Microsoft

“Taking recommendation to the masses” with Microsoft/Recommenders

Le Zhang Data Scientist, Microsoft

Acknowledgement:
Andreas Argyriou, Dan Ciborowski, Markus Cosowicz, Miguel Gonzalez-Fierro, Scott Graham, Nikhil Joglekar, Max Kaznady, Jianxun Lian, Micro Milletari, Jun Ki Min, Jeremy Reynolds, Xing Xie, and Tao Wu

Objective

- “Taking recommendation technology to the masses”
  - Helping researchers and developers to quickly select, prototype, demonstrate, and productionize a recommender system
  - Accelerating enterprise-grade development and deployment of a recommender system into production
- Key takeaways of the talk
  - Systematic overview of the recommendation technology from a pragmatic perspective
  - Best practices (with example codes) in developing recommender systems
  - State-of-the-art academic research in recommendation algorithms

Outline

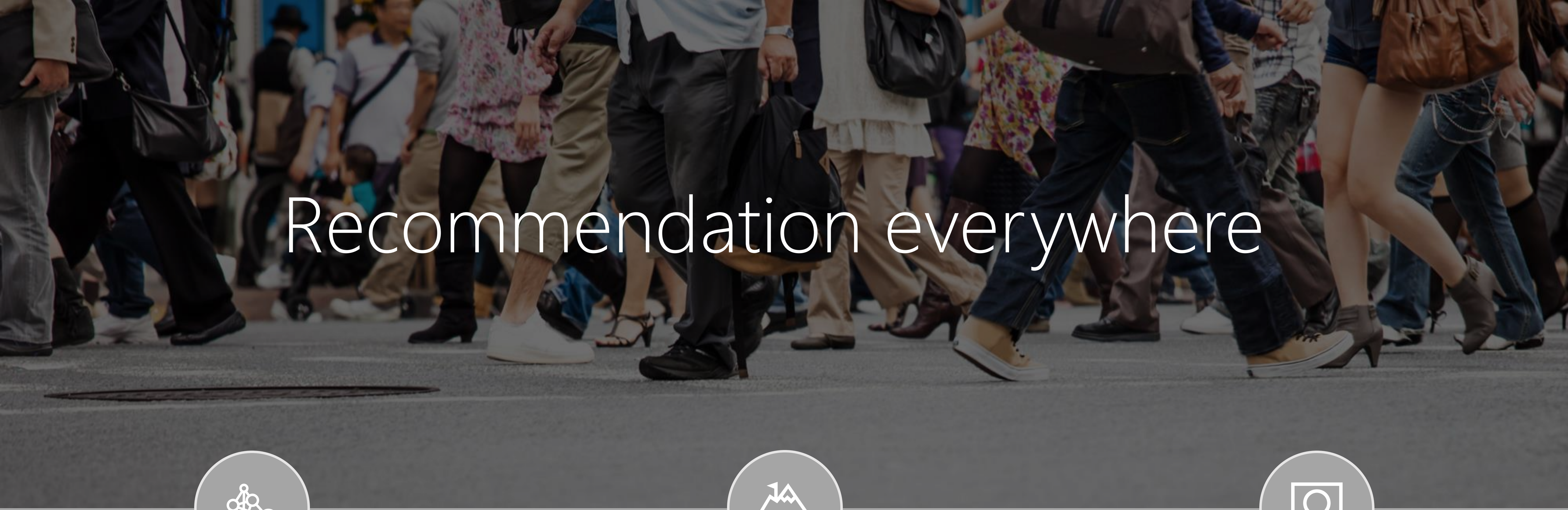
- Recommendation system in modern business (10min)
- Recommendation algorithms and implementations (20min)
- End to end example of building a scalable recommender (10min)
- Q & A (5min)

Recommendation system in modern business

“35% of what consumers purchase on Amazon and 75% of what they watch on Netflix come from recommendations algorithms”

McKinsey & Co

Recommendation everywhere



Recommendation everywhere

Brand/news/product recommendation

Indirectly drive revenue by increasing customer engagement, networking effect, etc.

Business metric prediction

Directly drive revenue through ad clicks, internet traffics, etc.

Customer segmentation and personalization

Indirectly drive revenue by precisely reaching customers with market campaign or product.

Challenges

Limited resource	Fragmented solutions	Fast-growing area
There is *limited* reference and guidance to build a recommender system on scale to support enterprise-grade scenarios	Packages/tools/modules off-the-shelf are very fragmented, not scalable, and not well compatible with each other	New algorithms sprout every day – not many people have such expertise to implement and deploy a recommender by using the state-of-the-arts algorithms

Microsoft/Recommenders

- Microsoft/Recommenders
  - Collaborative development efforts of Microsoft Cloud & AI data scientists, Microsoft Research researchers, academia researchers, etc.
  - Github url: https://github.com/Microsoft/Recommenders
  - Contents
    - Utilities: modular functions for model creation, data manipulation, evaluation, etc.
      - Algorithms: SVD, SAR, ALS, NCF, Wide&Deep, xDeepFM, DKN, etc.
    - Notebooks: HOW-TO examples for end to end recommender building.
  - Highlights
    - 3700+ stars on GitHub
    - Featured in YC Hacker News, O'Reily Data Newsletter, GitHub weekly trending list, etc.
  - Any contribution to the repo will be highly appreciated!
    - Create issue/PR directly in the GitHub repo
    - Send email to RecoDevTeam@service.microsoft.com for any collaboration

Recommendation algorithms and implementations

“Share our similarities, celebrate our differences”

M. Scott Peck

Recommendation models

- Various recommendation scenarios
    - Collaborative filtering, context-aware models, knowledge-aware model,…
- Integrating both Microsoft invented/contributed and excellent third-party tools
    - SAR, xDeepFM, DKN, Vowpal Wabbit (VW), LightGBM,…
    - Wide&Deep, ALS, NCF, FastAI, Surprise, …

- No best model, but most suitable model

Collaborative Filtering

- User feedback from multiple users *in a collaborative way* to predict missing feedback
- Intuition: users who give similar ratings to the same items will have similar preferences → should produce similar recommendations to them
- E.g. users A and B like western movies but hate action films, users C and D like comedies but hate dramas

