wrong solution
  - Random shuffling at each epoch can make this look like very inconsistent results from run to run



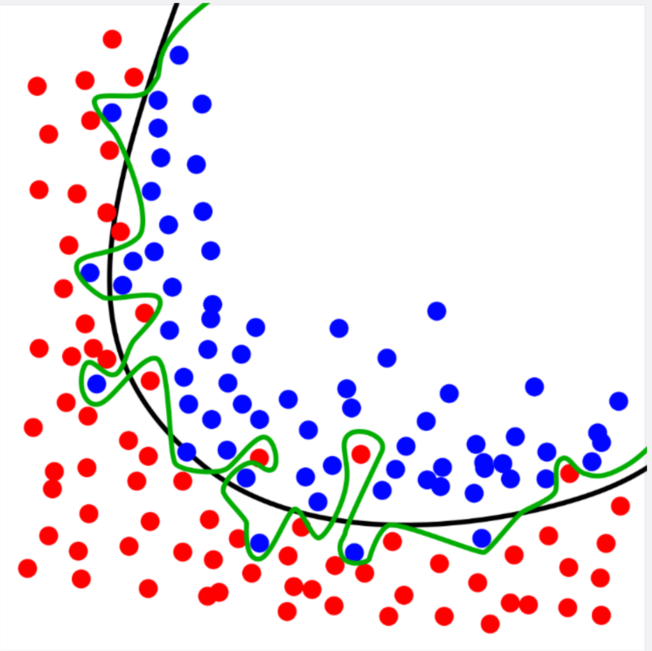
To Recap

- Small batch sizes tend to not get stuck in local minima
- Large batch sizes can converge on the wrong solution at random
- Large learning rates can overshoot the correct solution
- Small learning rates increase training time

neural network regularization

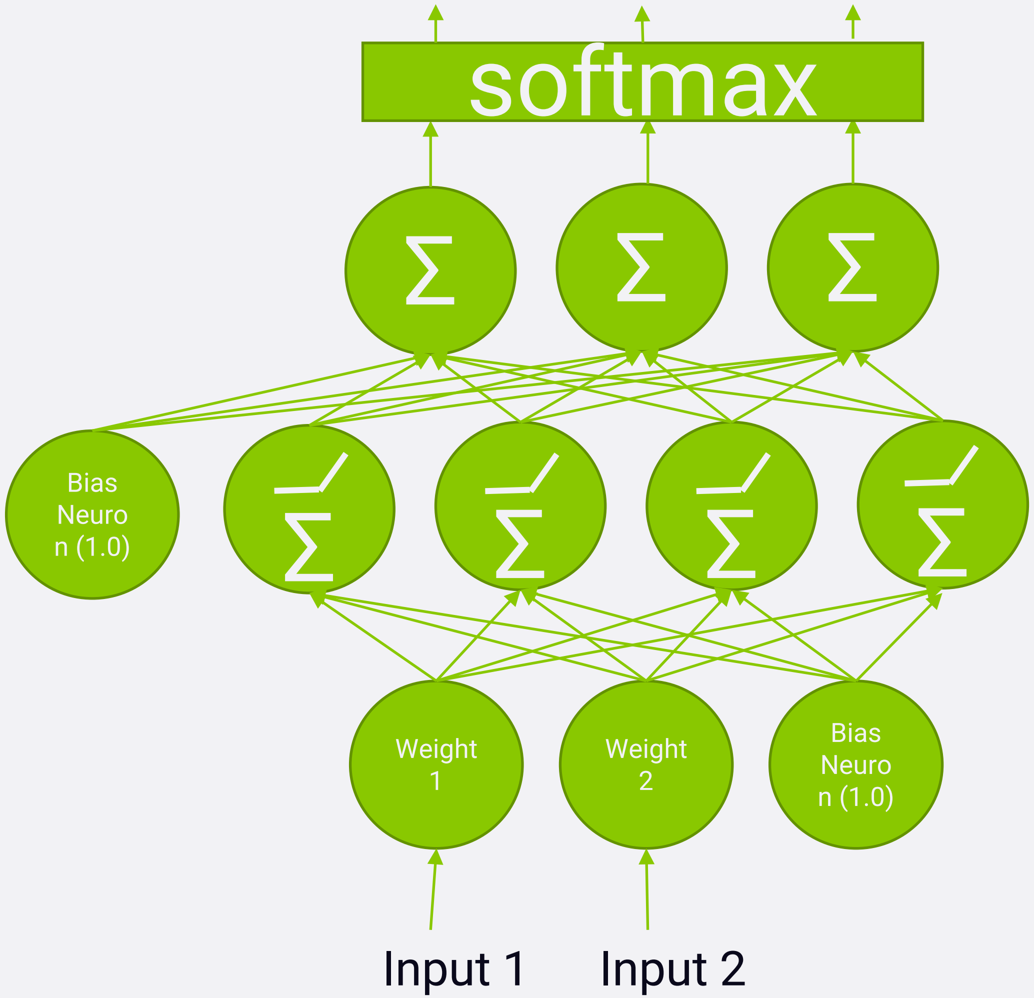
what is regularization?

- Preventing *overfitting*
  - Models that are good at making predictions on the data they were trained on, but not on new data it hasn't seen before
  - Overfitted models have learned patterns in the training data that don't generalize to the real world
  - Often seen as high accuracy on training data set, but lower accuracy on test or evaluation data set.
    - When training and evaluating a model, we use *training*, *evaluation*, and *testing* data sets.
- Regularization techniques are intended to prevent overfitting.

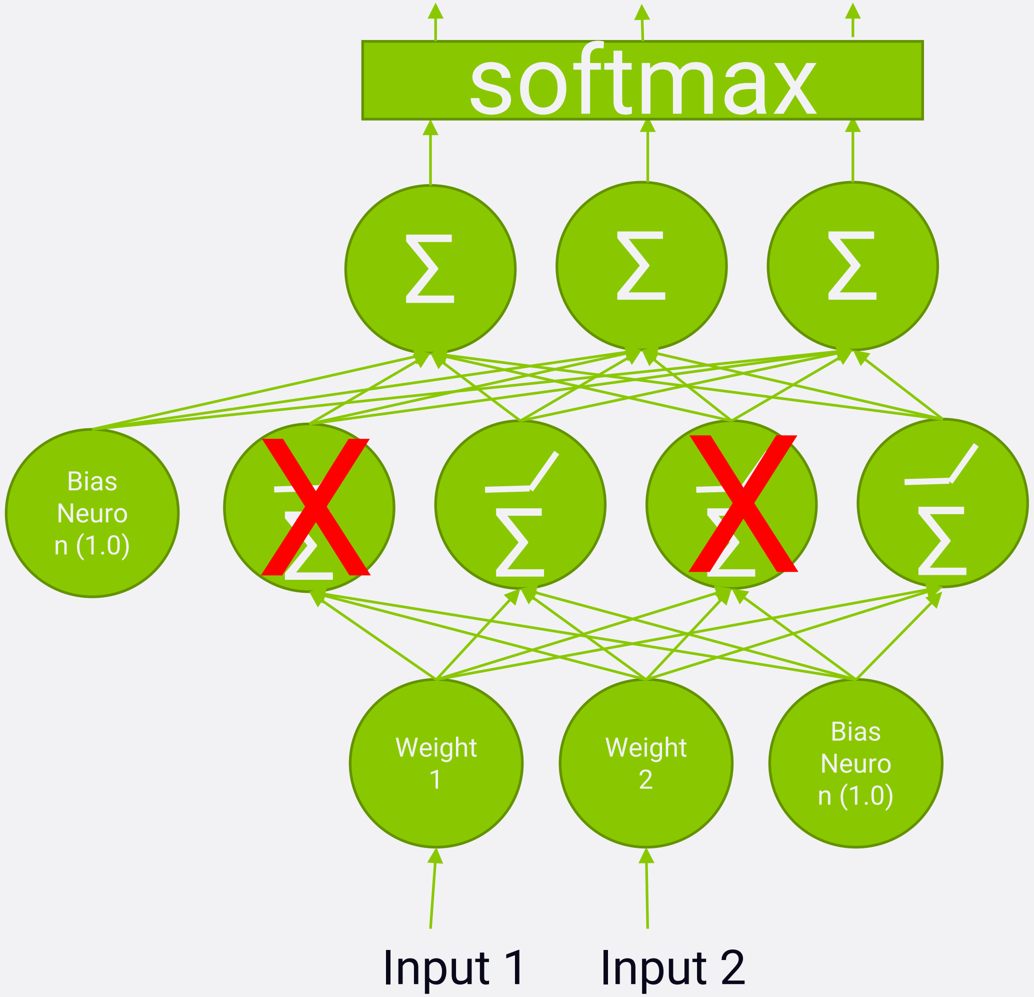


Chabacano [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)]

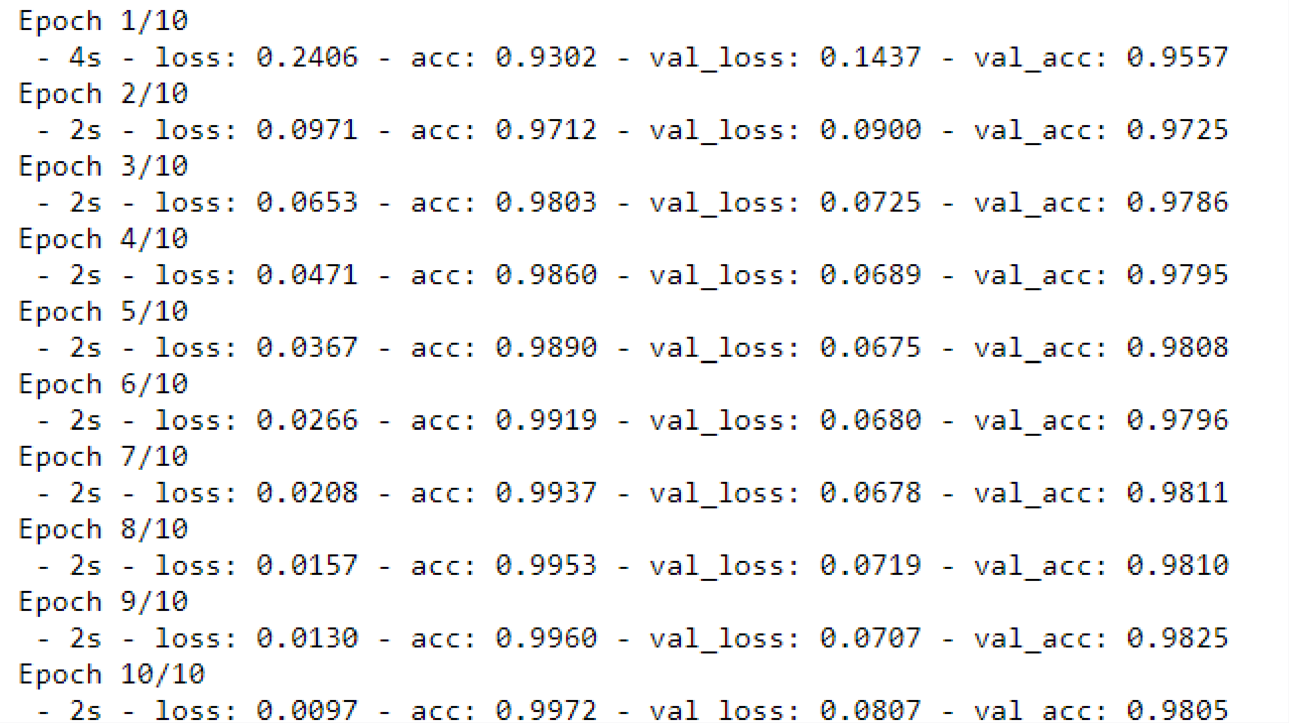
Too many layers?
Too many neurons?


softmax
Σ
Σ
Σ
Bias Neuron (1.0)
Σ
Σ
Σ
Σ
Weight 1
Weight 2
Bias Neuron (1.0)
Input 1
Input 2

Dropout


softmax
Σ
Σ
Σ
Bias Neuron (1.0)
X
Σ
Σ
X
Σ
Σ
Weight 1
Weight 2
Bias Neuron (1.0)
Input 1
Input 2

Early Stopping



```
Epoch 1/10
 - 4s - loss: 0.2406 - acc: 0.9302 - val_loss: 0.1437 - val_acc: 0.9557
Epoch 2/10
 - 2s - loss: 0.0971 - acc: 0.9712 - val_loss: 0.0900 - val_acc: 0.9725
Epoch 3/10
 - 2s - loss: 0.0653 - acc: 0.9803 - val_loss: 0.0725 - val_acc: 0.9786
Epoch 4/10
 - 2s - loss: 0.0471 - acc: 0.9860 - val_loss: 0.0689 - val_acc: 0.9795
Epoch 5/10
 - 2s - loss: 0.0367 - acc: 0.9890 - val_loss: 0.0675 - val_acc: 0.9808
Epoch 6/10
 - 2s - loss: 0.0266 - acc: 0.9919 - val_loss: 0.0680 - val_acc: 0.9796
Epoch 7/10
 - 2s - loss: 0.0208 - acc: 0.9937 - val_loss: 0.0678 - val_acc: 0.9811
Epoch 8/10
 - 2s - loss: 0.0157 - acc: 0.9953 - val_loss: 0.0719 - val_acc: 0.9810
Epoch 9/10
 - 2s - loss: 0.0130 - acc: 0.9960 - val_loss: 0.0707 - val_acc: 0.9825
Epoch 10/10
 - 2s - loss: 0.0097 - acc: 0.9972 - val loss: 0.0807 - val acc: 0.9805
```

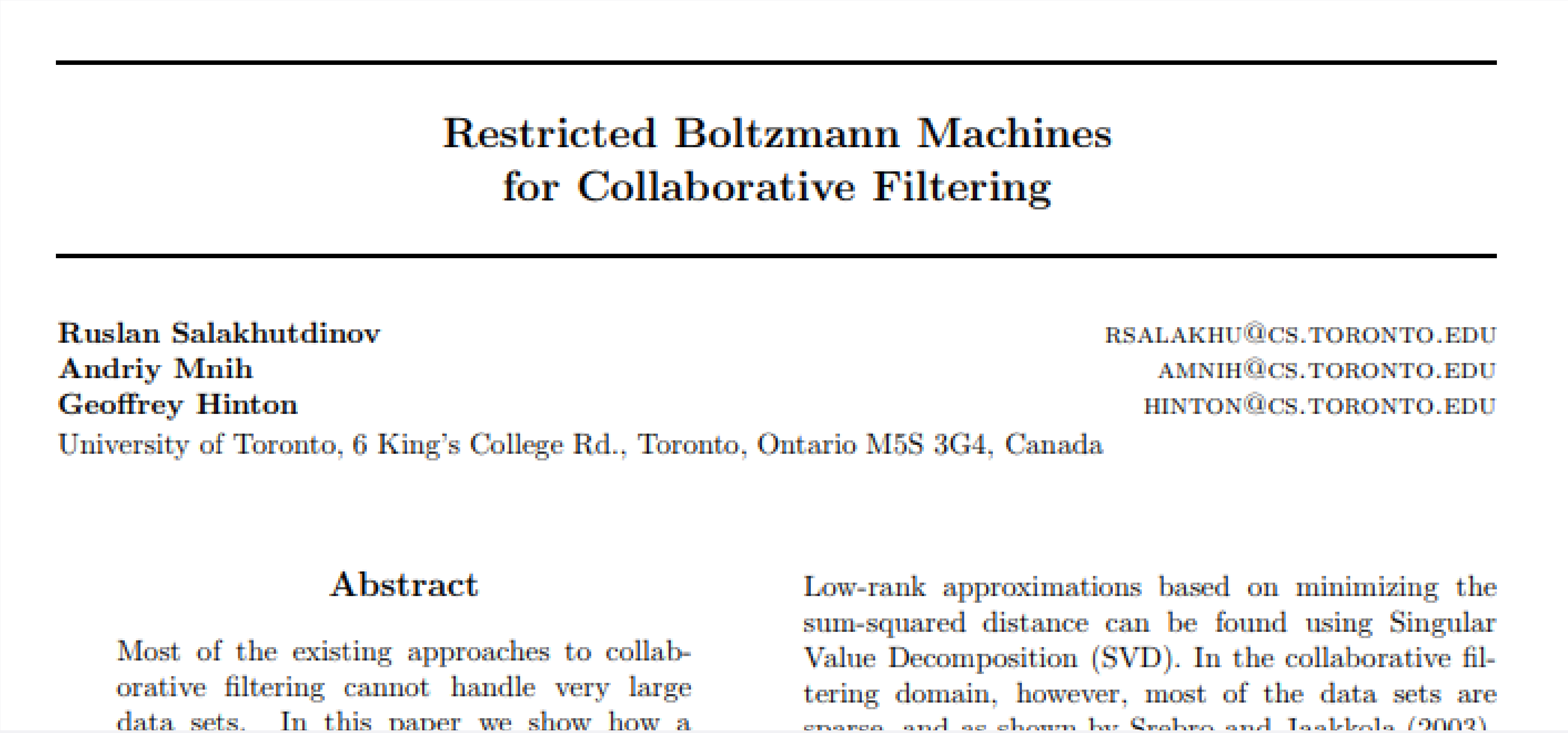
wrapping up

recommendations with deep learning

is deep learning overkill?

restricted boltzmann machines (rbm)

rbm's: the paper



Restricted Boltzmann Machines for Collaborative Filtering

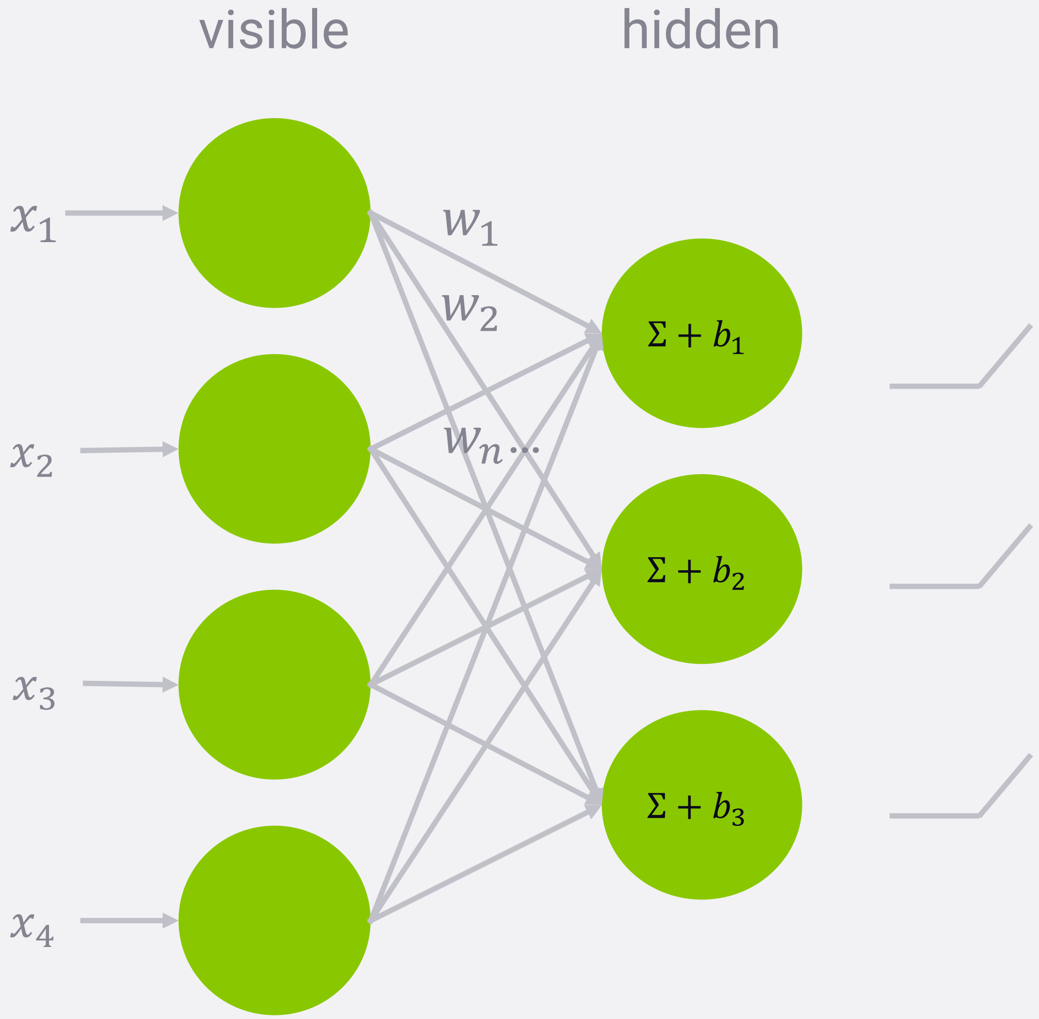
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Andriy Mnih AMNIH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
University of Toronto, 6 King's College Rd., Toronto, Ontario M5S 3G4, Canada

Abstract

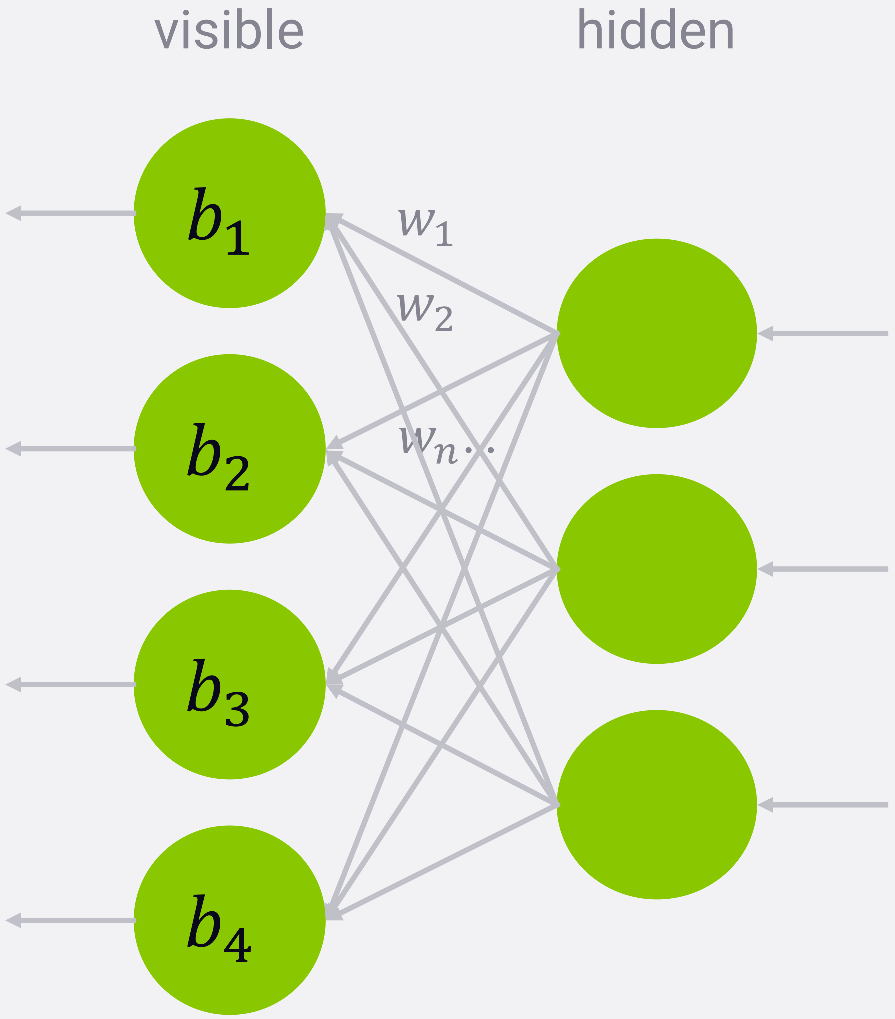
Most of the existing approaches to collaborative filtering cannot handle very large data sets. In this paper we show how a

Low-rank approximations based on minimizing the sum-squared distance can be found using Singular Value Decomposition (SVD). In the collaborative filtering domain, however, most of the data sets are sparse, and as shown by Srebro and Jaakkola (2003),

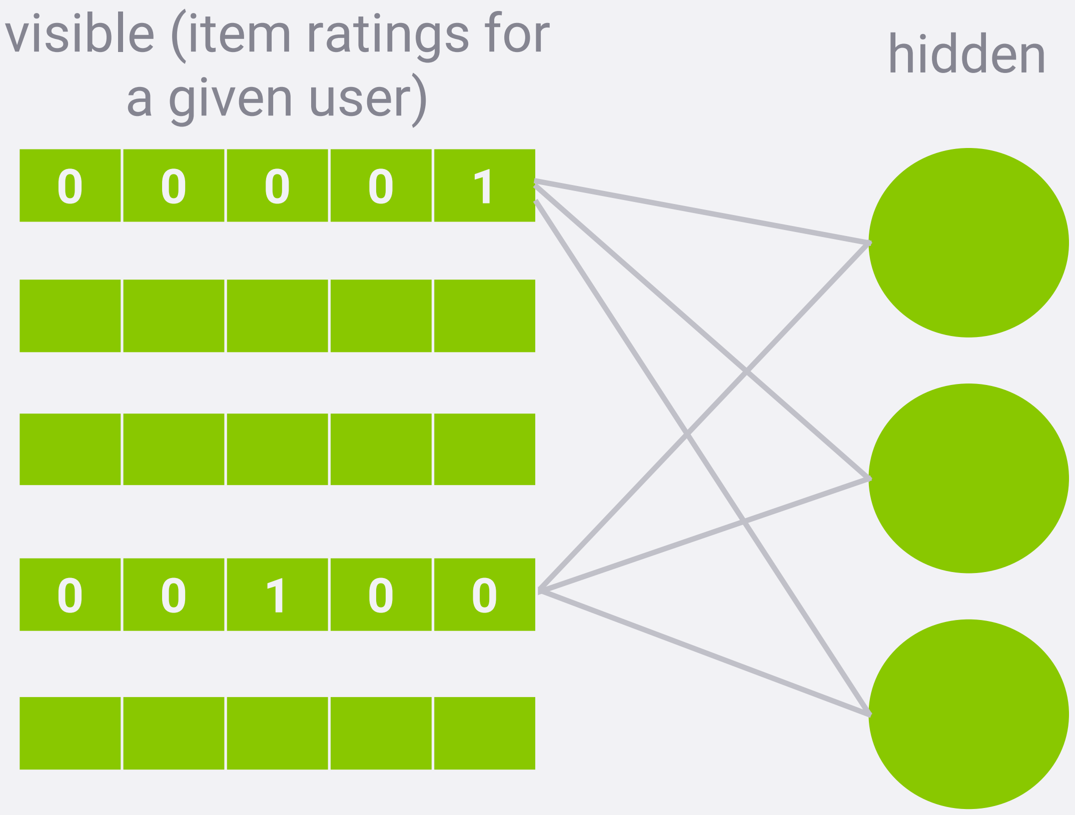
what is a rbm


visible
hidden
x_1
x_2
x_3
x_4
w_1
w_2
$w_n \ldots$
$\Sigma + b_1$
$\Sigma + b_2$
$\Sigma + b_3$

rbm backward pass


visible
hidden
b_1
b_2
b_3
b_4
w_1
w_2
w_n...

rbm's for recommender systems


visible (item ratings for a given user)
hidden
0 0 0 0 1
0 0 1 0 0

contrastive divergence

gibbs sampler

code walkthrough

code walkthrough

code walkthrough

exercise

Find the best set of hyperparameters for the rbm algorithm.

code walkthrough

Generative Adversarial Networks

Generative Adversarial Networks

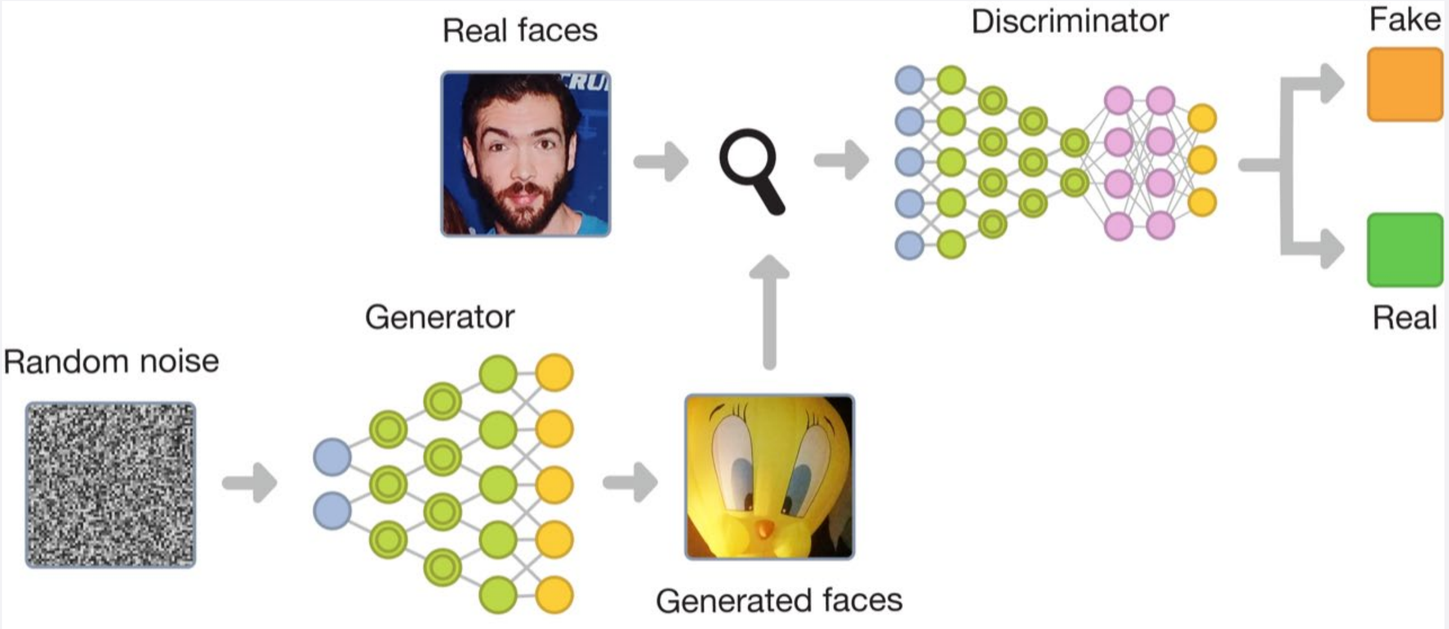
- Yes, it's the tech behind "deepfakes" and all those viral face-swapping and aging apps
- But researchers had nobler intentions…
  - Generating synthetic datasets to remove private info
  - Anomaly detection
  - Self-driving
  - Art, music

This person doesn't exist.

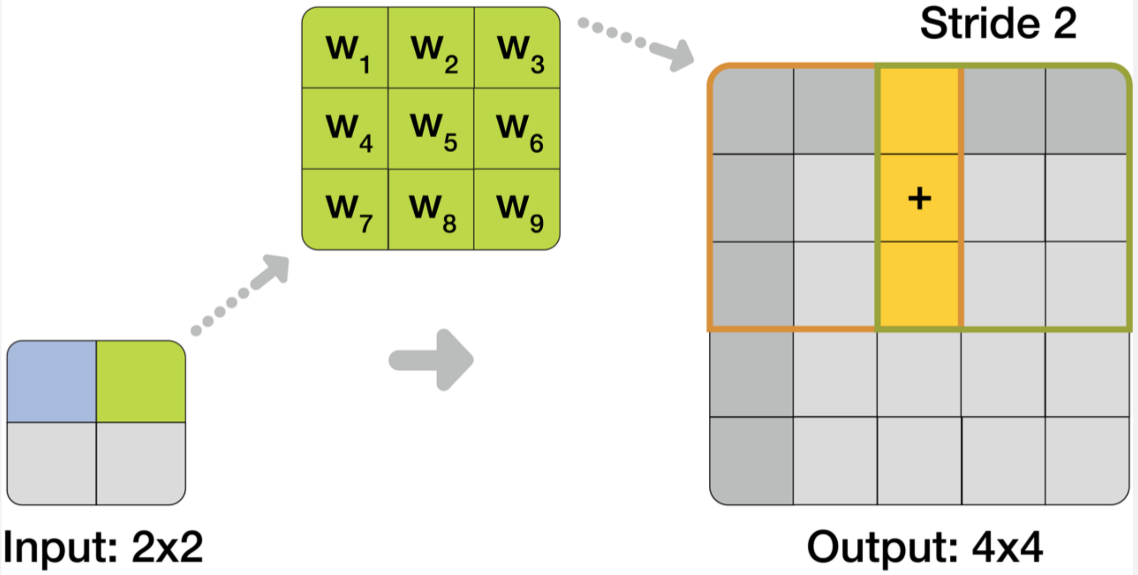
Datasciencearabic1, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

GAN’s

- Learns the actual distribution of latent vectors
    - Doesn’t assume Gaussian normal distributions like VAE’s
- The **generator** maps random noise(!) to a probability distribution
- The **discriminator** learns to identify real images from generated (fake) images
- The generator is trying to fool the discriminator into thinking its images are real
- The discriminator is trying to catch the generator
- The generator and discriminator are *adversarial*, hence the name…
- Once the discriminator can’t tell the difference anymore, we’re done (in theory)


Real faces
Discriminator
Fake
Real
Generator
Random noise
Generated faces

transpose convolution



w_1 w_2 w_3
w_4 w_5 w_6
w_7 w_8 w_9
Stride 2
+
Input: 2x2
Output: 4x4

- The generator may use Conv2DTranspose layers to reconstruct images from random input
- It learns weights used to create new image pixels from lower-dimensional representations
    - Well, it can be used on more than just images
- Stride of 2 is often used
- Can use max-unpooling (inverse of max-pooling)
- Think of the decoder as a CNN that works backwards.

fancy math

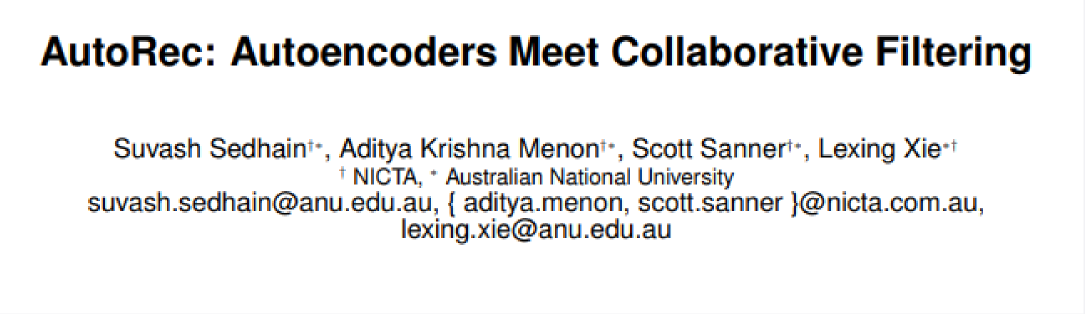
$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))].$$

- That’s the adversarial loss function.
- We call it a “min-max game”
  - The generator is minimizing its loss in creating realistic images
  - The discriminator, at the same time, is maximizing its ability to detect fakes
- It is complicated and delicate.
  - Training is very unstable; lots of trial & error / hyperparameter tuning
  - Mode collapse
  - Vanishing gradients

code walkthrough

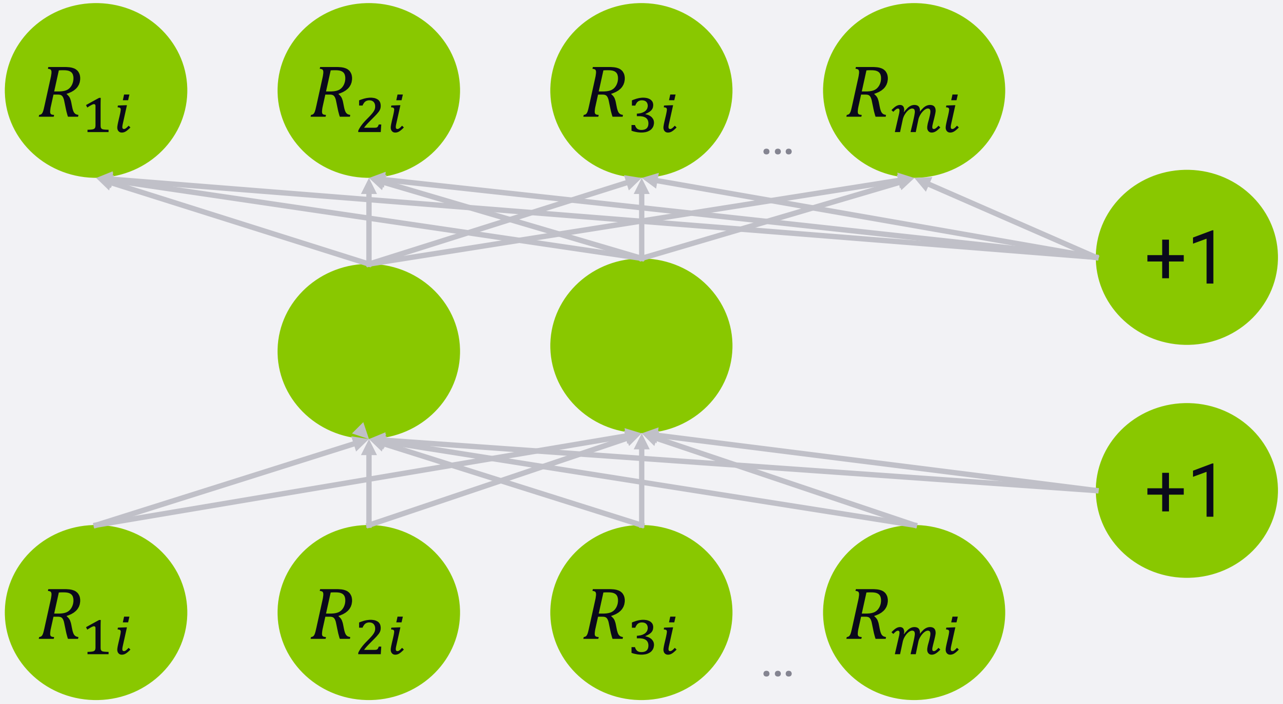
deep neural networks for recommendations

autoencoders for recommendations ("autorec")

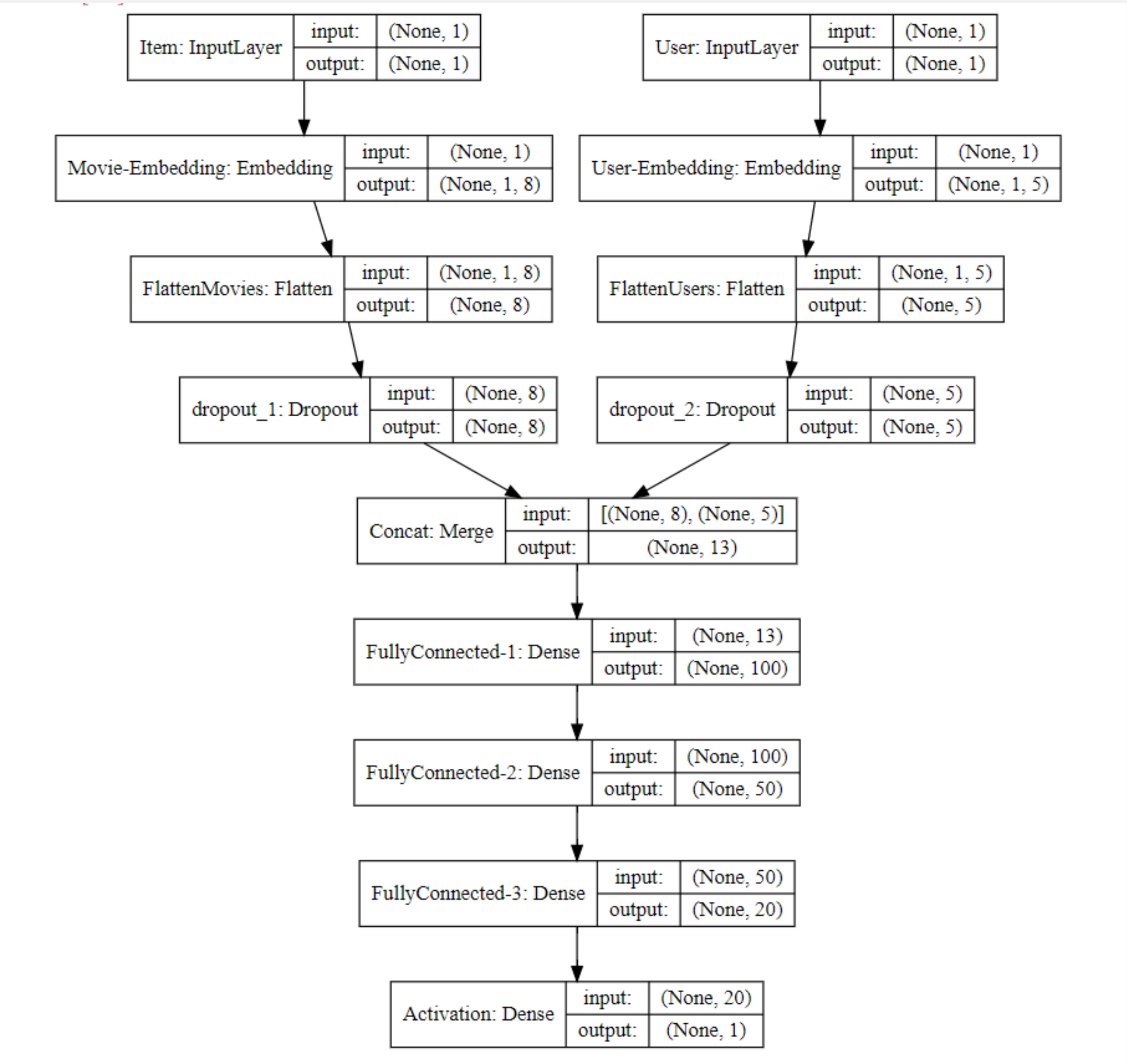


AutoRec: Autoencoders Meet Collaborative Filtering

Suvash Sedhain[†*], Aditya Krishna Menon[†*], Scott Sanner[†*], Lexing Xie[*†]
[†] NICTA, [*] Australian National University
suvash.sedhain@anu.edu.au, { aditya.menon, scott.sanner }@nicta.com.au,
lexing.xie@anu.edu.au



deeper networks with keras

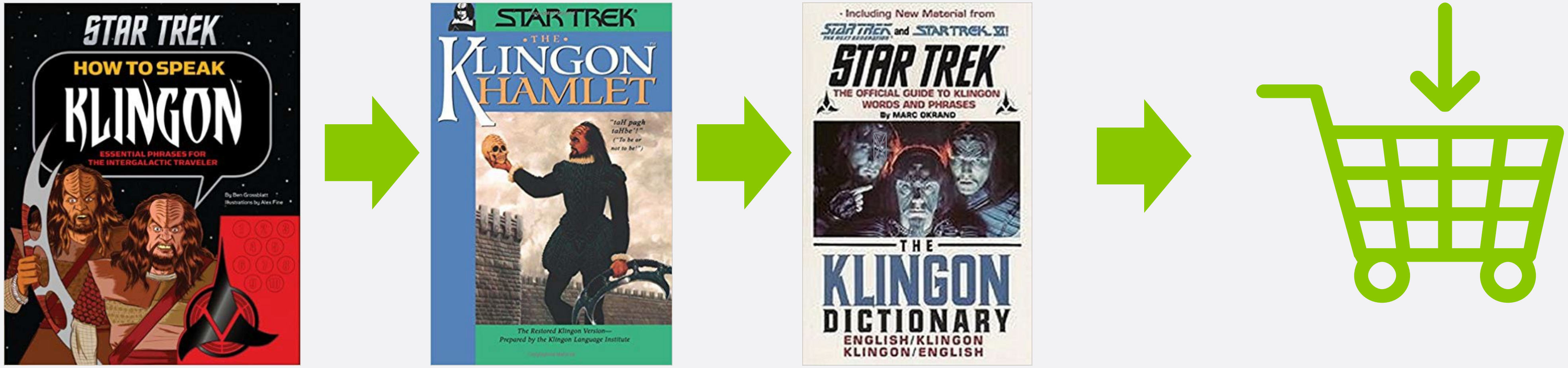

Item: InputLayer | input: (None, 1) | output: (None, 1)
User: InputLayer | input: (None, 1) | output: (None, 1)
Movie-Embedding: Embedding | input: (None, 1) | output: (None, 1, 8)
User-Embedding: Embedding | input: (None, 1) | output: (None, 1, 5)
FlattenMovies: Flatten | input: (None, 1, 8) | output: (None, 8)
FlattenUsers: Flatten | input: (None, 1, 5) | output: (None, 5)
dropout_1: Dropout | input: (None, 8) | output: (None, 8)
dropout_2: Dropout | input: (None, 5) | output: (None, 5)
Concat: Merge | input: [(None, 8), (None, 5)] | output: (None, 13)
FullyConnected-1: Dense | input: (None, 13) | output: (None, 100)
FullyConnected-2: Dense | input: (None, 100) | output: (None, 50)
FullyConnected-3: Dense | input: (None, 50) | output: (None, 20)
Activation: Dense | input: (None, 20) | output: (None, 1)

Credit:
https://nipunbatra.github.io/blog/2017/recommend-keras.html

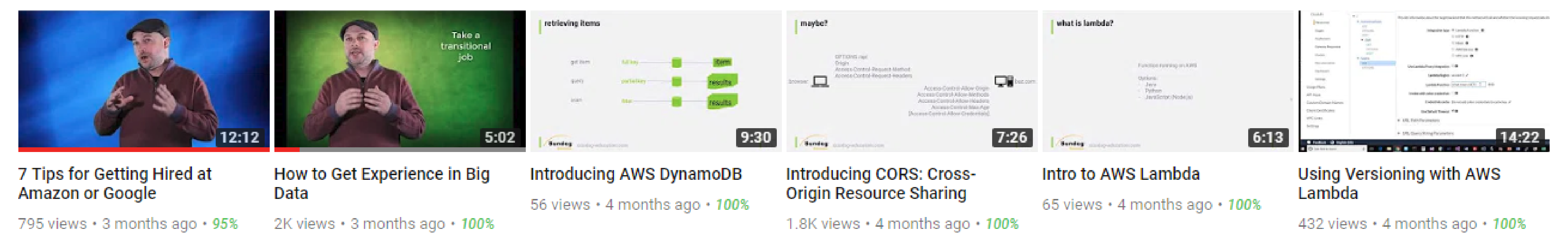
code walkthrough

session-based recommendations with rnn's

e-commerce clickstream


STAR TREK
HOW TO SPEAK
KLINGON
ESSENTIAL PHRASES FOR THE INTERGALACTIC TRAVELER
STAR TREK
THE KLINGON HAMLET
Including New Material from STAR TREK and STAR TREK VI
STAR TREK
THE OFFICIAL GUIDE TO KLINGON WORDS AND PHRASES
By MARC OKRAND
THE KLINGON DICTIONARY
ENGLISH/KLINGON
KLINGON/ENGLISH

video views


12:12
Take a transitional job
5:02
retrieving items
9:30
maybe?
7:26
what is lambda?
6:13
14:22

7 Tips for Getting Hired at Amazon or Google
795 views • 3 months ago • 95%

How to Get Experience in Big Data
2K views • 3 months ago • 100%

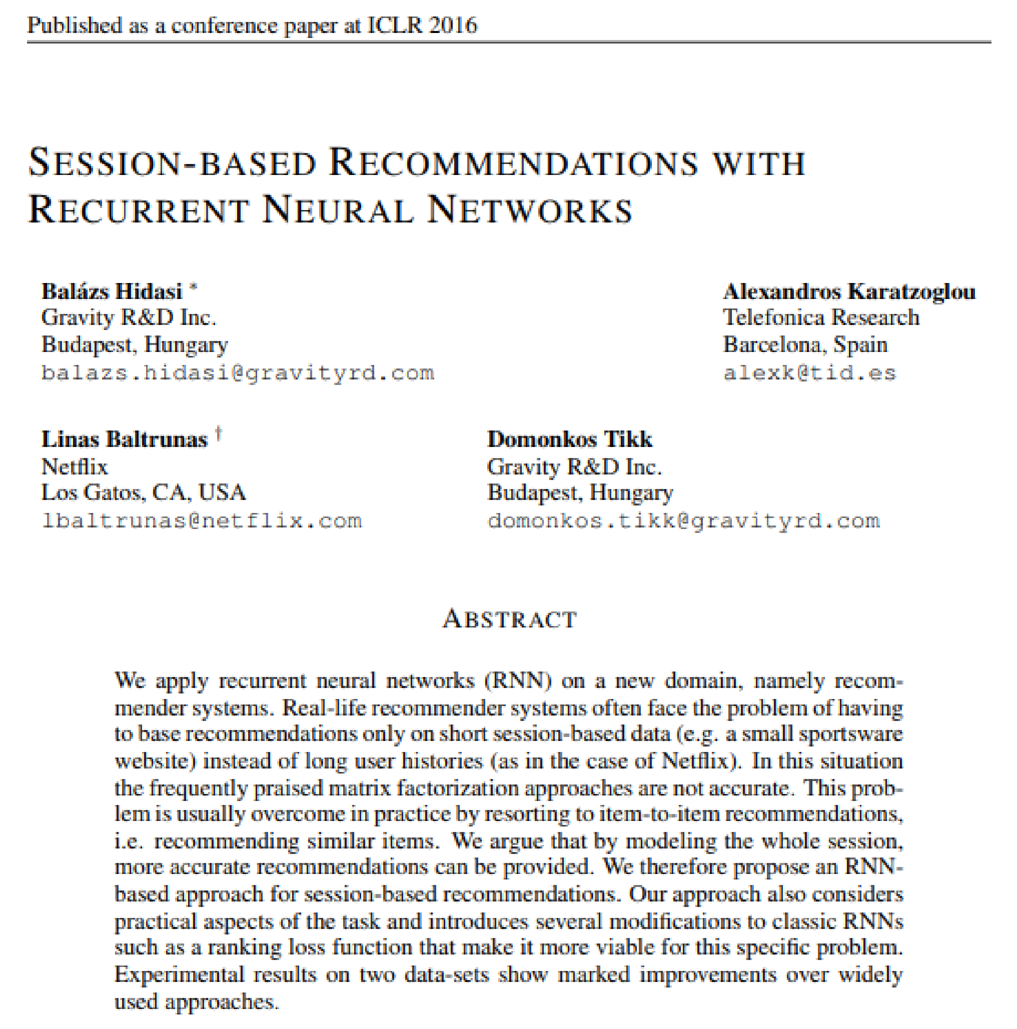
Introducing AWS DynamoDB
56 views • 4 months ago • 100%

Introducing CORS: Cross-Origin Resource Sharing
1.8K views • 4 months ago • 100%

Intro to AWS Lambda
65 views • 4 months ago • 100%

Using Versioning with AWS Lambda
432 views • 4 months ago • 100%

the paper



Published as a conference paper at ICLR 2016

SESSION-BASED RECOMMENDATIONS WITH RECURRENT NEURAL NETWORKS

Balázs Hidasi *
Gravity R&D Inc.
Budapest, Hungary
balazs.hidasi@gravityrd.com

Alexandros Karatzoglou
Telefonica Research
Barcelona, Spain
alexk@tid.es

Linas Baltrunas †
Netflix
Los Gatos, CA, USA
lbaltrunas@netflix.com

Domonkos Tikk
Gravity R&D Inc.
Budapest, Hungary
domonkos.tikk@gravityrd.com

ABSTRACT

We apply recurrent neural networks (RNN) on a new domain, namely recommender systems. Real-life recommender systems often face the problem of having to base recommendations only on short session-based data (e.g. a small sportsware website) instead of long user histories (as in the case of Netflix). In this situation the frequently praised matrix factorization approaches are not accurate. This problem is usually overcome in practice by resorting to item-to-item recommendations, i.e. recommending similar items. We argue that by modeling the whole session, more accurate recommendations can be provided. We therefore propose an RNN-based approach for session-based recommendations. Our approach also considers practical aspects of the task and introduces several modifications to classic RNNs such as a ranking loss function that make it more viable for this specific problem. Experimental results on two data-sets show marked improvements over widely used approaches.

GRU4Rec (gated recurrent unit)

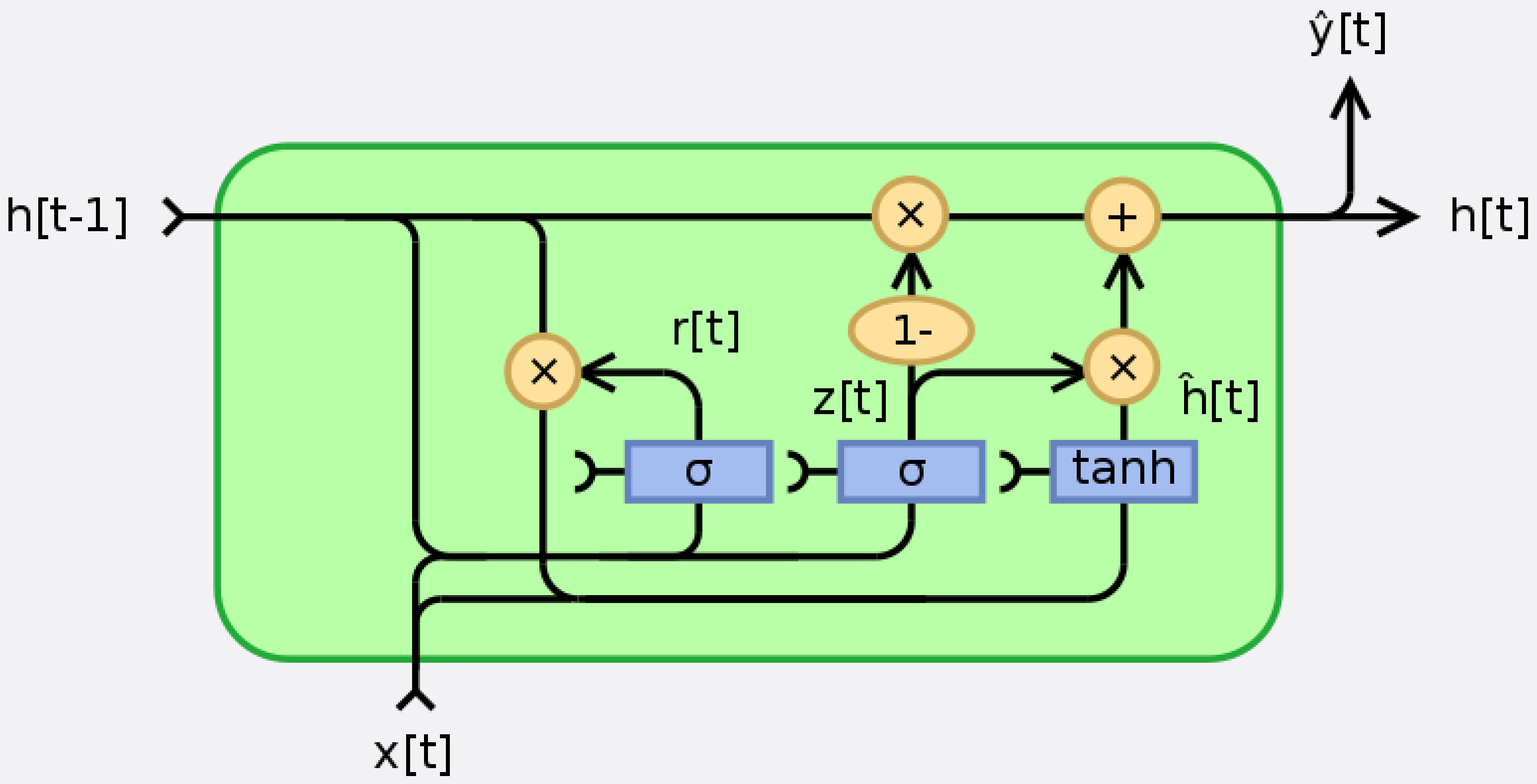
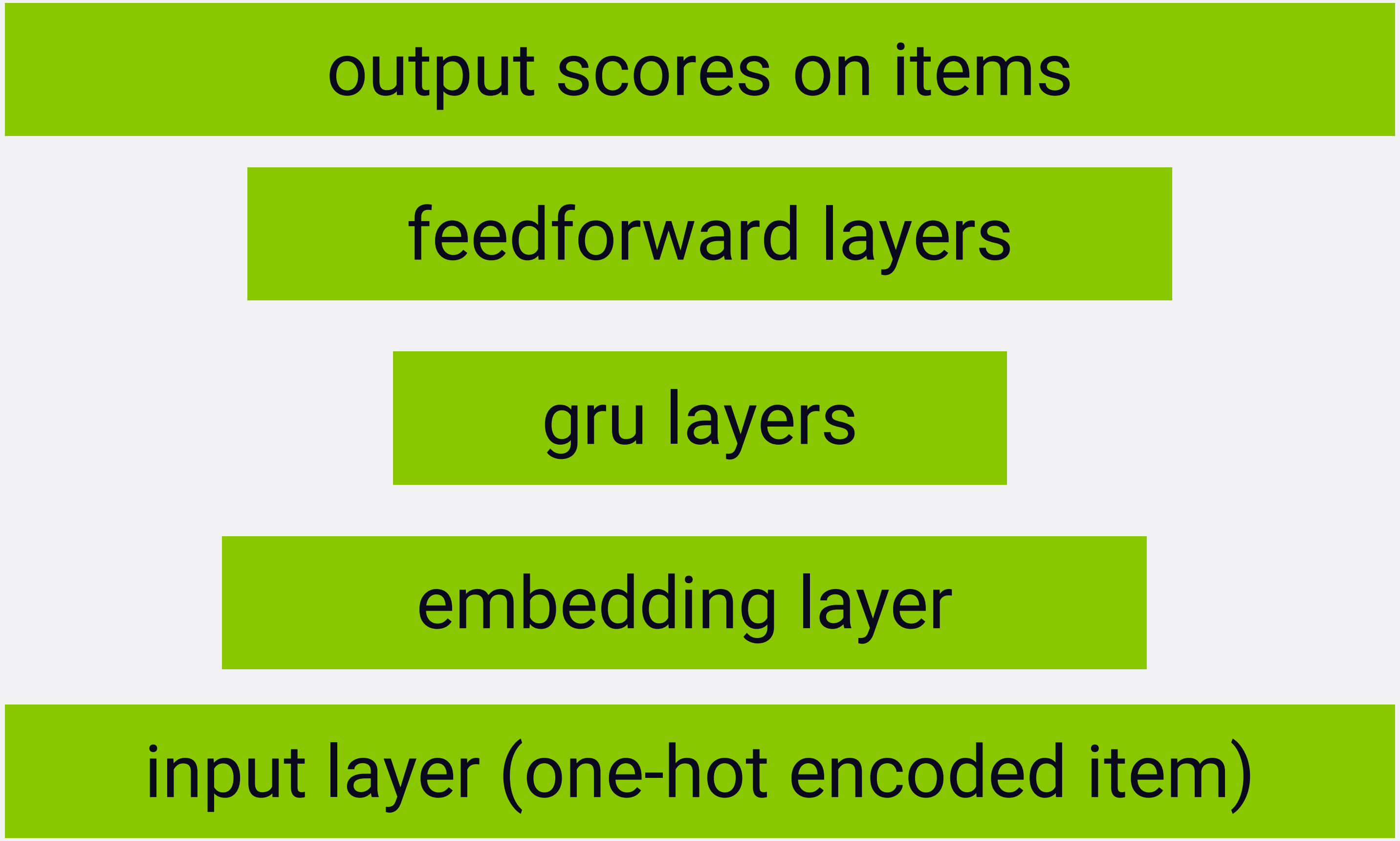

ŷ[t]
h[t-1]
h[t]
r[t]
1-
z[t]
ĥ[t]
σ
σ
tanh
x[t]

Image: Jeblad / CC BY-SA 4.0

GRU4Rec



output scores on items

feedforward layers

gru layers

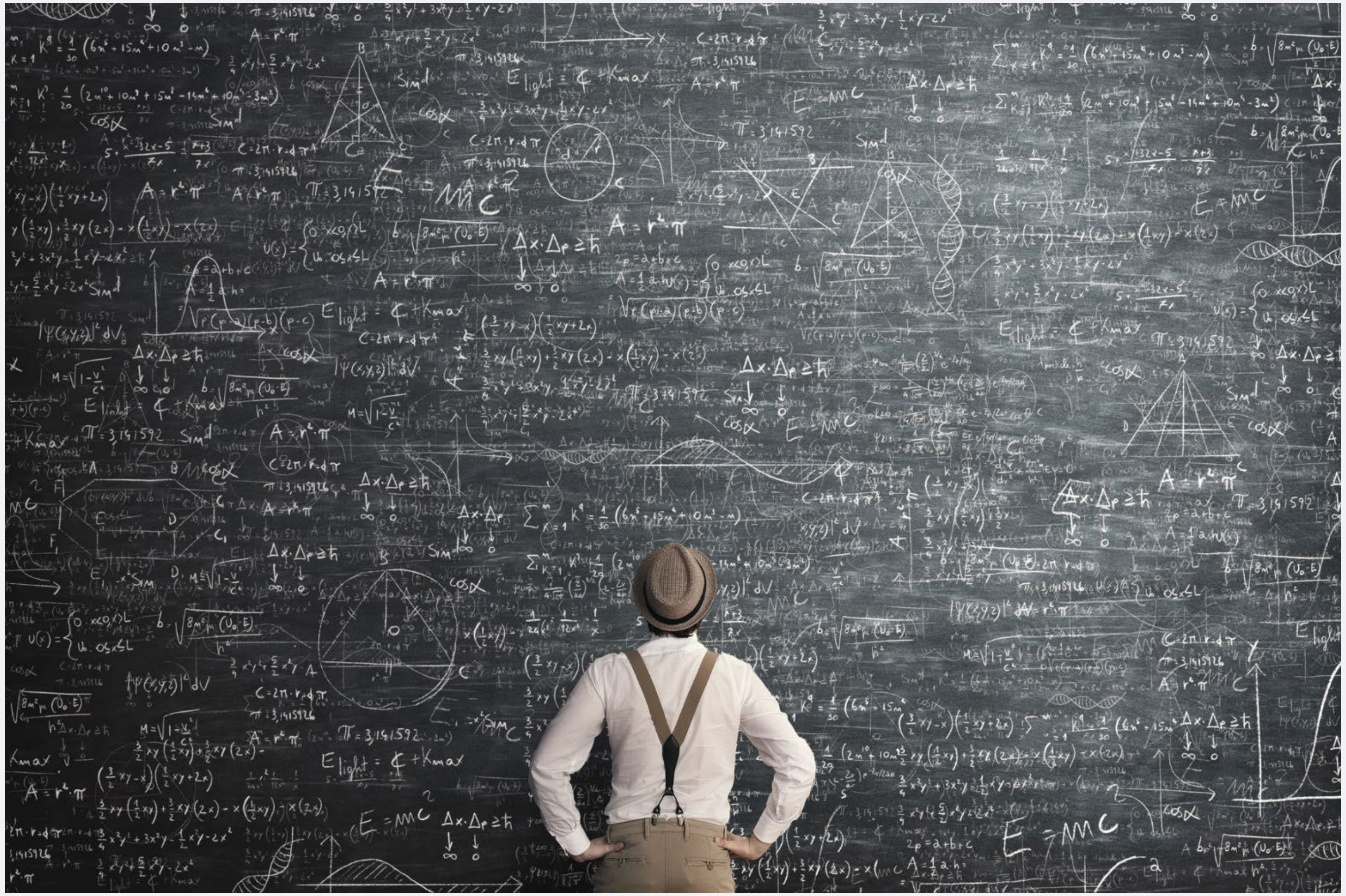
embedding layer

input layer (one-hot encoded item)

GRU4Rec

- session-parallel mini-batches
- sampling the output
- ranking loss

is it overly complex?



exercise

https://bit.ly/2zsr6Lh

convert to python 3 (xrange/range, sort/sort_values)

import pandas and scikit-learn

adapt to the new data set format

create a train/test split

always run with a fresh kernel

my solution

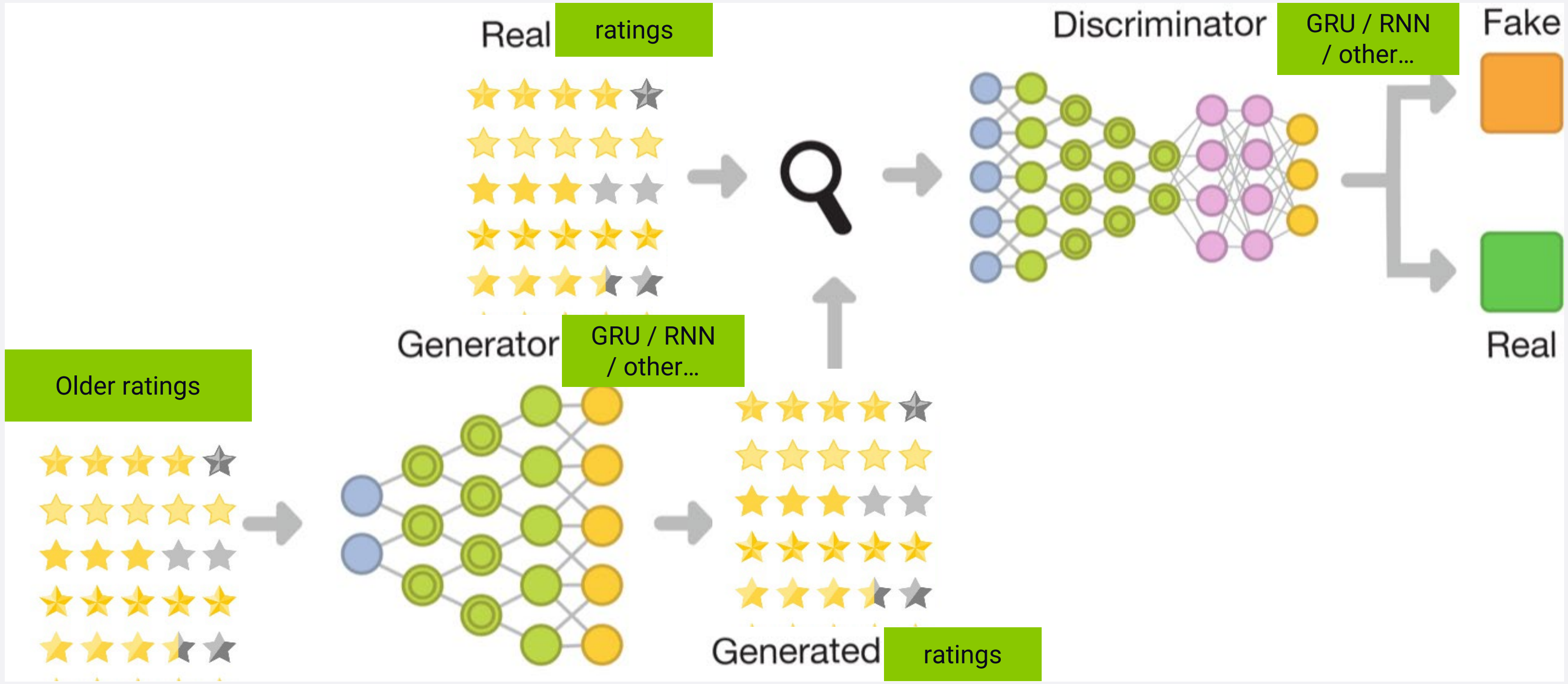
http://tinyurl.com/y9ducpag

code walkthrough

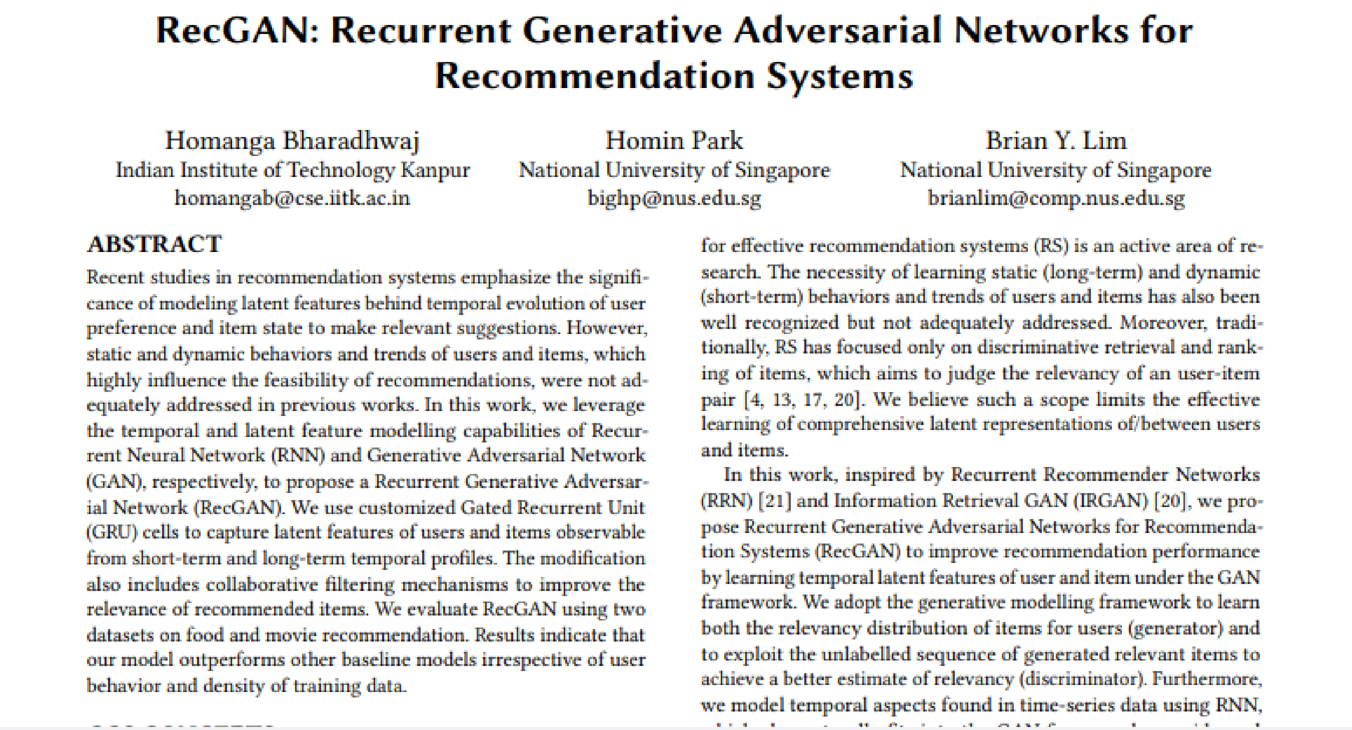
bleeding edge alert!

GAN’s for recommenders

GAN's with a twist


Real ratings
Discriminator GRU / RNN / other…
Fake
Real
Generator GRU / RNN / other…
Older ratings
Generated ratings

the paper (well, one of many really)



RecGAN: Recurrent Generative Adversarial Networks for Recommendation Systems

Homanga Bharadhwaj
Indian Institute of Technology Kanpur
homangab@cse.iitk.ac.in

Homin Park
National University of Singapore
bighp@nus.edu.sg

Brian Y. Lim
National University of Singapore
brianlim@comp.nus.edu.sg

ABSTRACT

Recent studies in recommendation systems emphasize the significance of modeling latent features behind temporal evolution of user preference and item state to make relevant suggestions. However, static and dynamic behaviors and trends of users and items, which highly influence the feasibility of recommendations, were not adequately addressed in previous works. In this work, we leverage the temporal and latent feature modelling capabilities of Recurrent Neural Network (RNN) and Generative Adversarial Network (GAN), respectively, to propose a Recurrent Generative Adversarial Network (RecGAN). We use customized Gated Recurrent Unit (GRU) cells to capture latent features of users and items observable from short-term and long-term temporal profiles. The modification also includes collaborative filtering mechanisms to improve the relevance of recommended items. We evaluate RecGAN using two datasets on food and movie recommendation. Results indicate that our model outperforms other baseline models irrespective of user behavior and density of training data.

for effective recommendation systems (RS) is an active area of research. The necessity of learning static (long-term) and dynamic (short-term) behaviors and trends of users and items has also been well recognized but not adequately addressed. Moreover, traditionally, RS has focused only on discriminative retrieval and ranking of items, which aims to judge the relevancy of an user-item pair [4, 13, 17, 20]. We believe such a scope limits the effective learning of comprehensive latent representations of/between users and items.

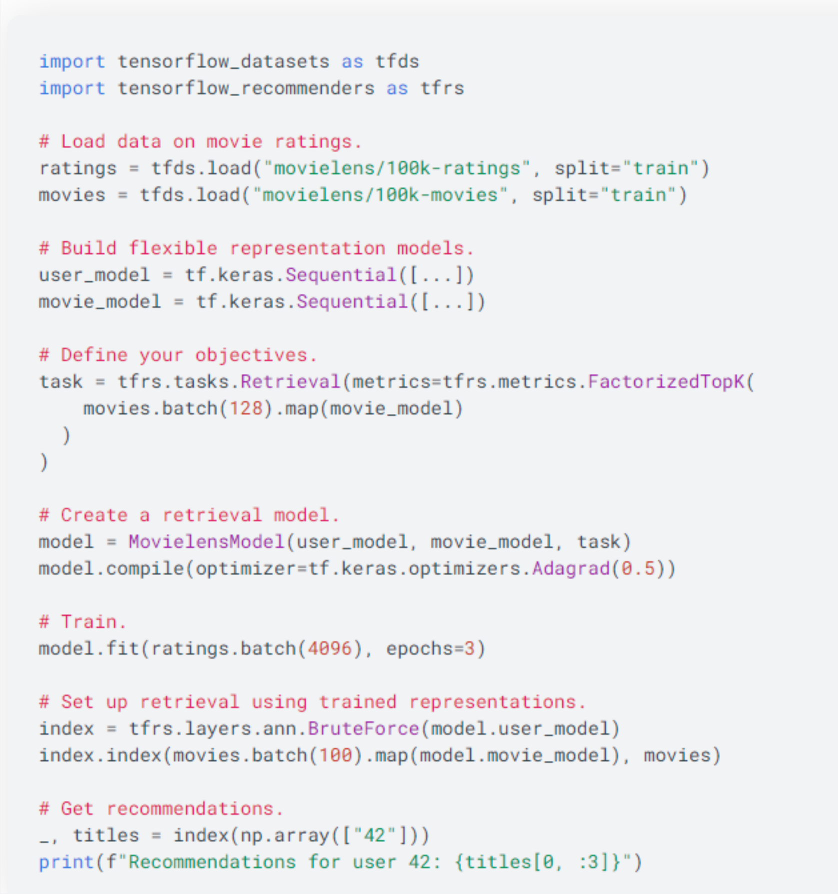
In this work, inspired by Recurrent Recommender Networks (RRN) [21] and Information Retrieval GAN (IRGAN) [20], we propose Recurrent Generative Adversarial Networks for Recommendation Systems (RecGAN) to improve recommendation performance by learning temporal latent features of user and item under the GAN framework. We adopt the generative modelling framework to learn both the relevancy distribution of items for users (generator) and to exploit the unlabelled sequence of generated relevant items to achieve a better estimate of relevancy (discriminator). Furthermore, we model temporal aspects found in time-series data using RNN,

https://homangab.github.io/papers/recgan.pdf

TensorFlow Recommenders (TFRS)

TensorFlow Recommenders

- From Google!
- Built on top of Keras
- Easy to use, but highly flexible



```
import tensorflow_datasets as tfds
import tensorflow_recommenders as tfrs

# Load data on movie ratings.
ratings = tfds.load("movielens/100k-ratings", split="train")
movies = tfds.load("movielens/100k-movies", split="train")

# Build flexible representation models.
user_model = tf.keras.Sequential([...])
movie_model = tf.keras.Sequential([...])

# Define your objectives.
task = tfrs.tasks.Retrieval(metrics=tfrs.metrics.FactorizedTopK(
    movies.batch(128).map(movie_model)
  )
)

# Create a retrieval model.
model = MovielensModel(user_model, movie_model, task)
model.compile(optimizer=tf.keras.optimizers.Adagrad(0.5))

# Train.
model.fit(ratings.batch(4096), epochs=3)

# Set up retrieval using trained representations.
index = tfrs.layers.ann.BruteForce(model.user_model)
index.index(movies.batch(100).map(model.movie_model), movies)

# Get recommendations.
_, titles = index(np.array(["42"]))
print(f"Recommendations for user 42: {titles[0, :3]}")
```

TFRS: retrieval

A **retrieval stage** selects recommendation candidates

A **ranking stage** selects the best candidates and ranks them

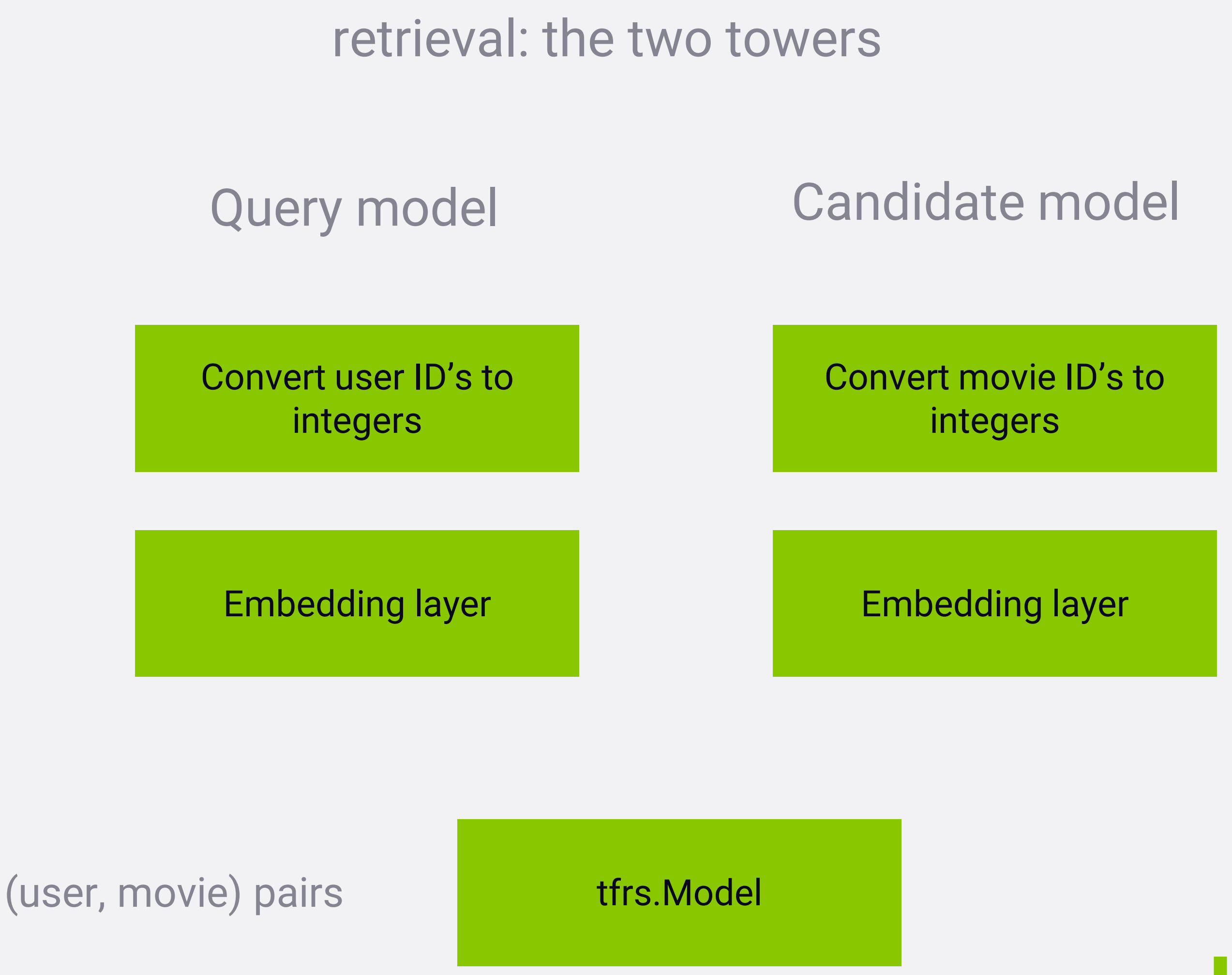
The retrieval model embeds user ID’s and movie ID’s of rated movies into **embedding layers** of the same dimension

- Each ID is mapped to a vector of N dimensions
- Position in this N-dimensional space represents similarity!

The two are multiplied to create query-candidate affinity scores for each rating during training

If the affinity score for the rating is higher than other for other candidates, our model is good

Top-K recs via “brute force” sorting all candidates



retrieval: the two towers

Query model

Convert user ID’s to integers

Embedding layer

Candidate model

Convert movie ID’s to integers

Embedding layer

(user, movie) pairs

tfrs.Model

code walkthrough

TFRS: ranking

As ranking uses a subset of candidates generated by retrieval, you can do fancier stuff.

For example, actually try to predict ratings using multiple stacked dense layers.

User AND movie embeddings

256 Dense layer

64 Dense layer

1 layer (final prediction)

code walkthrough

TFRS: side features

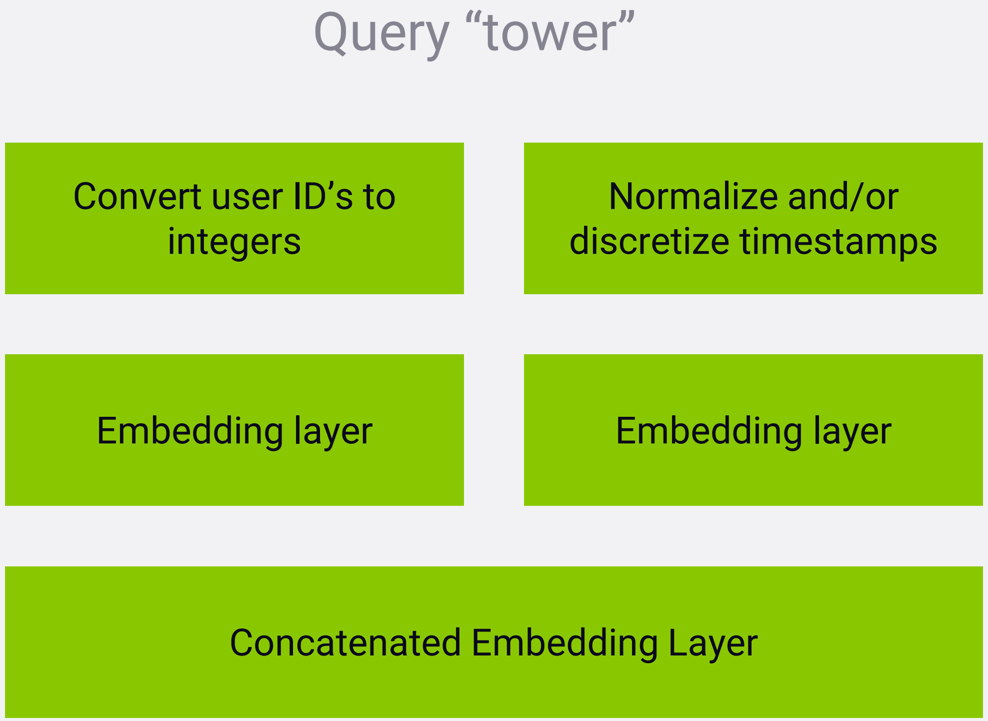
You can augment ratings data with content-based data, or any other features really

- Data should add **context**
- Helps cold-start

Just add them into the query or candidate towers as additional embeddings

Preprocessing is up to you

- Categorical data should turn into embeddings
- Continuous features should be normalized (ie timestamps)
- Standardization
- Discretization
- Vectorizing text



Query "tower"

Convert user ID's to integers

Normalize and/or discretize timestamps

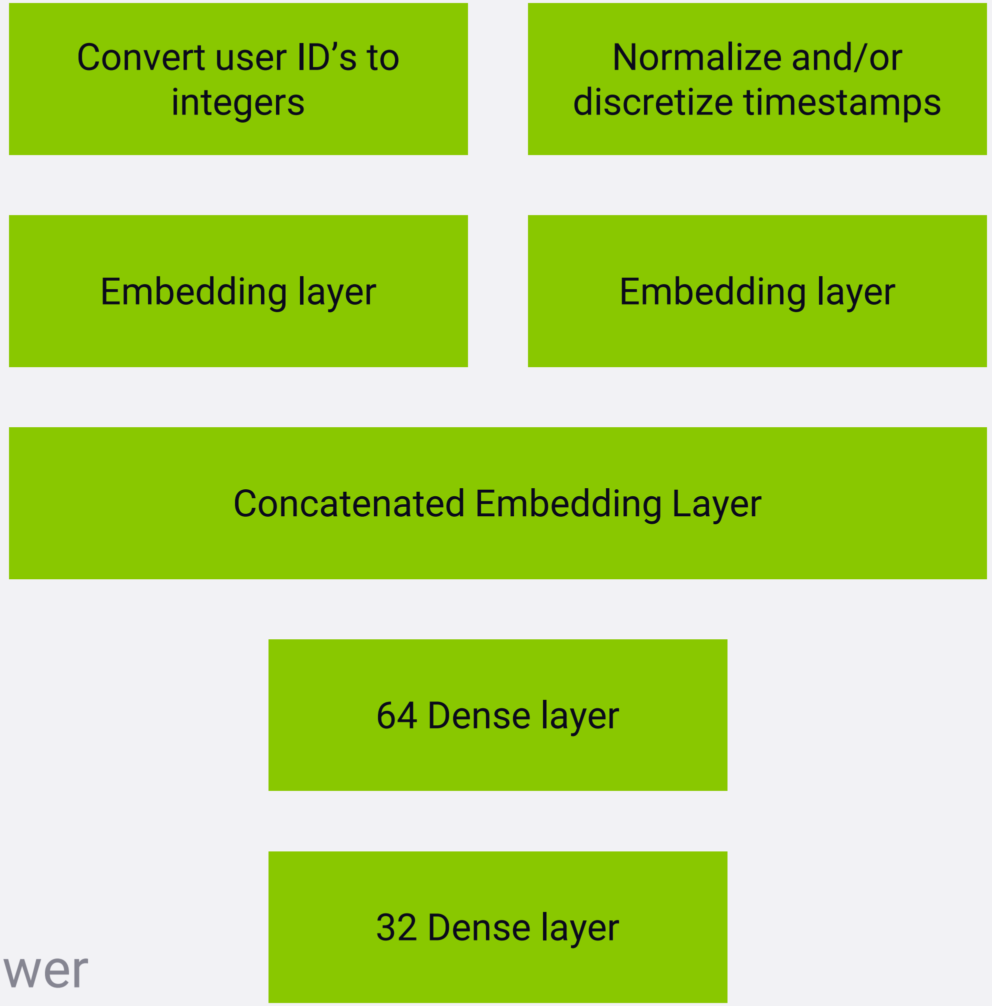
Embedding layer

Embedding layer

Concatenated Embedding Layer

TFRS: deep retrieval models

Query "tower"



Convert user ID's to integers

Normalize and/or discretize timestamps

Embedding layer

Embedding layer

Concatenated Embedding Layer

64 Dense layer

32 Dense layer

…can do similar stuff on the candidate tower

TFRS: multi-task recommenders

Combine different kinds of user behavior

- Page views
- Image Clicks
- Cart adds
- Purchases
- Reviews
- Returns
- Ratings

A joint model may perform better than multiple task-specific models

Multiple objectives & loss functions

Use transfer learning to learn representations from a task with more data for a task with less

TFRS: deep & cross networks

Feature crosses are hard

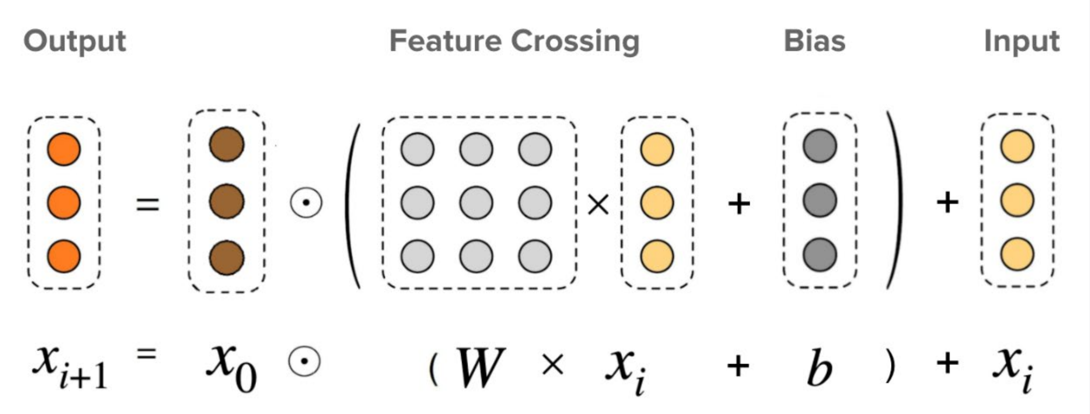
Recommendations where combined features provide additional context

If you bought fruit AND cookbooks, recommend a blender

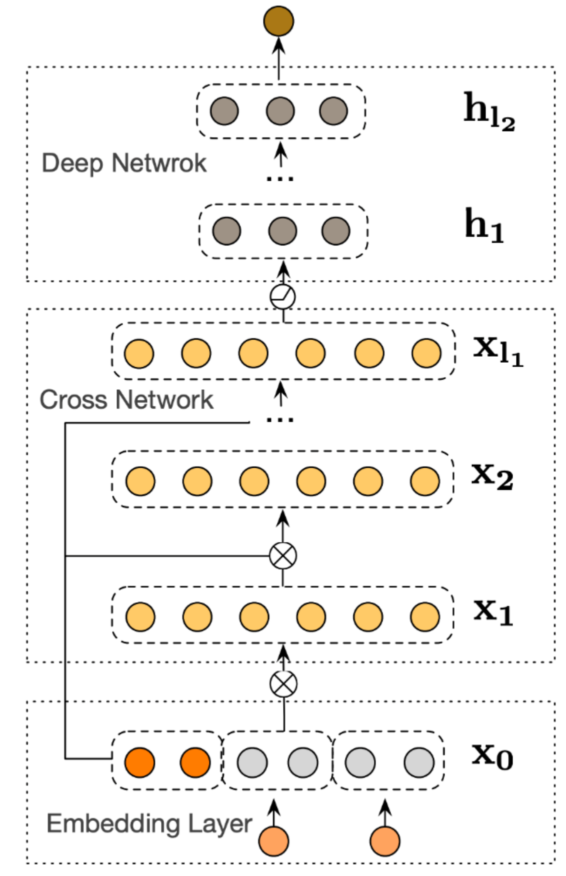
OFF ON MIN
MAX
PULSE
ICE CRUSH
SMOOTHIE

TFRS: deep & cross networks

Cross Networks explicitly apply feature crossing at each layer


Output
Feature Crossing
Bias
Input
=
⊙
×
+
+

$$x_{i+1} = x_0 \odot (W \times x_i + b) + x_i$$


$\mathbf{h_{l_2}}$
Deep Netwrok
$\mathbf{h_1}$
$\mathbf{x_{l_1}}$
Cross Network
$\mathbf{x_2}$
$\mathbf{x_1}$
$\mathbf{x_0}$
Embedding Layer

- Combine with a Deep Network (MLP) to make a DCN
- Stacked, or in parallel
- tfrs.layers.dcn.Cross() makes it easy

Images: tensorflow.org

TFRS: into production

Retrieval models are slow when evaluated with brute force

- Approximate Nearest Neighbor search (ANN)
- ScaNN package from Google does this
- tfrs.layers.factorized_top_k.ScaNN
- It is approximate! But way faster

Serving the results in production

- Export saved models to SavedModel format
- Serve the SavedModel via Tensorflow Serving
- See end of retrieval sample for an example

Accelerating Large-Scale Inference with Anisotropic Vector Quantization

Ruiqi Guo*, Philip Sun*, Erik Lindgren*, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar
Google Research
`{guorq, sunphil, erikml, qgeng, dsimcha, fchern, sanjivk}@google.com`

arXiv:1908.10396v5 [cs.LG] 4 Dec 2020

Abstract

Quantization based techniques are the current state-of-the-art for scaling maximum inner product search to massive databases. Traditional approaches to quantization aim to minimize the reconstruction error of the database points. Based on the observation that for a given query, the database points that have the largest inner products are more relevant, we develop a family of anisotropic quantization loss functions. Under natural statistical assumptions, we show that quantization with these loss functions leads to a new variant of vector quantization that more greatly penalizes the parallel component of a datapoint's residual relative to its orthogonal component. The proposed approach, whose implementation is open-source, achieves state-of-the-art results on the public benchmarks available at `ann-benchmarks.com`.

1 Introduction

Maximum inner product search (MIPS) has become a popular paradigm for solving large scale classification and retrieval tasks. For example, in recommendation systems, user queries and documents are embedded into a dense vector space of the same dimensionality and MIPS is used to find the most relevant documents given a user query (Cremonesi et al., 2010). Similarly, in extreme classification tasks (Dean et al., 2013), MIPS is used to predict the class label when a large number of classes, often on the order of millions or even billions are involved. Lately, MIPS has

Search (MIPS) problem, consider a database $X = \{x_i\}_{i=1,2,\ldots,n}$ with n datapoints, where each datapoint $x_i \in \mathbb{R}^d$ in a d-dimensional vector space. In the MIPS setup, given a query $q \in \mathbb{R}^d$, we would like to find the datapoint $x \in X$ that has the highest inner product with q, i.e., we would like to identify

$$x_i^* := \underset{x_i \in X}{\arg\max} \langle q, x_i \rangle .$$

Exhaustively computing the exact inner product between q and n datapoints is often expensive and sometimes infeasible. Several techniques have been proposed in the literature based on hashing, graph search, or quantization to solve the approximate maximum inner product search problem efficiently, and the quantization based techniques have shown strong performance (Ge et al., 2014; Babenko & Lempitsky, 2014; Johnson et al., 2017).

In most traditional quantization works, the objective in the quantization procedures is to minimize the reconstruction error for the database points. We show this is a suboptimal loss function for MIPS. This is because for a given query, quantization error for database points that score higher, or have larger inner products, is more important. Using this intuition, we propose a new family of score-aware quantization loss functions and apply it to multiple quantization techniques. Our contributions are as follows:

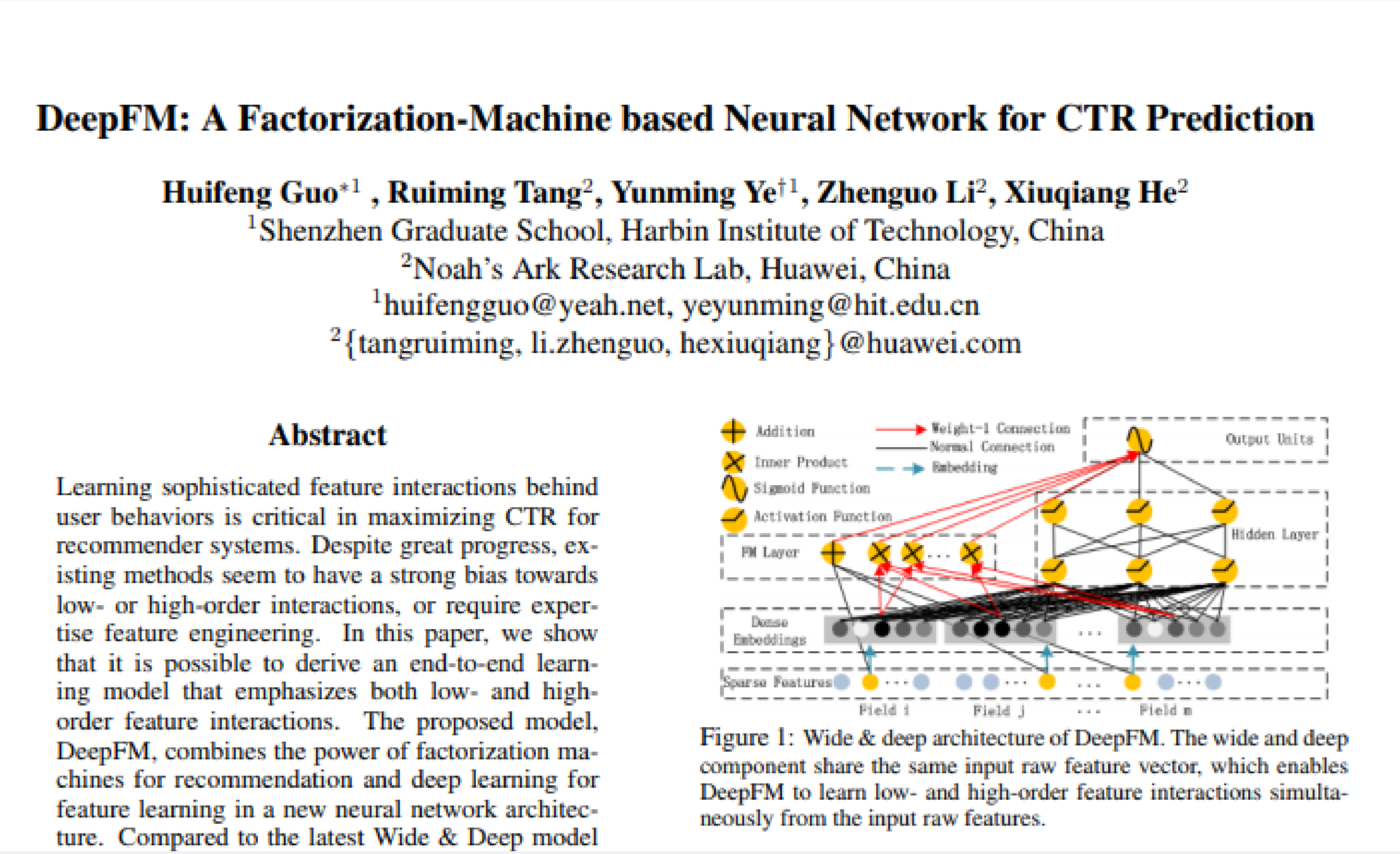
- We propose the score-aware quantization loss function. The proposed loss can work under any weighting function of the inner product and regardless of whether the datapoints vary in norm.

code walkthrough

bleeding edge alert!

deep factorization machines

the paper



DeepFM: A Factorization-Machine based Neural Network for CTR Prediction

Huifeng Guo*[1] , Ruiming Tang[2], Yunming Ye†[1], Zhenguo Li[2], Xiuqiang He[2]
[1]Shenzhen Graduate School, Harbin Institute of Technology, China
[2]Noah's Ark Research Lab, Huawei, China
[1]huifengguo@yeah.net, yeyunming@hit.edu.cn
[2]{tangruiming, li.zhenguo, hexiuqiang}@huawei.com

Abstract

Learning sophisticated feature interactions behind user behaviors is critical in maximizing CTR for recommender systems. Despite great progress, existing methods seem to have a strong bias towards low- or high-order interactions, or require expertise feature engineering. In this paper, we show that it is possible to derive an end-to-end learning model that emphasizes both low- and high-order feature interactions. The proposed model, DeepFM, combines the power of factorization machines for recommendation and deep learning for feature learning in a new neural network architecture. Compared to the latest Wide & Deep model

Addition
Inner Product
Sigmoid Function
Activation Function
Weight-1 Connection
Normal Connection
Embedding
Output Units
Hidden Layer
FM Layer
Dense Embeddings
Sparse Features
Field i
Field j
Field m

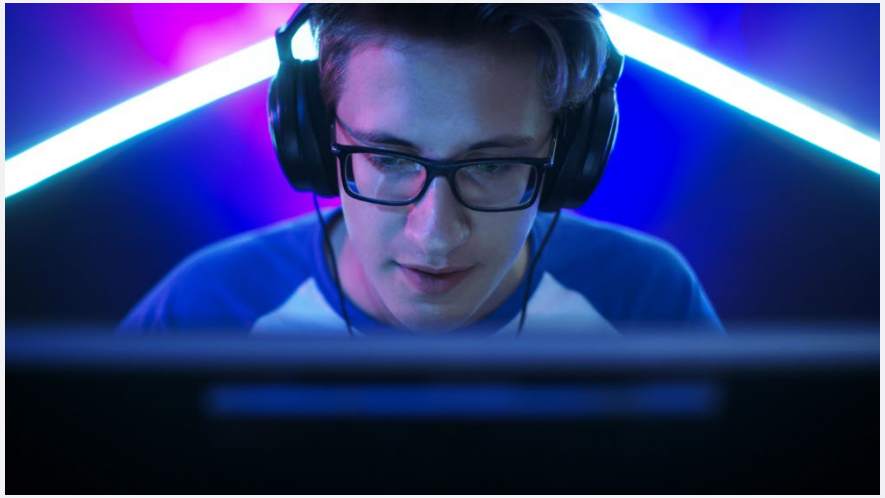
Figure 1: Wide & deep architecture of DeepFM. The wide and deep component share the same input raw feature vector, which enables DeepFM to learn low- and high-order feature interactions simultaneously from the input raw features.

higher-order feature interactions

FOOD
DELIVERY

+

- app category
- time



- app category
- gender
- age

deepfm architecture

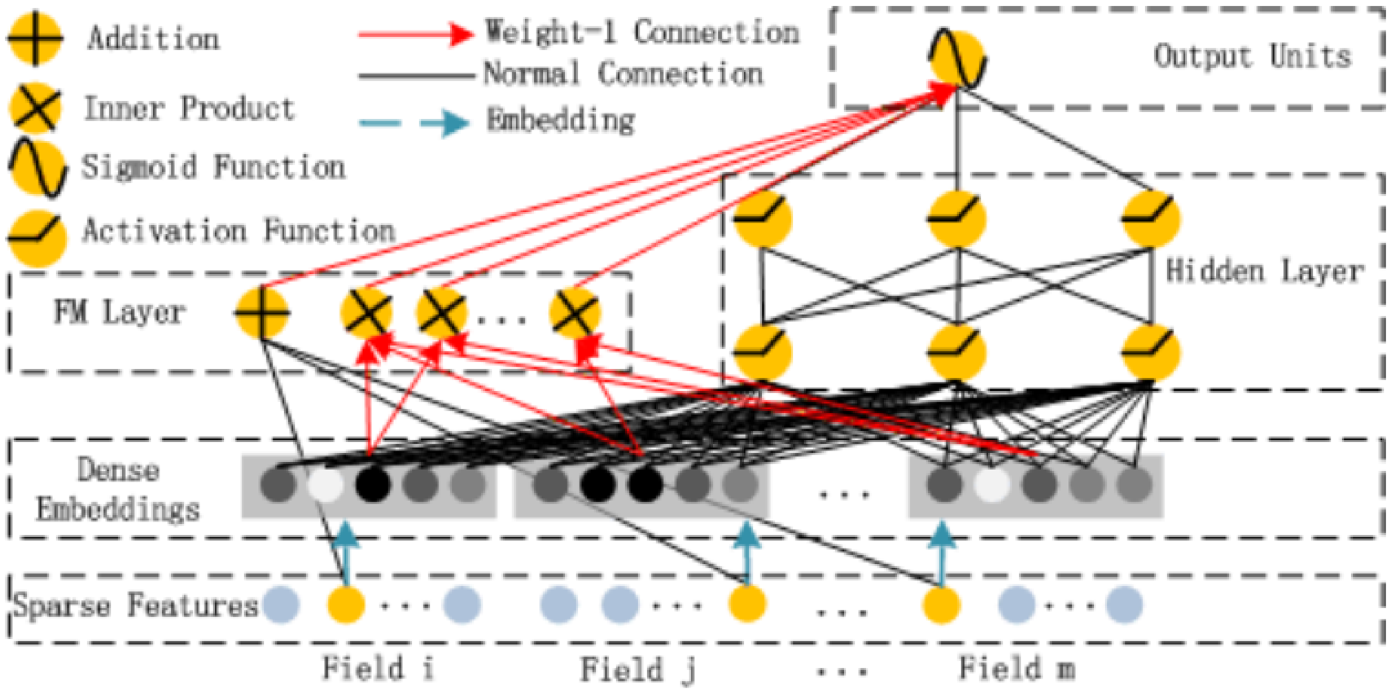

Addition
Inner Product
Sigmoid Function
Activation Function
Weight-1 Connection
Normal Connection
Embedding
Output Units
Hidden Layer
FM Layer
Dense Embeddings
Sparse Features
Field i
Field j
Field m
· · ·

Figure 1: Wide & deep architecture of DeepFM. The wide and deep component share the same input raw feature vector, which enables DeepFM to learn low- and high-order feature interactions simultaneously from the input raw features.

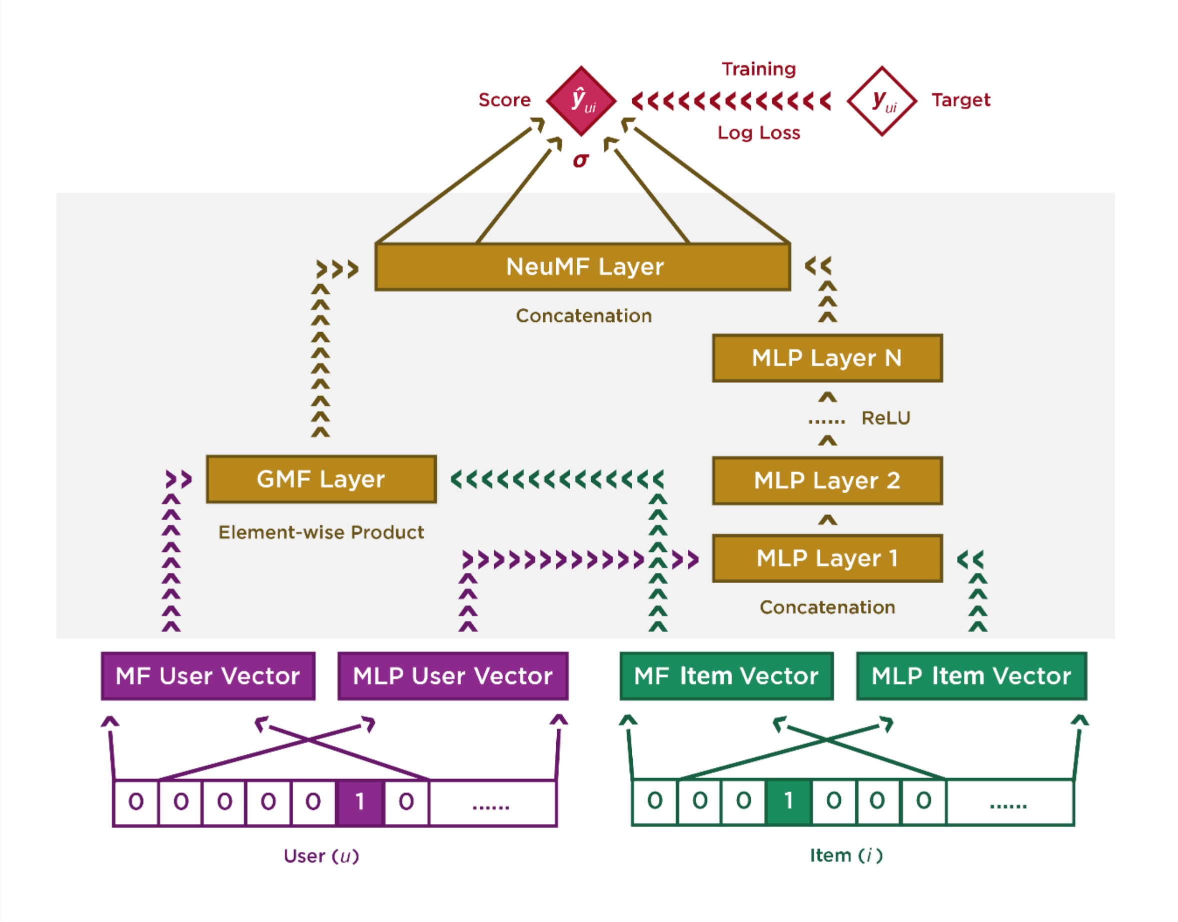
an ensemble approach

neural collaborative filtering

neural collaborative filtering (ncf)

- Combines the strengths of matrix factorization and neural networks
- Matrix factorization has no non-linear steps, and can’t capture non-linear relationships
- …But neural networks do!
- So, feed users and items through a Generalized Matrix Factorization (GMF) and a Multi-Layer Perceptron (MLP) in parallel
- Feed their outputs into a NeuMF layer that concatenates them

neural collaborative filtering


Score
$\hat{y}_{ui}$
Training
Log Loss
y_{ui}
Target
σ
NeuMF Layer
Concatenation
MLP Layer N
ReLU
MLP Layer 2
MLP Layer 1
Concatenation
GMF Layer
Element-wise Product
MF User Vector
MLP User Vector
MF Item Vector
MLP Item Vector
0 0 0 0 0 1 0 ......
User (u)
0 0 0 1 0 0 0 ......
Item (i)

introducing librecommender

- Like surpriselib, but with Tensorflow (or PyTorch, depending on the algorithm)
- Supports newer AI-based algorithms
  - Transformers!
  - GRU4Rec
  - YouTube
  - DeepFM
  - Item or user-based CF
  - ALS
  - Neural Collaborative Filtering
  - …and many more
- Hybrid recommenders with CF and content-based features
- Implicit or explicit data
- Includes both training (libreco) and serving (libserving) modules

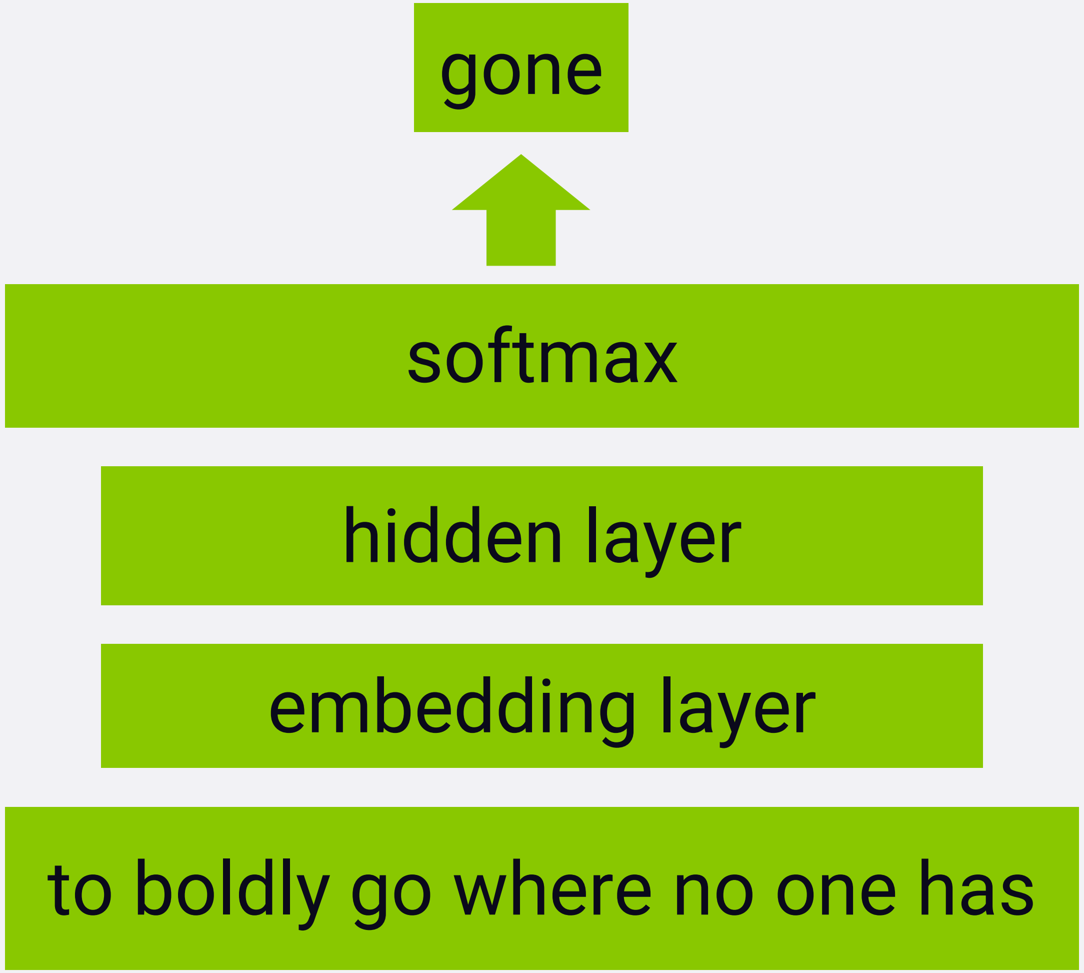
librecommender: simple example

```
train_data, eval_data, test_data = random_split(data, multi_ratios=[0.8, 0.1, 0.1])

  train_data, data_info = DatasetPure.build_trainset(train_data)
  eval_data = DatasetPure.build_evalset(eval_data)
  test_data = DatasetPure.build_testset(test_data)
  print(data_info)  # n_users: 5894, n_items: 3253, data sparsity: 0.4172 %

  lightgcn = LightGCN(
      task="ranking",
      data_info=data_info,
      loss_type="bpr",
      embed_size=16,
      n_epochs=3,
      lr=1e-3,
      batch_size=2048,
      num_neg=1,
      device="cuda",
  )
  # monitor metrics on eval_data during training
  lightgcn.fit(
      train_data,
      neg_sampling=True,  # sample negative items for train and eval data
      verbose=2,
      eval_data=eval_data,
      metrics=["loss", "roc_auc", "precision", "recall", "ndcg"],
  )

  # predict preference of user 2211 to item 110
  print("prediction: ", lightgcn.predict(user=2211, item=110))
  # recommend 7 items for user 2211
  print("recommendation: ", lightgcn.recommend_user(user=2211, n_rec=7))
```

code walkthrough

more technologies to watch

word2vec



gone

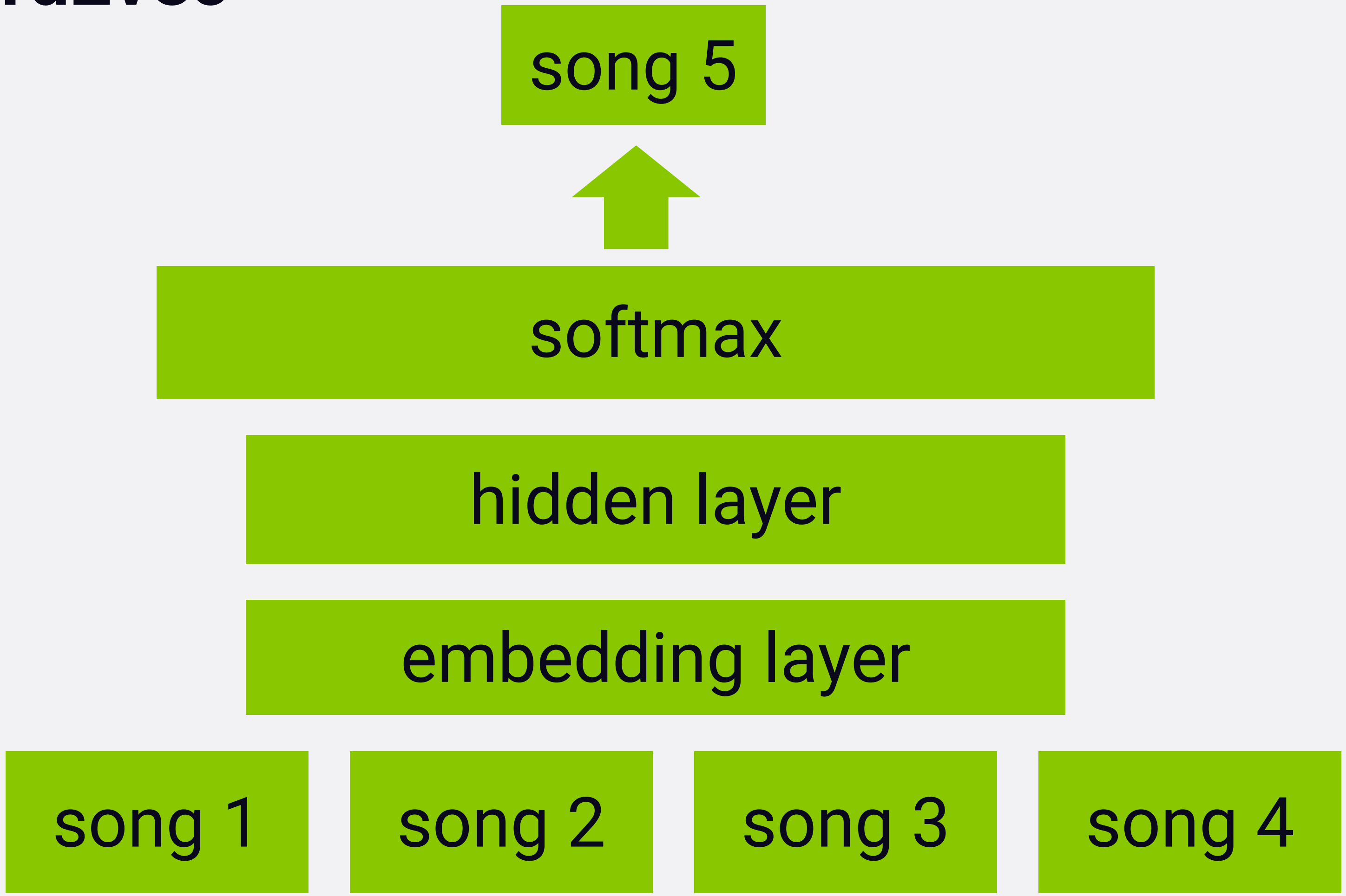
softmax

hidden layer

embedding layer

to boldly go where no one has

extending word2vec



song 5

softmax

hidden layer

embedding layer

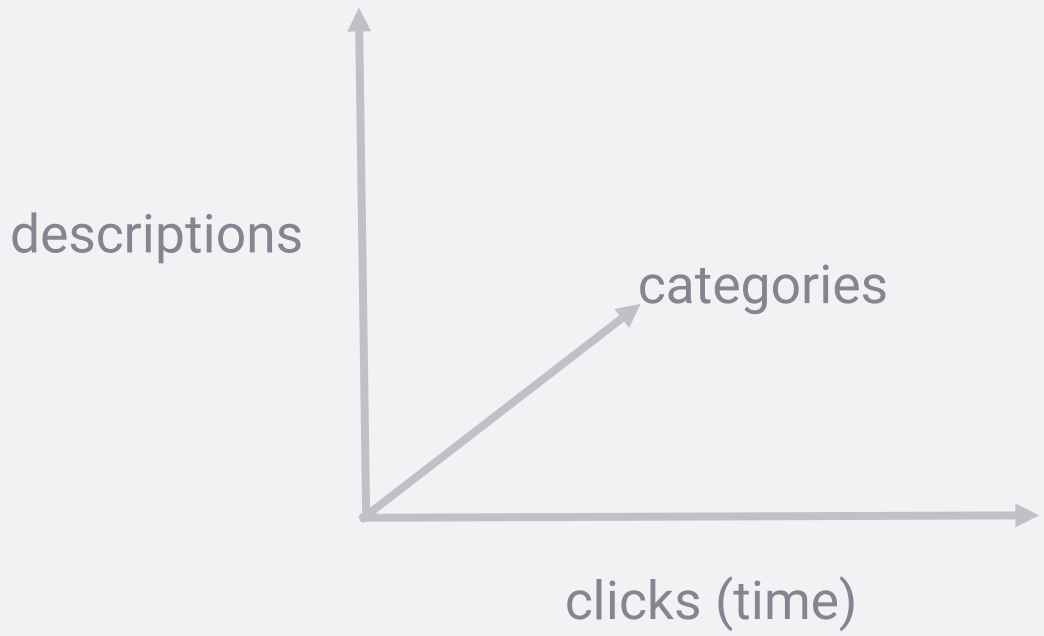
song 1

song 2

song 3

song 4

3D cnn's for session-based recs


descriptions
categories
clicks (time)

3D cnn's for session-based recs

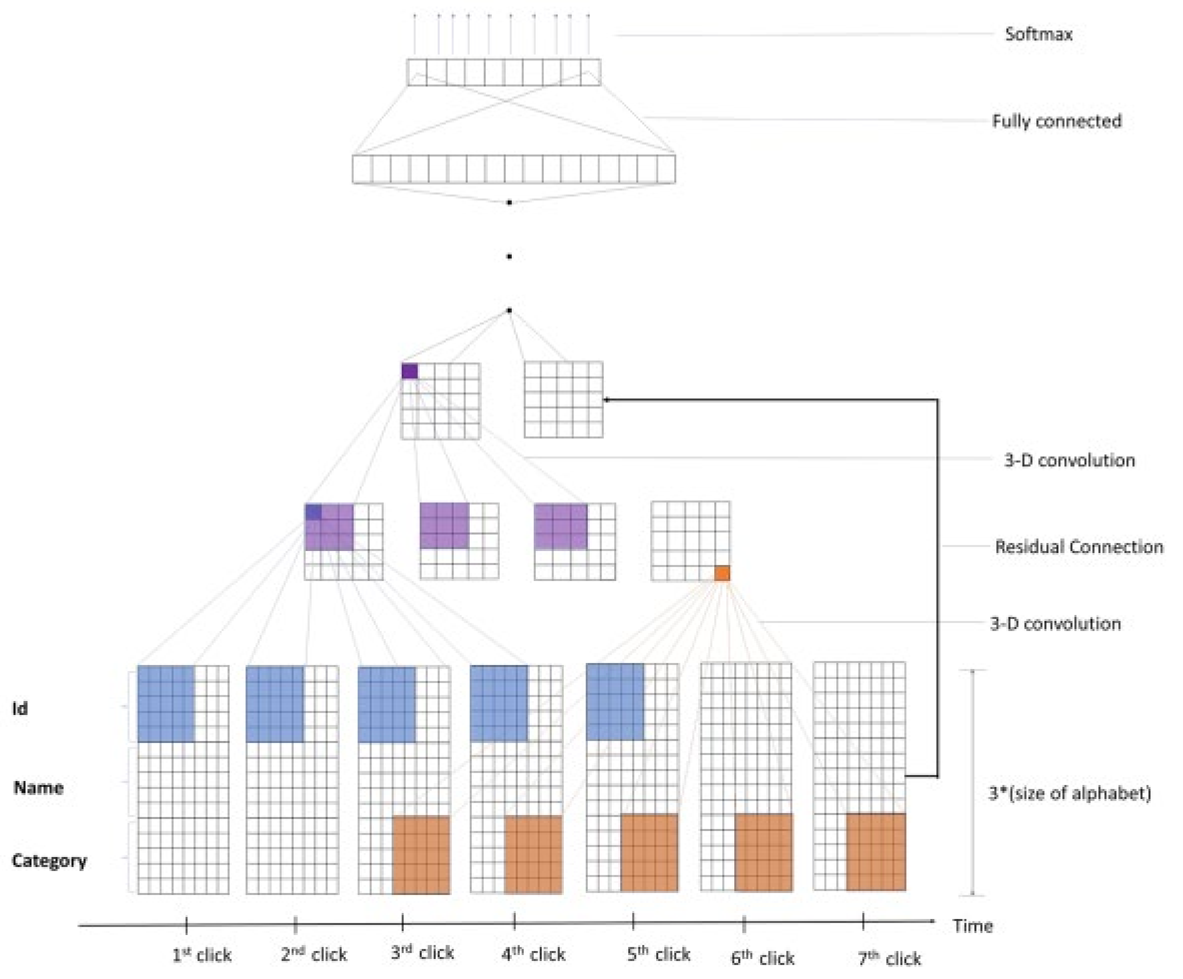
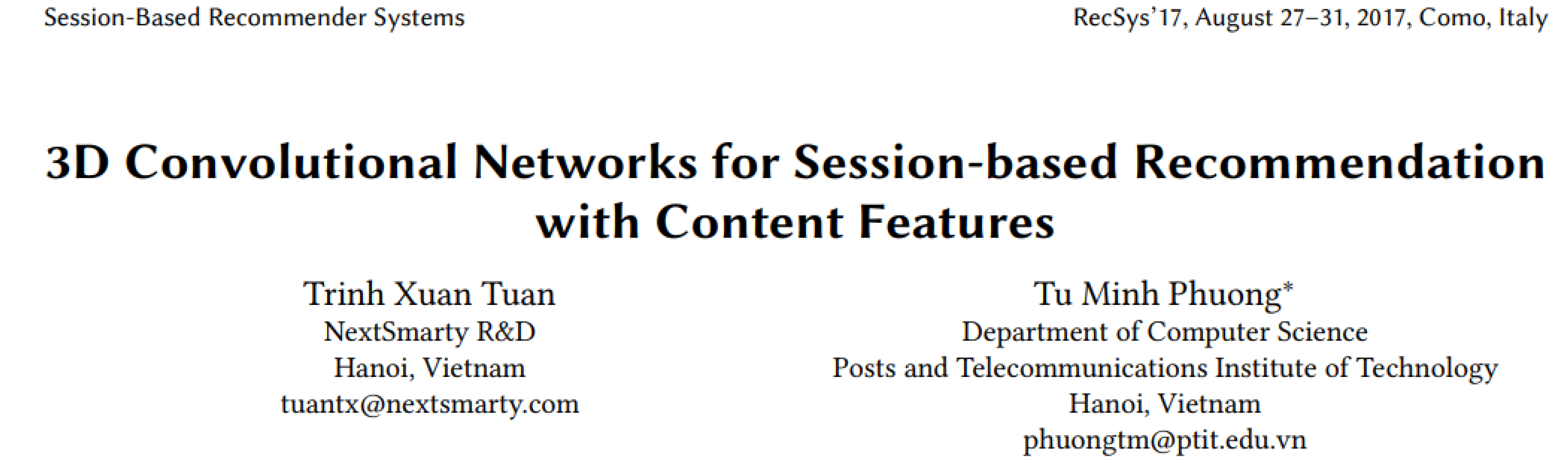

Softmax
Fully connected
3-D convolution
Residual Connection
3-D convolution
Id
Name
Category
3*(size of alphabet)
Time
1st click
2nd click
3rd click
4th click
5th click
6th click
7th click

Figure 2: Illustration of the first and last layers.

the paper



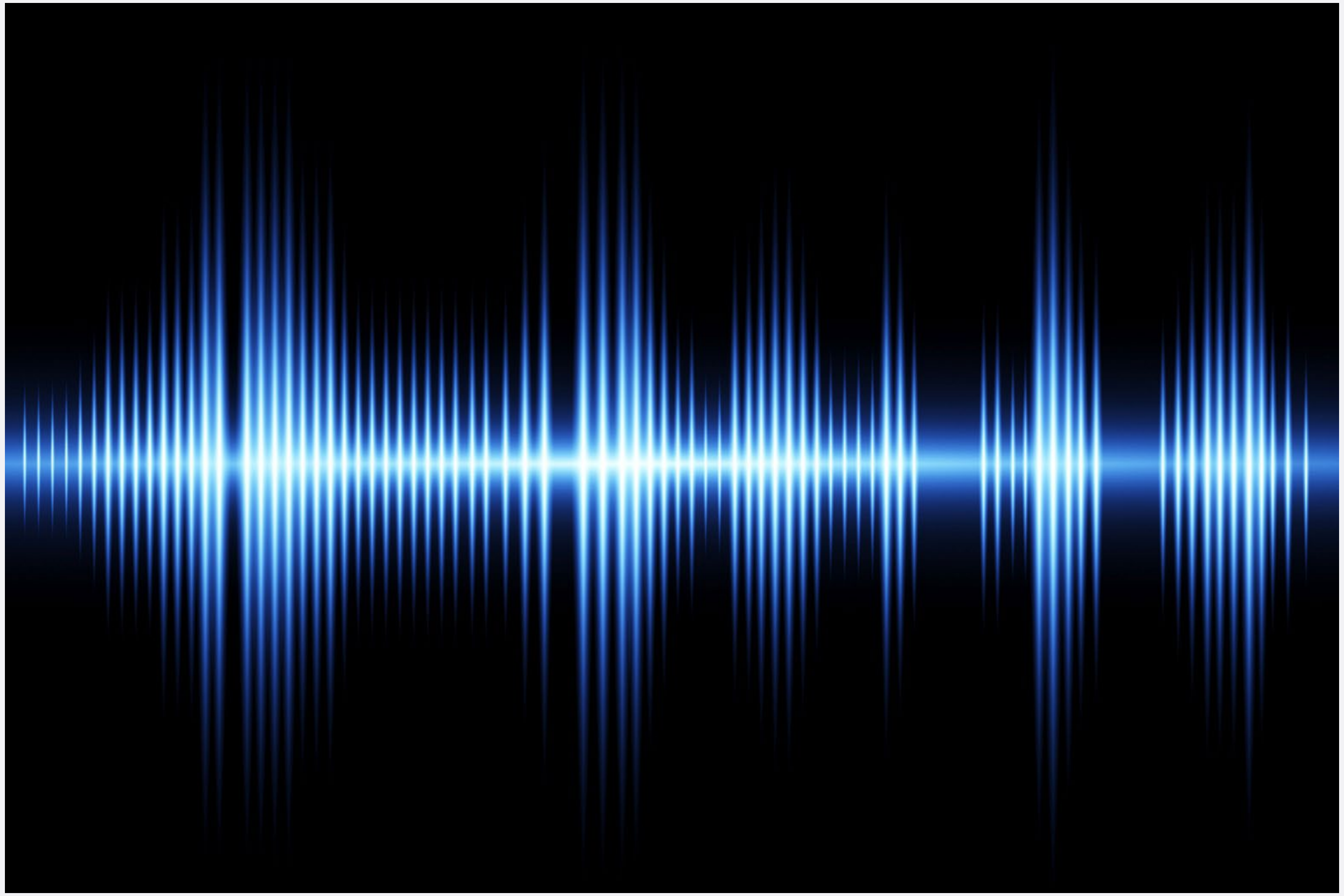
Session-Based Recommender Systems RecSys'17, August 27–31, 2017, Como, Italy

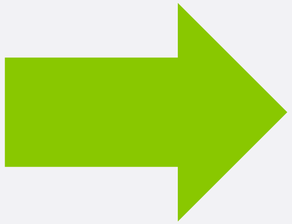
3D Convolutional Networks for Session-based Recommendation with Content Features

Trinh Xuan Tuan
NextSmarty R&D
Hanoi, Vietnam
tuantx@nextsmarty.com

Tu Minh Phuong*
Department of Computer Science
Posts and Telecommunications Institute of Technology
Hanoi, Vietnam
phuongtm@ptit.edu.vn

deep feature extraction with cnn's





classical

scaling it up

apache spark

installing spark (if you're brave)

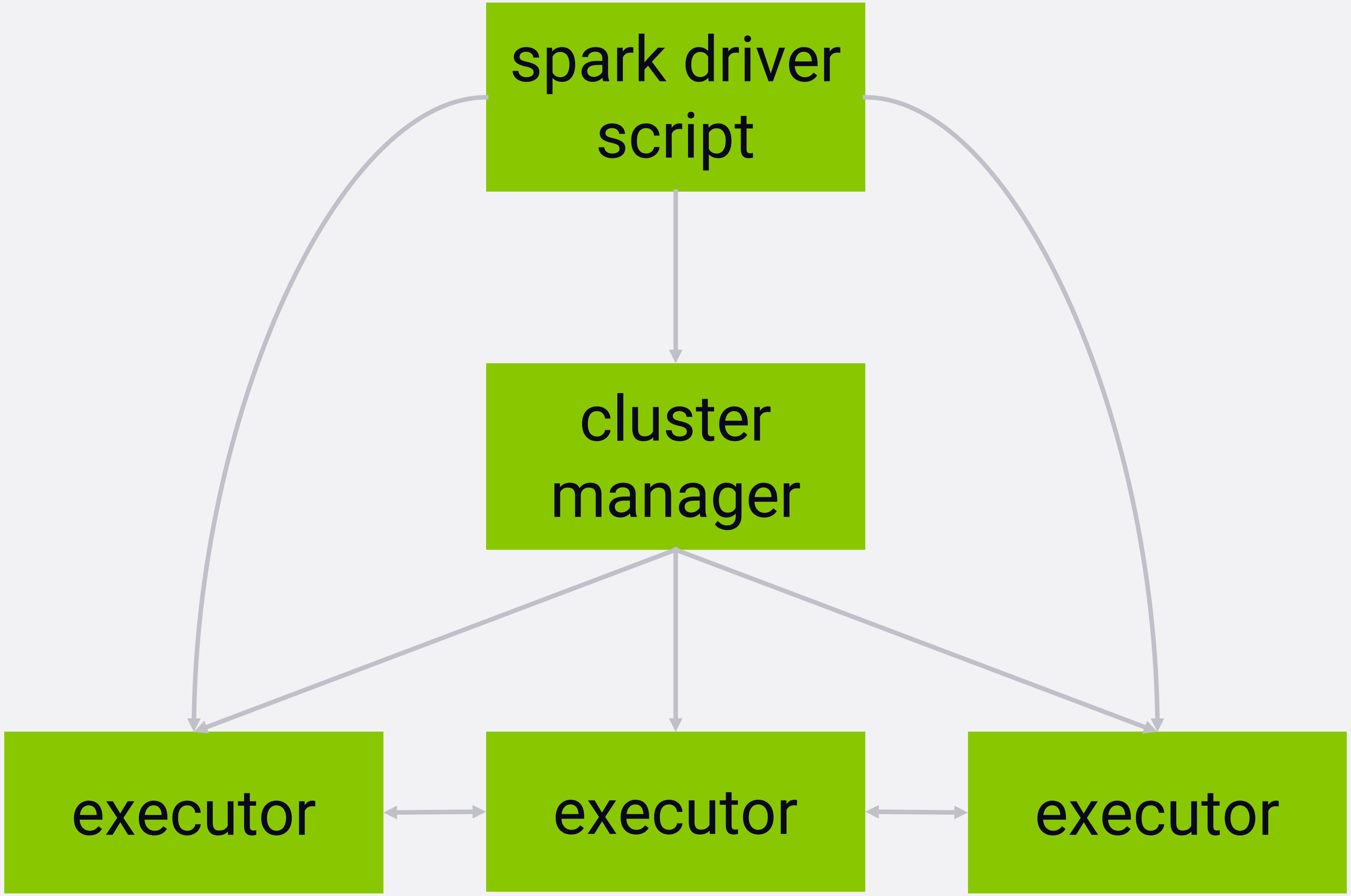
Install Java **8** SDK from Oracle to **c:\jdk**

Add **JAVA_HOME** environment variable to where you installed it
    Unix: export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
    Windows: Use the system control panel, and set JAVA_HOME to c:\jdk

Windows only:
    Create **C:\winutils\bin** and copy the **winutils.exe** file from the ScalingUp folder into it
    Set **HADOOP_HOME** environment variable to c:\winutils\
    Add %HADOOP_HOME%\bin to your PATH environment variable
    Restart your PC.

Install **pyspark** using Anaconda Navigator into your RecSys environment.

spark in a nutshell



spark driver script

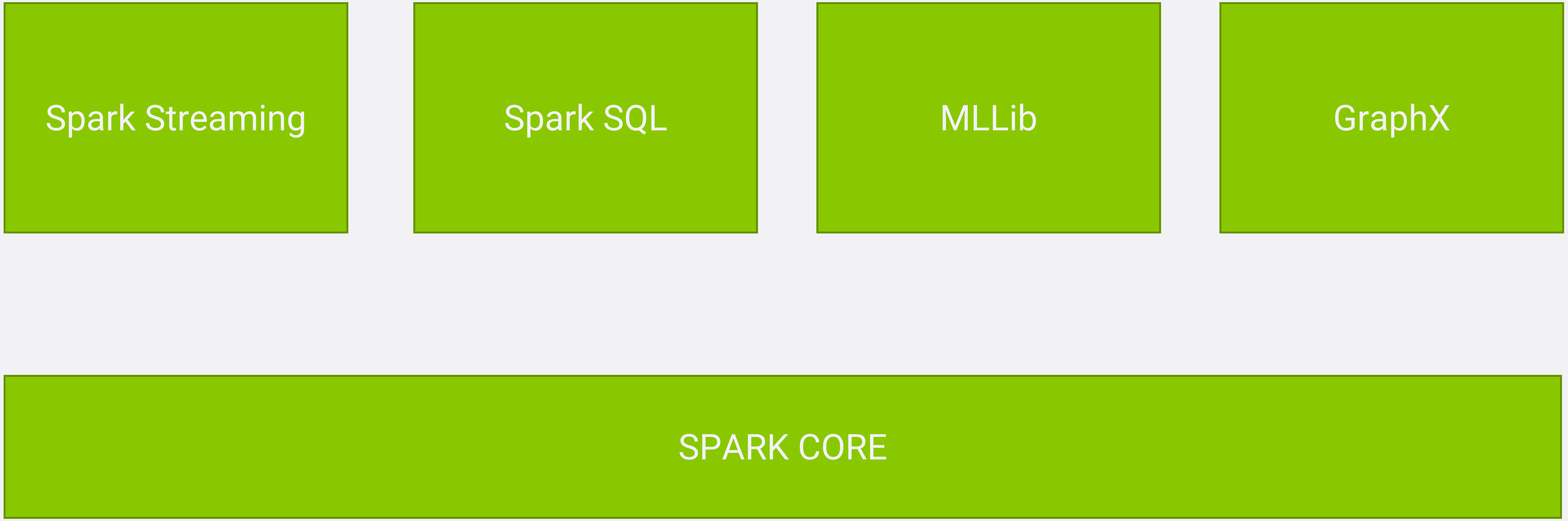
cluster manager

executor

executor

executor

spark software architecture



Spark Streaming

Spark SQL

MLLib

GraphX

SPARK CORE

rdd's

resilient

distributed

dataset

evolution of the spark api

RDD

jvm objects

DataFrame

row objects

DataSet

internally rows,
externally jvm objects

code walkthrough

code walkthrough

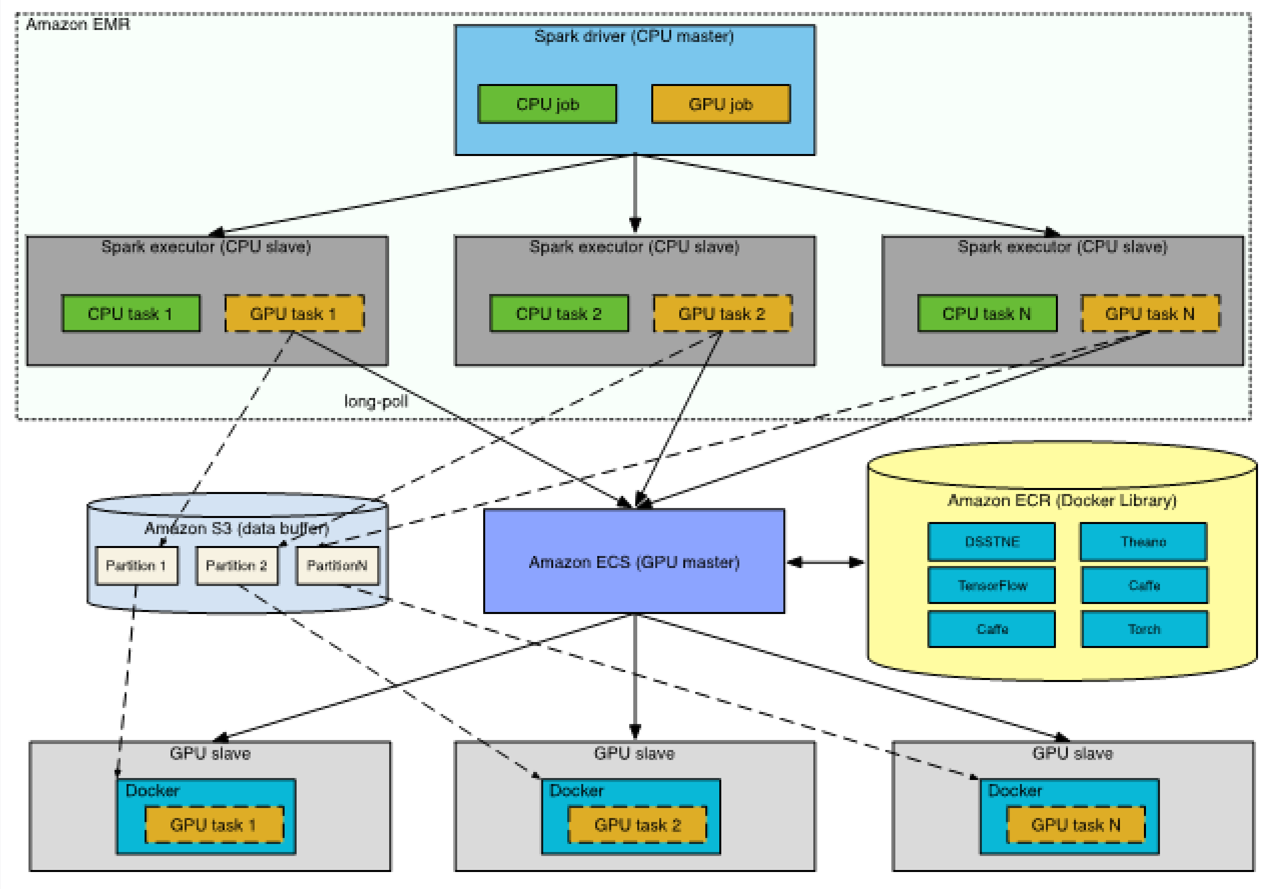
amazon dsstne

a sample config file

```
{
    "Version" : 0.7,
    "Name" : "AE",
    "Kind" : "FeedForward",
    "SparsenessPenalty" : {
        "p" : 0.5,
        "beta" : 2.0
    },

    "ShuffleIndices" : false,

    "Denoising" : {
        "p" : 0.2
    },

    "ScaledMarginalCrossEntropy" : {
        "oneTarget" : 1.0,
        "zeroTarget" : 0.0,
        "oneScale" : 1.0,
        "zeroScale" : 1.0
    },
    "Layers" : [
        { "Name" : "Input", "Kind" : "Input", "N" : "auto", "DataSet" : "gl_input", "Sparse" : true },
        { "Name" : "Hidden", "Kind" : "Hidden", "Type" : "FullyConnected", "N" : 128, "Activation" : "Sigmoid", "Sparse" : true },
        { "Name" : "Output", "Kind" : "Output", "Type" : "FullyConnected", "DataSet" : "gl_output", "N" : "auto", "Activation" : "Sigmoid", "Sparse" : true }
    ],

    "ErrorFunction" : "ScaledMarginalCrossEntropy"
}
```

code walkthrough

scaling up dsstne

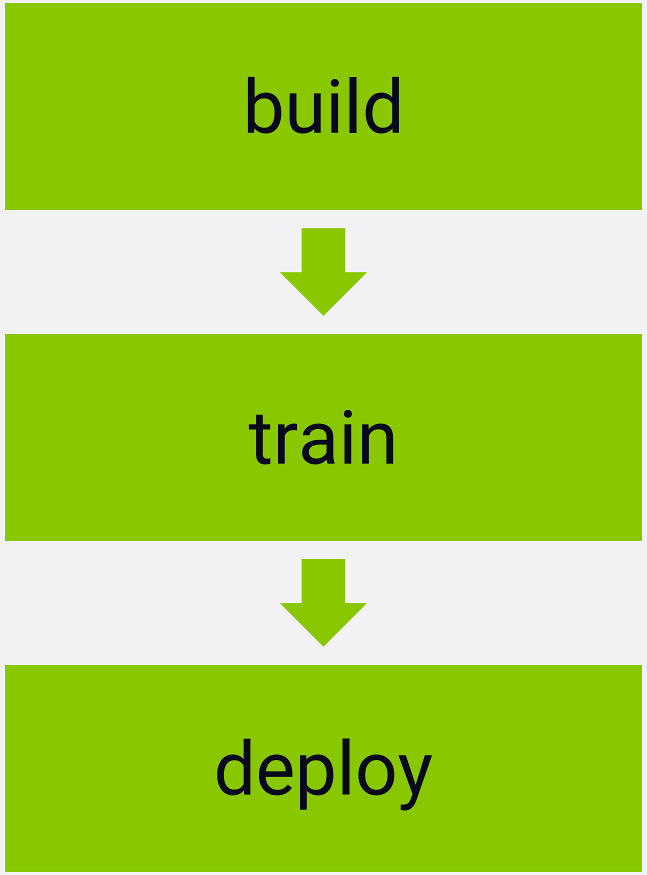

Amazon EMR
Spark driver (CPU master)
CPU job
GPU job
Spark executor (CPU slave)
CPU task 1
GPU task 1
Spark executor (CPU slave)
CPU task 2
GPU task 2
Spark executor (CPU slave)
CPU task N
GPU task N
long-poll
Amazon S3 (data buffer)
Partition 1
Partition 2
PartitionN
Amazon ECS (GPU master)
Amazon ECR (Docker Library)
DSSTNE
Theano
TensorFlow
Caffe
Caffe
Torch
GPU slave
Docker
GPU task 1
GPU slave
Docker
GPU task 2
GPU slave
Docker
GPU task N

learning more

https://amzn.to/2I69kAw

amazon sagemaker

sagemaker


build

train

deploy

movielens + sagemaker

load ml-1m ratings

one-hot encode user & movie

build binary label vector

convert to protobuf & write to s3

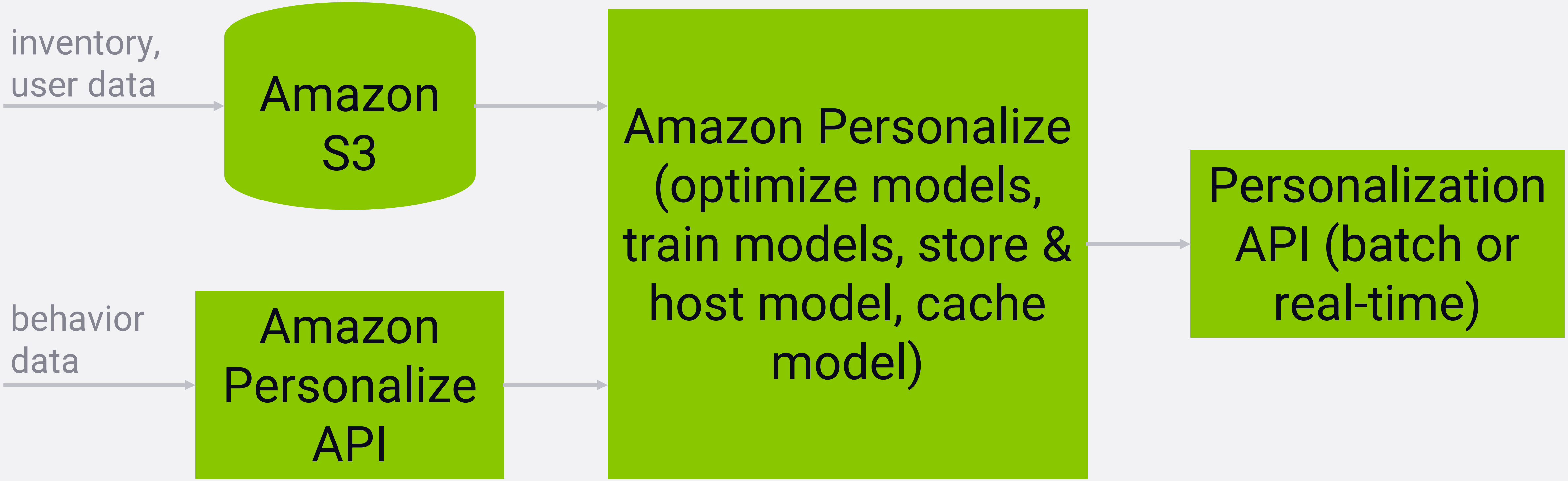
train, deploy, predict

code walkthrough

other systems of note

let's be clear about surpriselib

amazon personalize



inventory, user data

Amazon S3

behavior data

Amazon Personalize API

Amazon Personalize (optimize models, train models, store & host model, cache model)

Personalization API (batch or real-time)

recombee

AI-powered recommendation engine

RESTful API / SDK (JavaScript, Python, Node.js, PHP. Java, etc.)

you send it activity data, it gives you recommendations.

3 tiers of pricing based on usage ($99/mo - $1499/mo)

```
var client = new recombee.ApiClient('database-id', dbPublicToken); // Send a view of item 'item_x' by user 'user_42'

client.send( new recombee.AddDetailView('user_42', 'item_x')); // Get 5 recommended items for user 'user_42'. Recommend only items which haven't expired yet.

client.send( new recombee.RecommendItemsToUser('user_42', 5, {filter: " 'expires' > now()"}), (err, resp) => { // Show recommendations } );
```

predictionIO

apache, open-source machine learning server

not specifically for recommenders

simplifies deployment of web services to host trained models

similar in spirit to SageMaker

for recommendations, you’re limited to Apache Spark out of the box

- but you can add your own.

richrelevance

the granddaddy of hosted, personalization-as-a-service

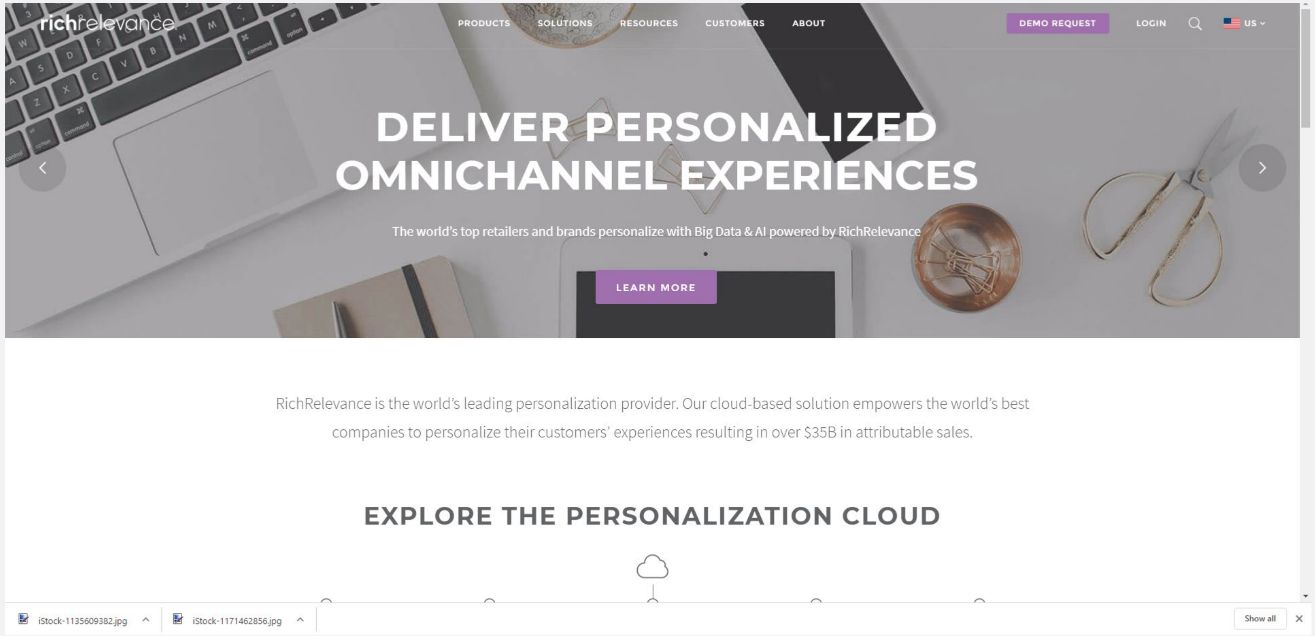
lots of big-name clients

started by some ex-Amazon guys

“Xen AI” – not just a black box

“personalization cloud” – personalized recs, nav, content, search

pricing: if you have to ask…


richrelevance
PRODUCTS
SOLUTIONS
RESOURCES
CUSTOMERS
ABOUT
DEMO REQUEST
LOGIN
US
LEARN MORE
iStock-1135609382.jpg
iStock-1171462856.jpg
Show all

DELIVER PERSONALIZED OMNICHANNEL EXPERIENCES

The world’s top retailers and brands personalize with Big Data & AI powered by RichRelevance

RichRelevance is the world’s leading personalization provider. Our cloud-based solution empowers the world’s best companies to personalize their customers’ experiences resulting in over $35B in attributable sales.

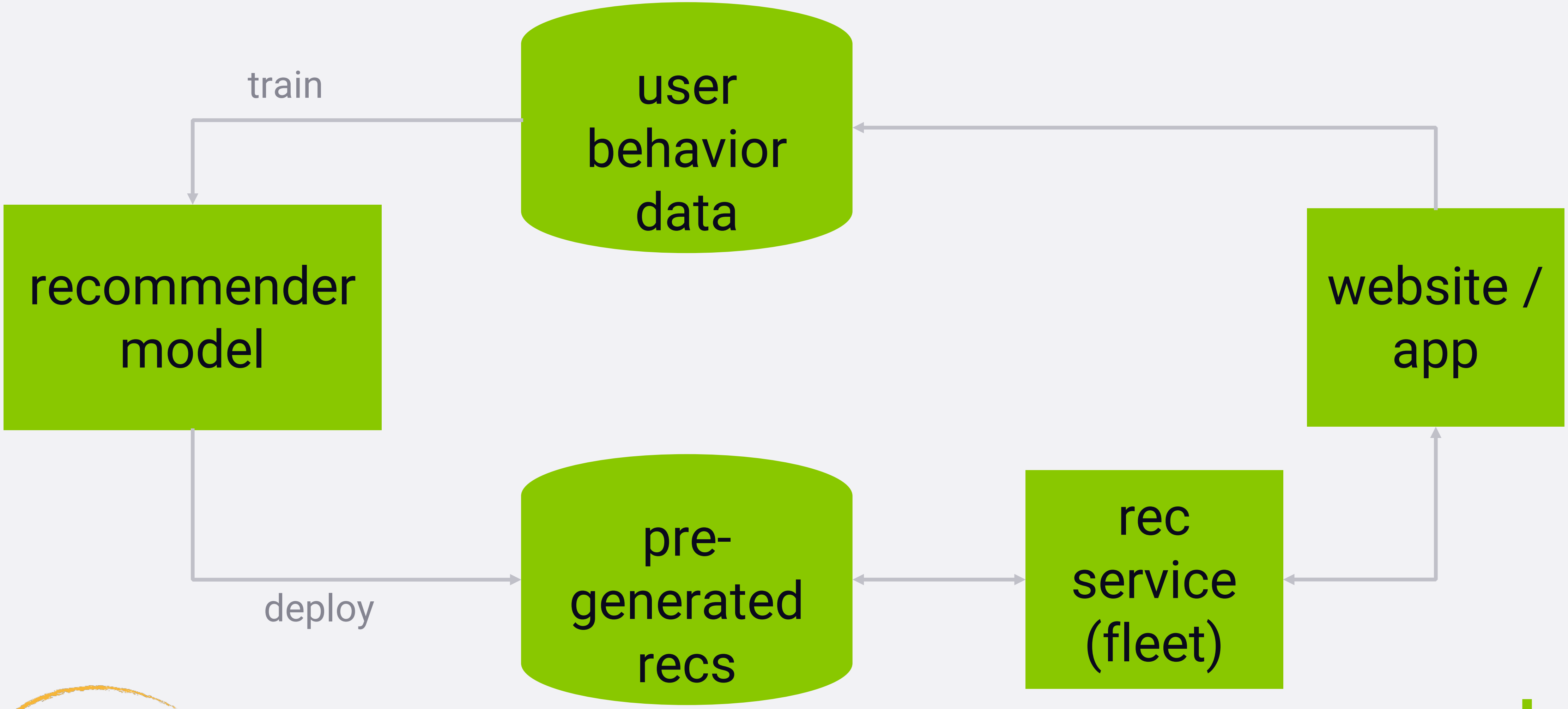
EXPLORE THE PERSONALIZATION CLOUD

many, many more

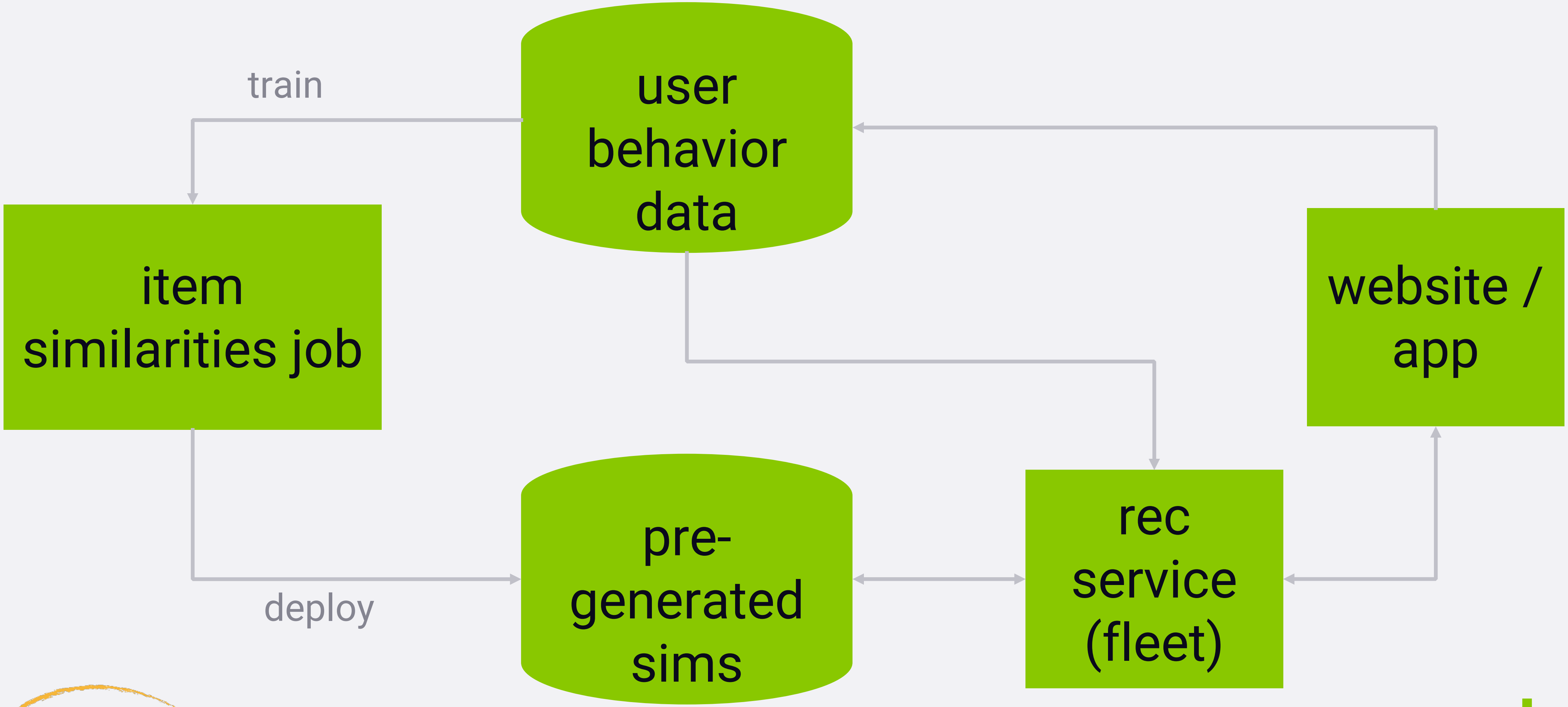
Peerius – Strands – SLI Systems – ParallelDots – Azure ML – Gravity R&D – Dressipi – Sajari – IBM Watson – Segmentify – Mr. Dlib – Raccoon – Universal Recommender – HapiGER – Mahout – RecDB – Oryx – Crab – LightFM – Rexy - QMF – Spotlight – tensorrec – hermes – CaseRecommender – ProbQA – Microsoft Recommenders – Gorse – Cornac - Devooght – LIBMF – RankSys – LibRec – Easyrec – Lenskit – Apache Giraph

system architecture

recommendations in the real world: pre-computed recs


train
user behavior data
recommender model
website / app
deploy
pre-generated recs
rec service (fleet)

recommendations in the real world: real-time collaborative filtering


train

user behavior data

item similarities job

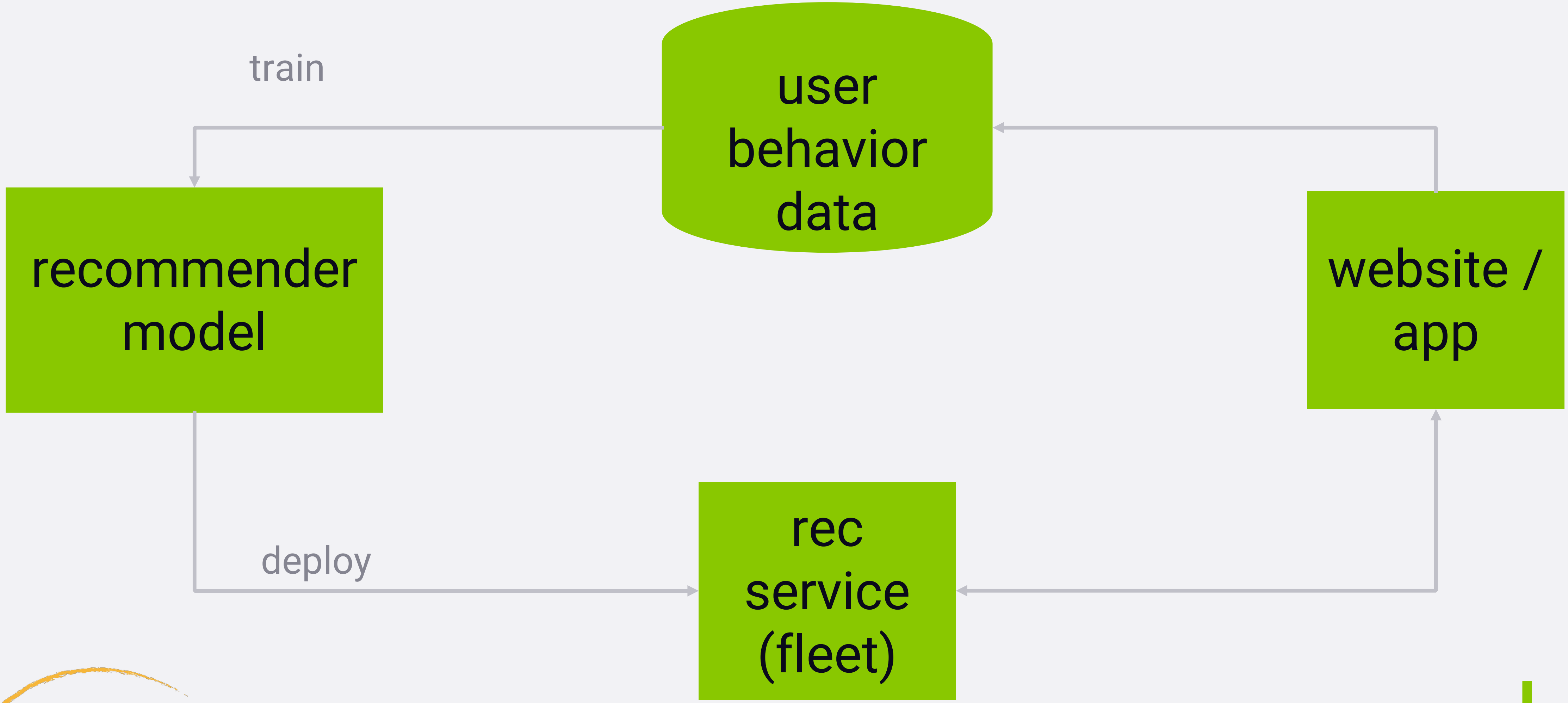
website / app

rec service (fleet)

pre-generated sims

deploy

recommendations in the real world: deploy a trained model



train

user behavior data

recommender model

website / app

deploy

rec service (fleet)

the cold-start problem

cold-start: new user solutions

- use implicit data
- use cookies (carefully)
- geo-ip
- recommend top-sellers or promotions
- interview the user

cold-start: new item solutions

- just don’t worry about it
- use content-based attributes
- map attributes to latent features (see LearnAROMA)
- random exploration

exercise: random exploration

code walkthrough

stoplists

things you might stoplist

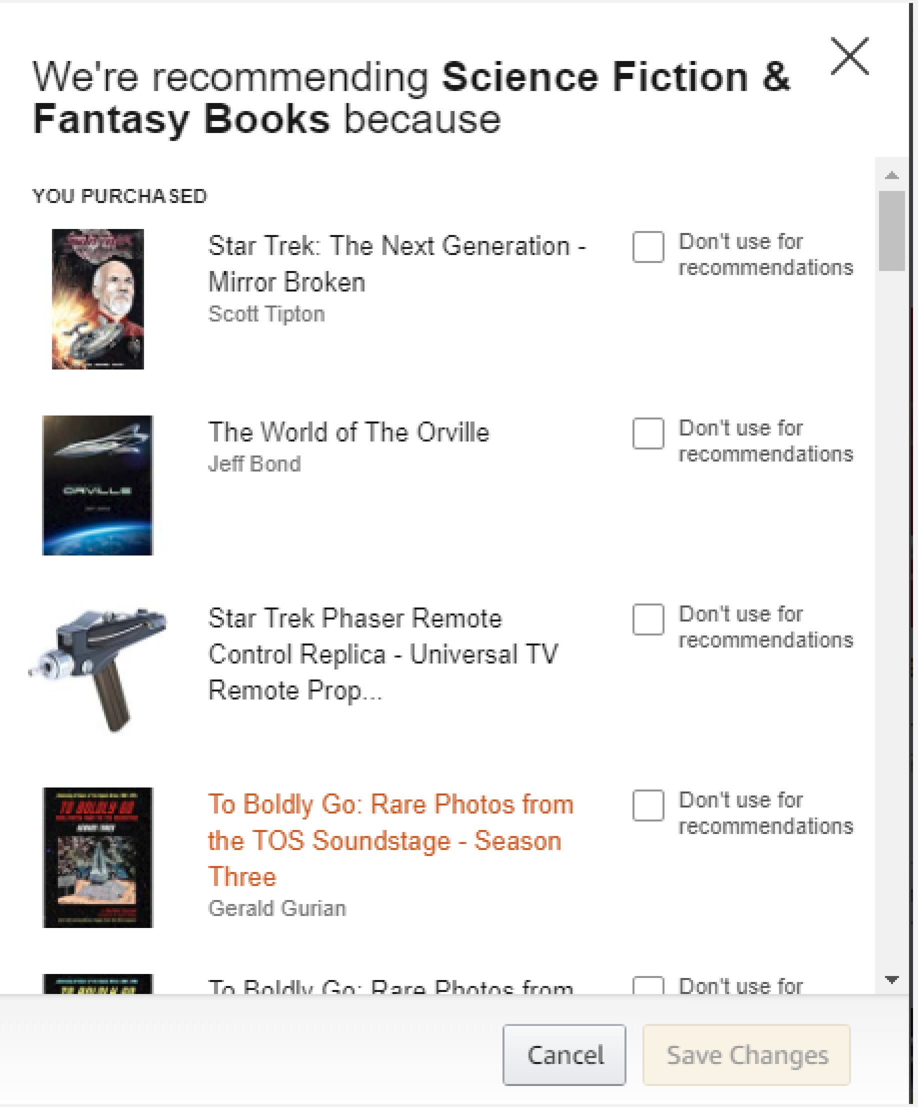
- adult-oriented content
- vulgarity
- legally prohibited topics (i.e. Mein Kampf)
- terrorism / political extremism
- bereavement / medical
- competing products
- drug use
- religion

exercise: implement a stoplist

code walkthrough

filter bubbles

transparency and trust



We're recommending **Science Fiction & Fantasy Books** because

YOU PURCHASED

Star Trek: The Next Generation - Mirror Broken
Scott Tipton
☐ Don't use for recommendations

The World of The Orville
Jeff Bond
☐ Don't use for recommendations

Star Trek Phaser Remote Control Replica - Universal TV Remote Prop...
☐ Don't use for recommendations

To Boldly Go: Rare Photos from the TOS Soundstage - Season Three
Gerald Gurian
☐ Don't use for recommendations

To Boldly Go: Rare Photos from
☐ Don't use for

outliers

exercise: filtering outliers

code walkthrough

gaming the system

implicit data, explicit problems.

international markets and laws

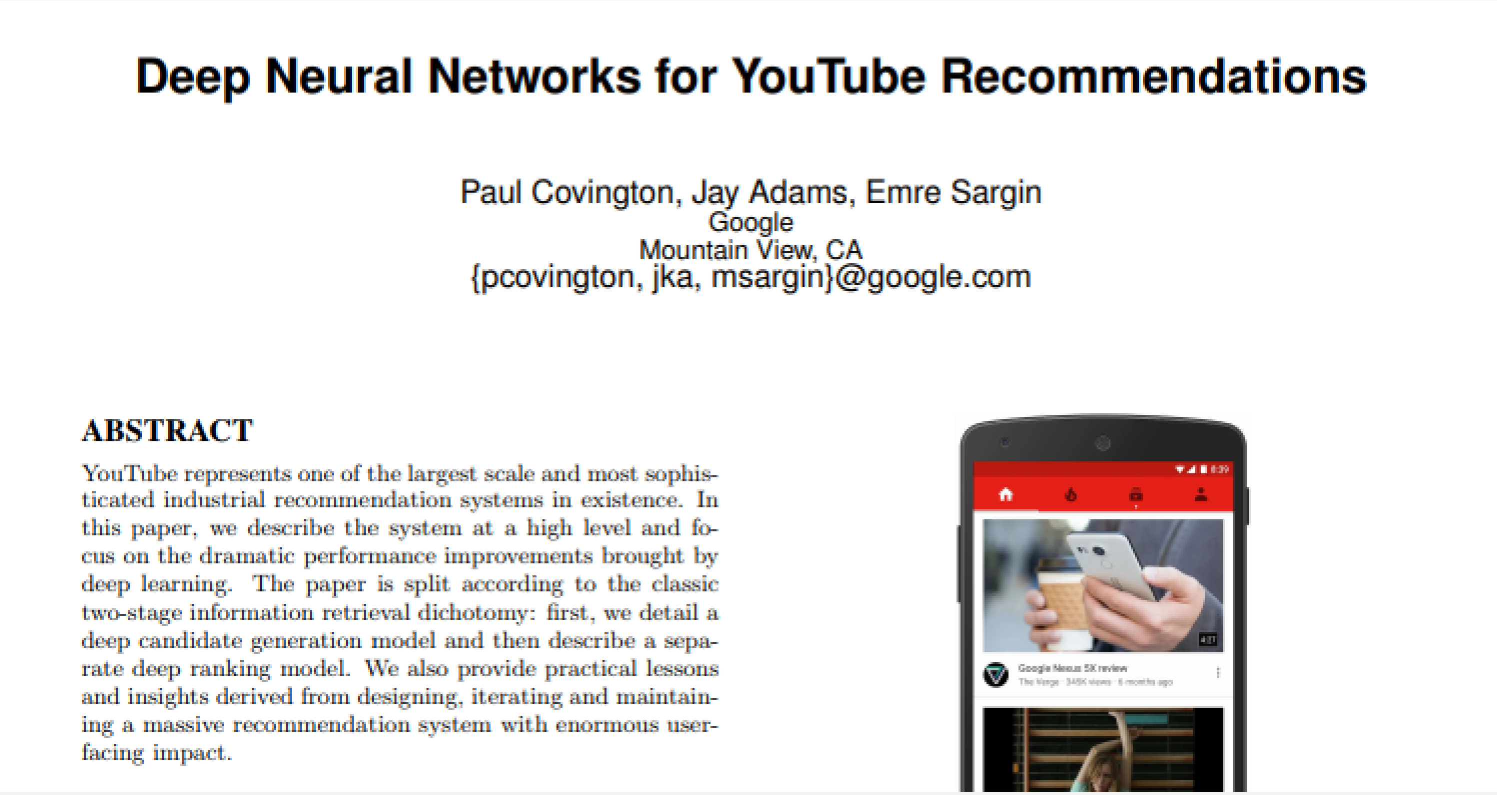
dealing with time

value-aware recommendations

case studies

youtube

the paper



Deep Neural Networks for YouTube Recommendations

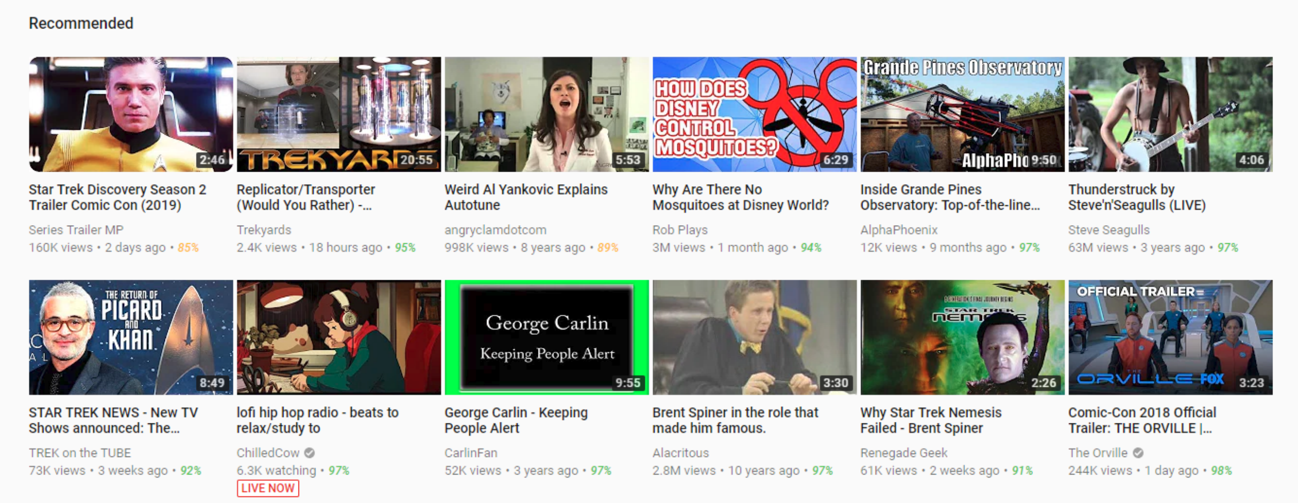
Paul Covington, Jay Adams, Emre Sargin
Google
Mountain View, CA
{pcovington, jka, msargin}@google.com

ABSTRACT

YouTube represents one of the largest scale and most sophisticated industrial recommendation systems in existence. In this paper, we describe the system at a high level and focus on the dramatic performance improvements brought by deep learning. The paper is split according to the classic two-stage information retrieval dichotomy: first, we detail a deep candidate generation model and then describe a separate deep ranking model. We also provide practical lessons and insights derived from designing, iterating and maintaining a massive recommendation system with enormous user-facing impact.

youtube's challenges

- scale
- freshness
- noise


2:46
TREKYARDS
20:55
5:53
HOW DOES DISNEY CONTROL MOSQUITOES?
6:29
Grande Pines Observatory
AlphaPho
9:50
4:06
THE RETURN OF PICARD AND KHAN
8:49
George Carlin
Keeping People Alert
9:55
3:30
STAR TREK NEMESIS
2:26
OFFICIAL TRAILER
ORVILLE
FOX
3:23

Recommended

Star Trek Discovery Season 2 Trailer Comic Con (2019)
Series Trailer MP
160K views • 2 days ago • 85%

Replicator/Transporter (Would You Rather) -...
Trekyards
2.4K views • 18 hours ago • 95%

Weird Al Yankovic Explains Autotune
angryclamdotcom
998K views • 8 years ago • 89%

Why Are There No Mosquitoes at Disney World?
Rob Plays
3M views • 1 month ago • 94%

Inside Grande Pines Observatory: Top-of-the-line...
AlphaPhoenix
12K views • 9 months ago • 97%

Thunderstruck by Steve'n'Seagulls (LIVE)
Steve Seagulls
63M views • 3 years ago • 97%

STAR TREK NEWS - New TV Shows announced: The...
TREK on the TUBE
73K views • 3 weeks ago • 92%

lofi hip hop radio - beats to relax/study to
ChilledCow ✓
6.3K watching • 97%
LIVE NOW

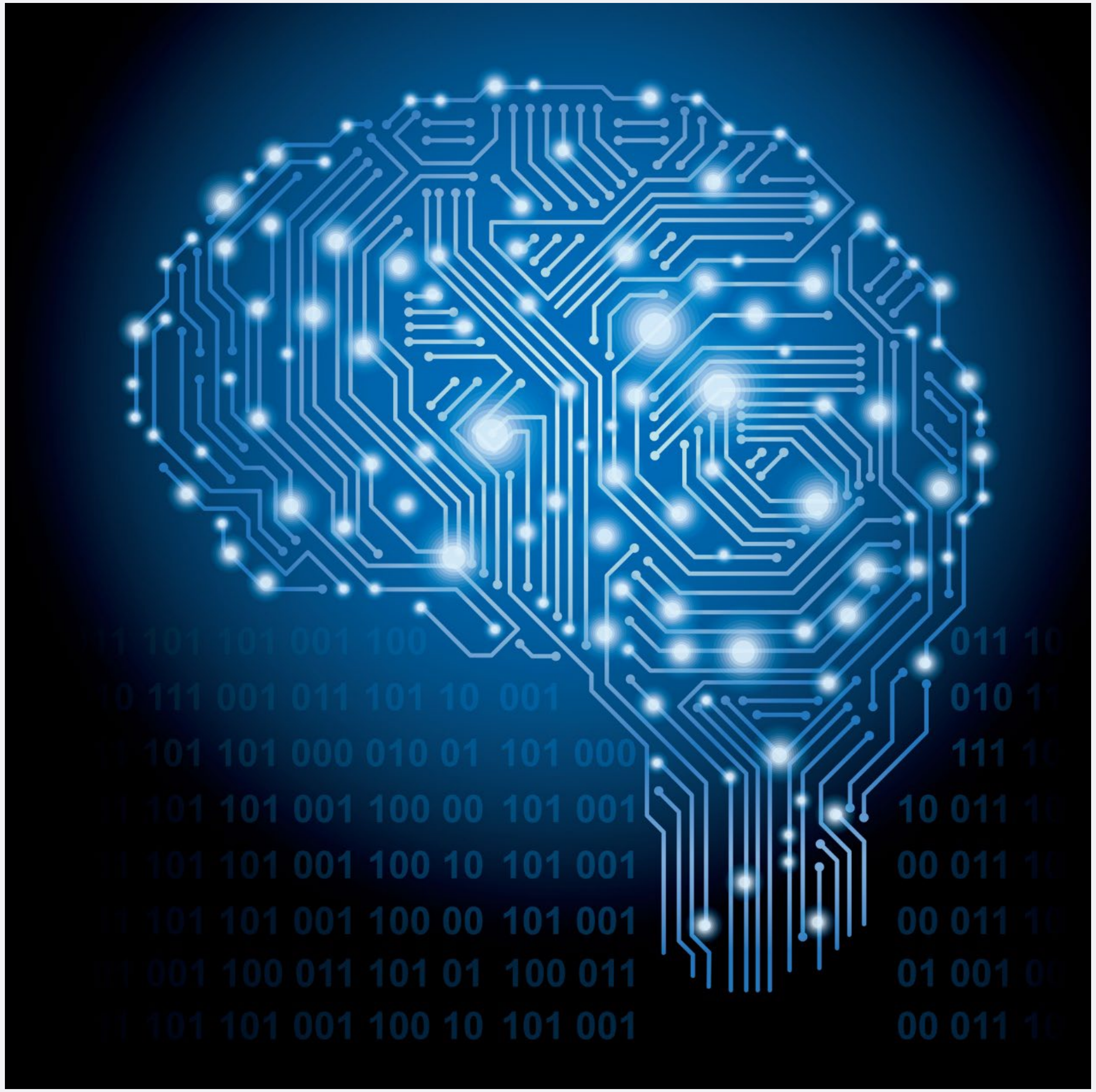
George Carlin - Keeping People Alert
CarlinFan
52K views • 3 years ago • 97%

Brent Spiner in the role that made him famous.
Alacritous
2.8M views • 10 years ago • 97%

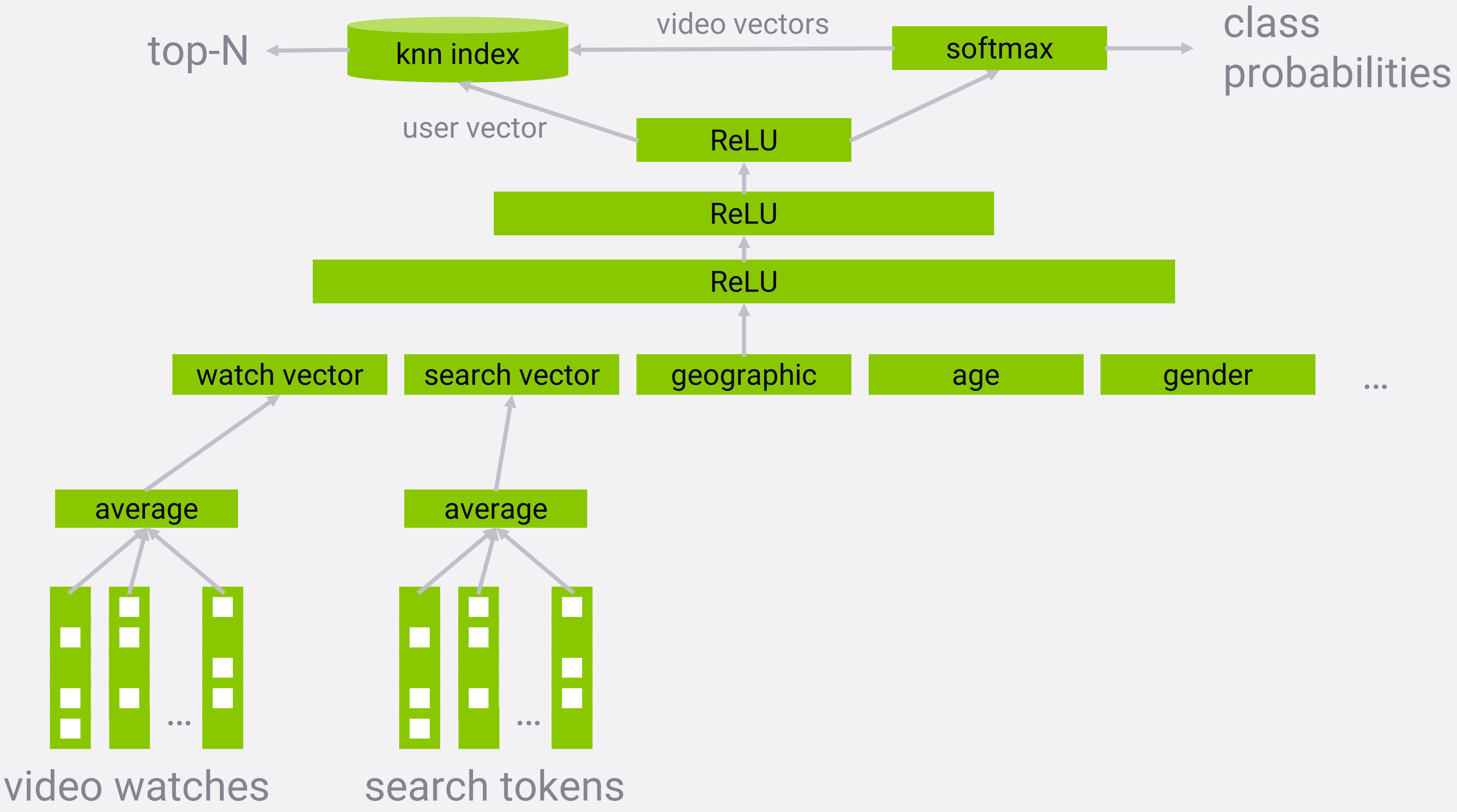
Why Star Trek Nemesis Failed - Brent Spiner
Renegade Geek
61K views • 2 weeks ago • 91%

Comic-Con 2018 Official Trailer: THE ORVILLE |...
The Orville ✓
244K views • 1 day ago • 98%

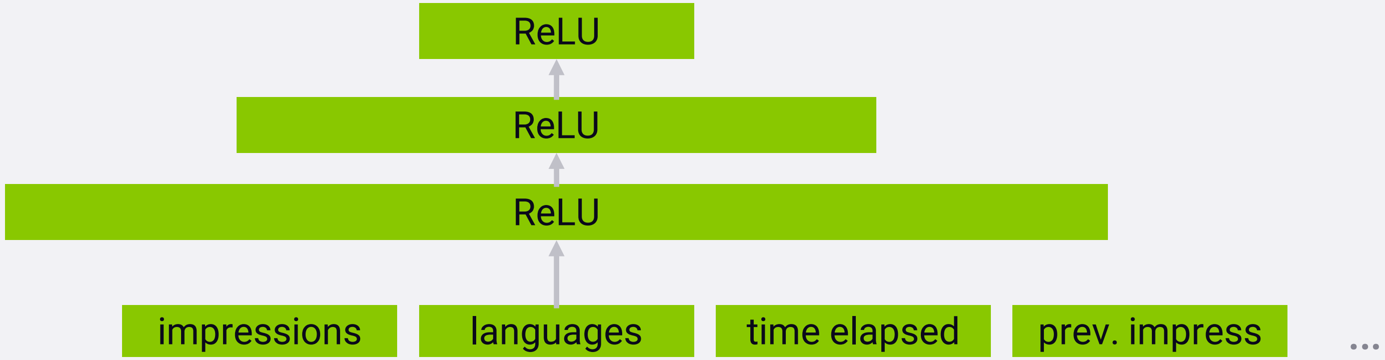
youtube's (and google's) answer to everything



youtube's candidate generation


top-N
knn index
video vectors
softmax
class probabilities
user vector
ReLU
ReLU
ReLU
watch vector
search vector
geographic
age
gender
…
average
average
…
…
video watches
search tokens

learning to rank


ReLU

ReLU

ReLU

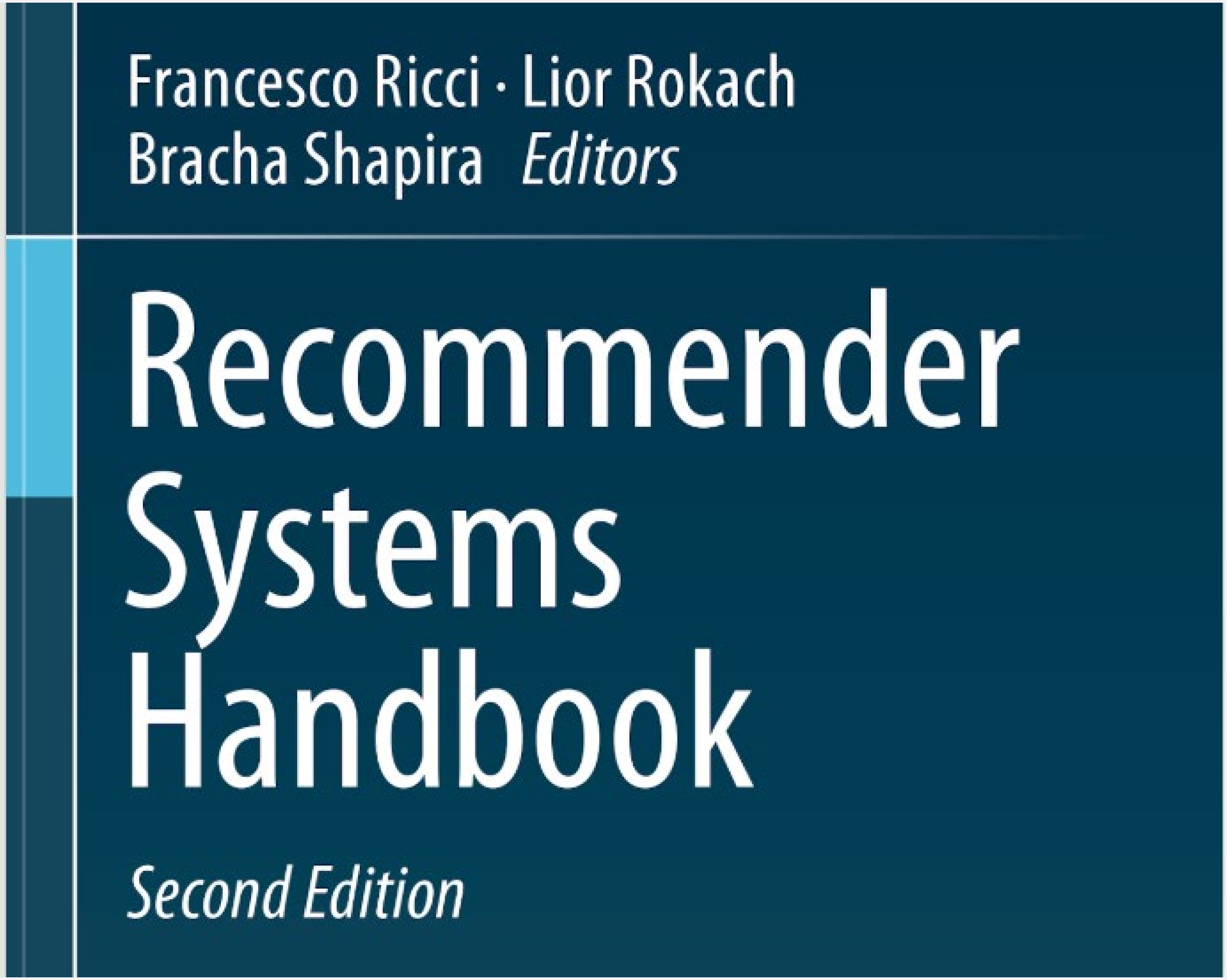
impressions languages time elapsed prev. impress …

learnings from youtube

- don’t train just on views
- withhold information
- dealing with series
- rank by consumption, not clicks
- learning-to-rank

netflix

netflix sources


Francesco Ricci · Lior Rokach
Bracha Shapira *Editors*

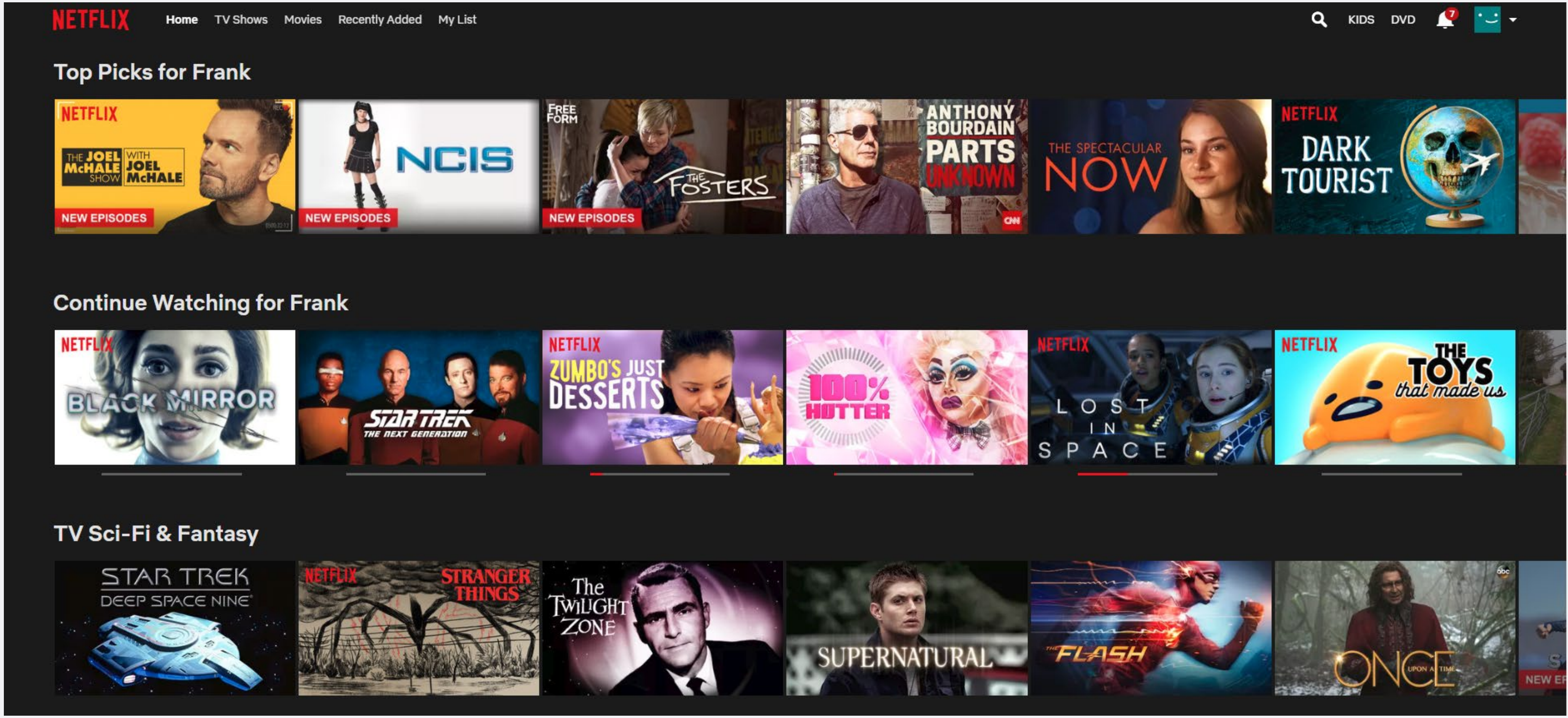
Recommender Systems Handbook

Second Edition

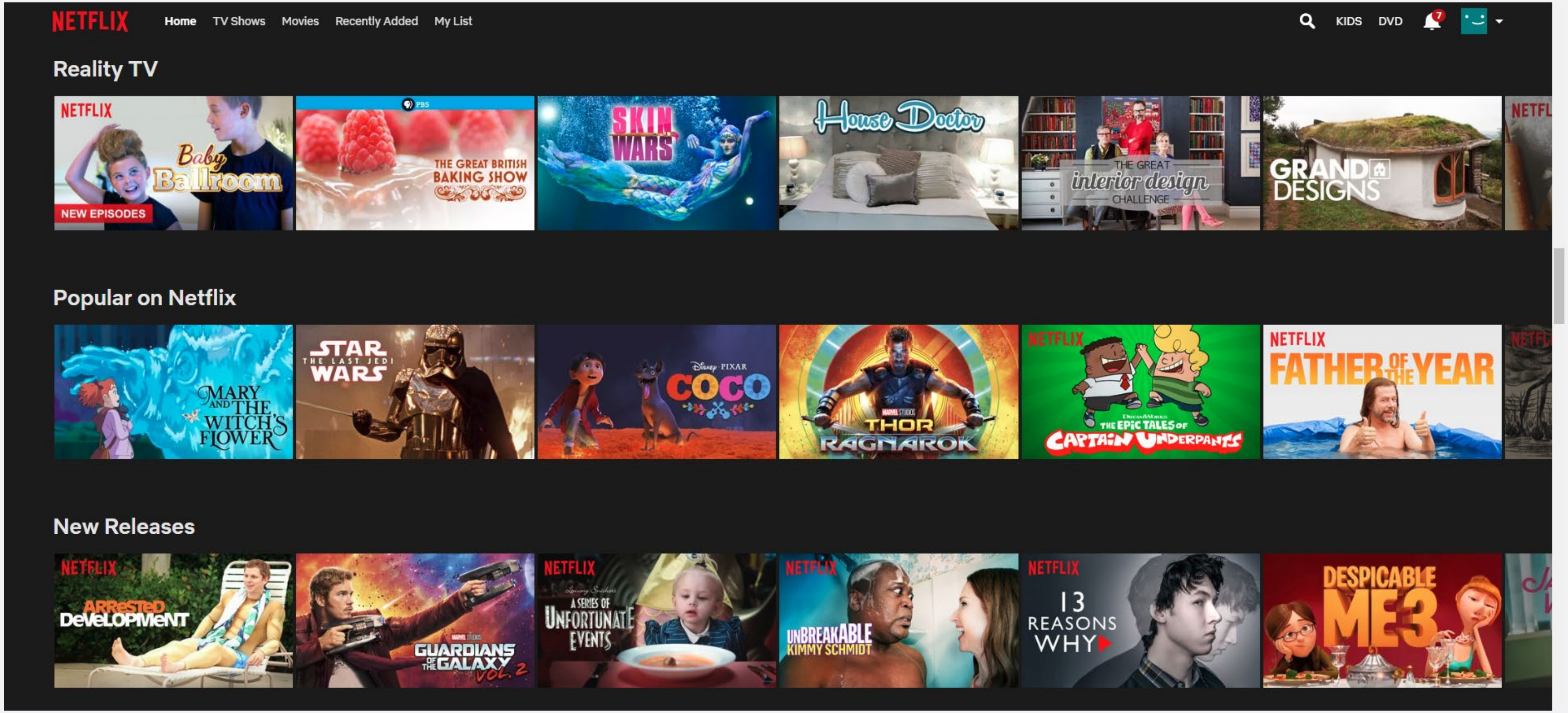
what model does netflix use?

all of them!

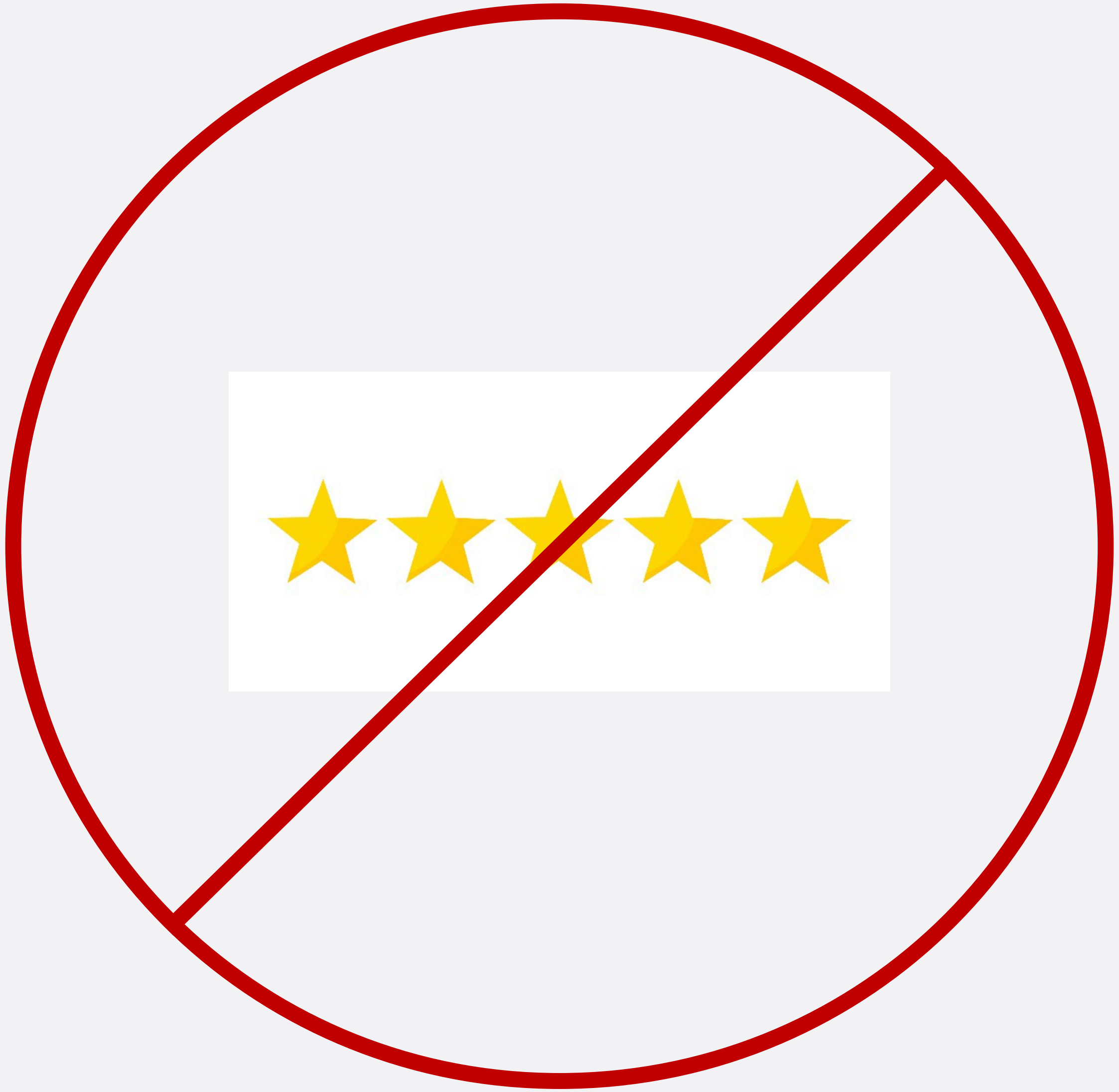
everything is a recommendation


NETFLIX
Home
TV Shows
Movies
Recently Added
My List
KIDS
DVD
Top Picks for Frank
Continue Watching for Frank
TV Sci-Fi & Fantasy
NEW EPISODES

whole-page optimization


NETFLIX
Home
TV Shows
Movies
Recently Added
My List
KIDS
DVD
Reality TV
Baby Ballroom
NEW EPISODES
THE GREAT BRITISH BAKING SHOW
SKIN WARS
House Doctor
THE GREAT interior design CHALLENGE
GRAND DESIGNS
Popular on Netflix
MARY AND THE WITCH'S FLOWER
STAR WARS THE LAST JEDI
COCO
THOR RAGNAROK
THE EPIC TALES OF CAPTAIN UNDERPANTS
FATHER OF THE YEAR
New Releases
ARRESTED DEVELOPMENT
GUARDIANS OF THE GALAXY VOL. 2
A SERIES OF UNFORTUNATE EVENTS
UNBREAKABLE KIMMY SCHMIDT
13 REASONS WHY
DESPICABLE ME 3

don’t predict ratings



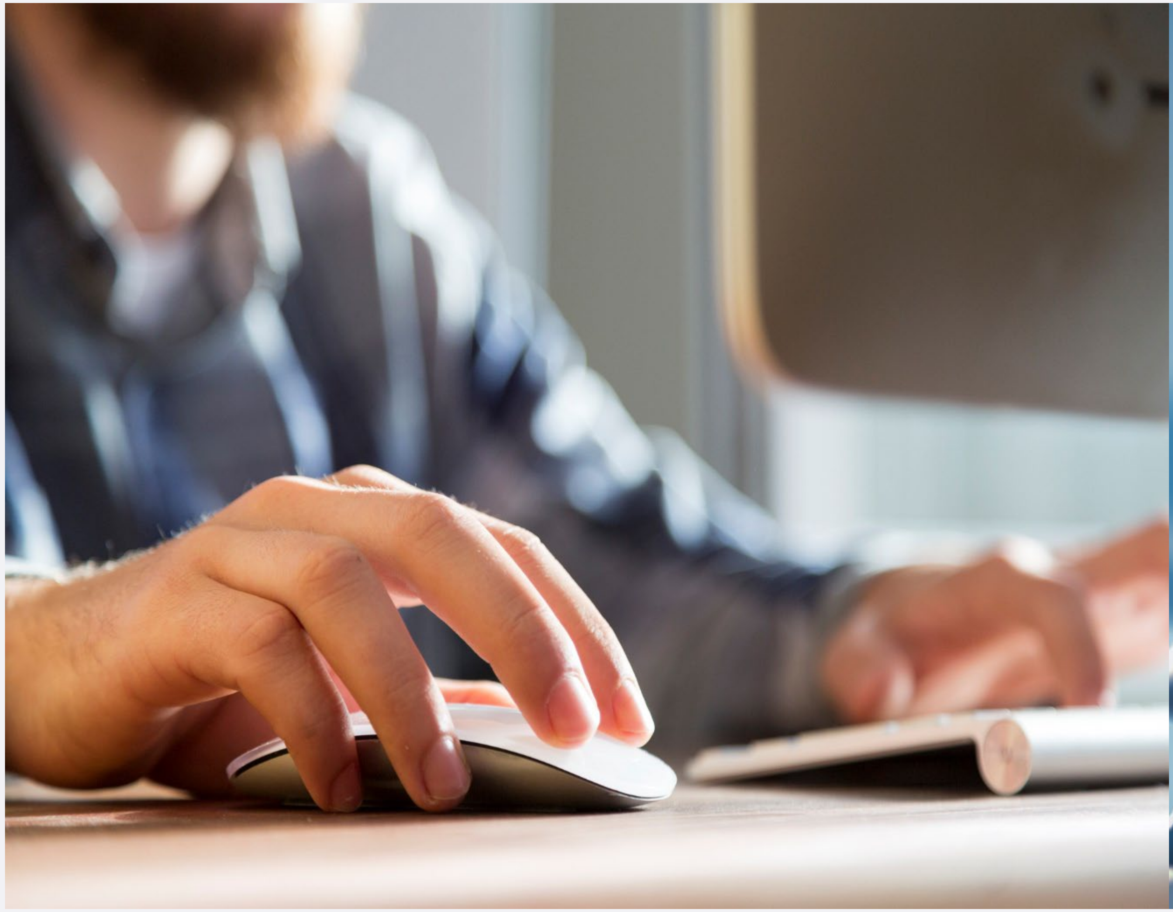
personalized ranking

context-aware

hybrid approaches

ensemble approaches

combining behavior and semantic data



Comedy
Thriller
Drama
Action

exercise: build a hybrid recommender

code walkthrough

learning more

current research: acm sigkdd

The ACM Conference Series on
Recommender Systems
HOME
RECSYS 2018
PAST CONFERENCES
HONORS
CONTACT
search...
RECSYS | Vancouver, BC | 2018

12th ACM Conference on Recommender Systems

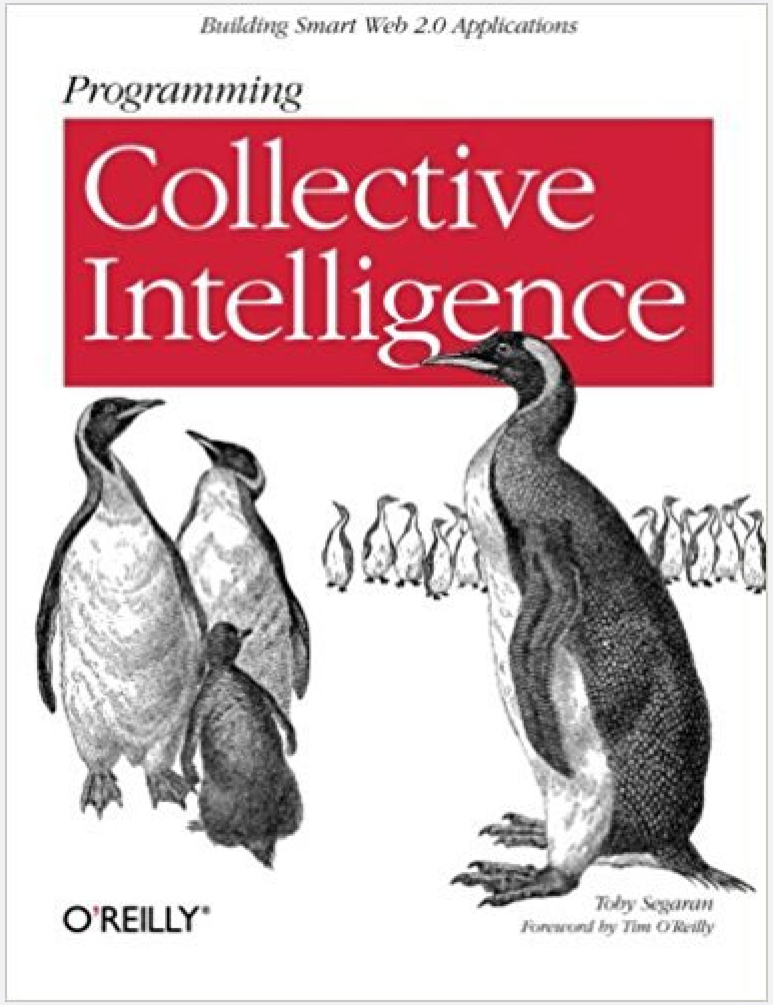
Vancouver, Canada, 2nd-7th October 2018

The ACM Recommender Systems conference (RecSys) is the premier international forum for the presentation of new research results, systems and techniques in the broad field of recommender systems. Recommendation is a particular form of information filtering, that exploits past behaviors and user similarities to generate a list of information items that is personally tailored to an end-user's preferences. As RecSys brings together the main international research groups working on

RECSYS 2018 (VANCOUVER)

- About the Conference
- Call for Contributions
- Registration
- Program

collaborative filtering


Building Smart Web 2.0 Applications

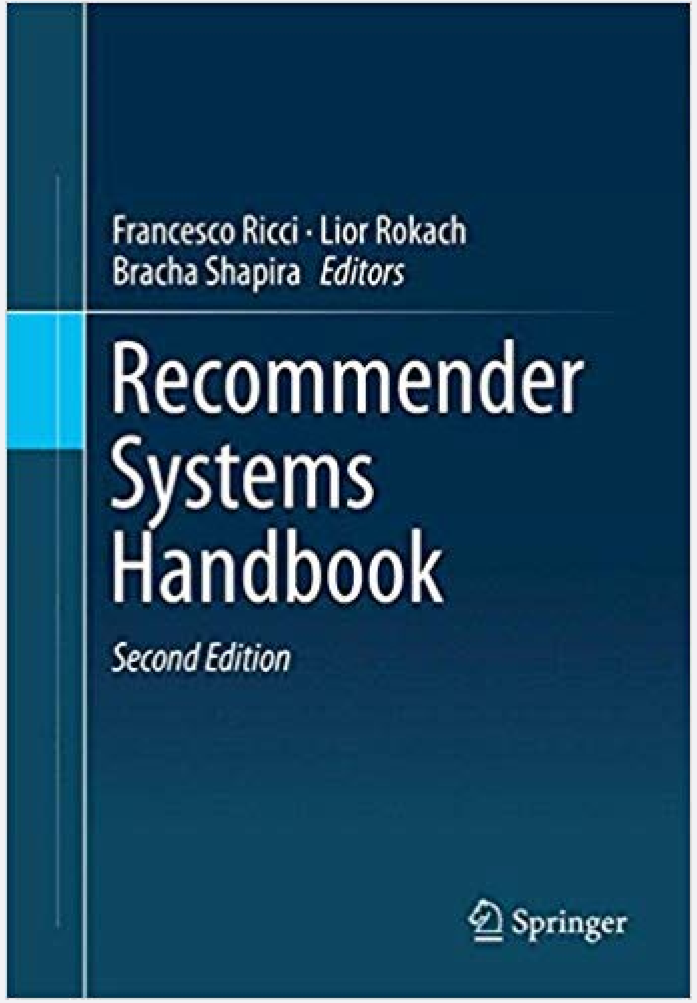
Programming Collective Intelligence

O'REILLY

Toby Segaran

Foreword by Tim O'Reilly

going all-in


Francesco Ricci · Lior Rokach
Bracha Shapira *Editors*

Recommender Systems Handbook

Second Edition

Springer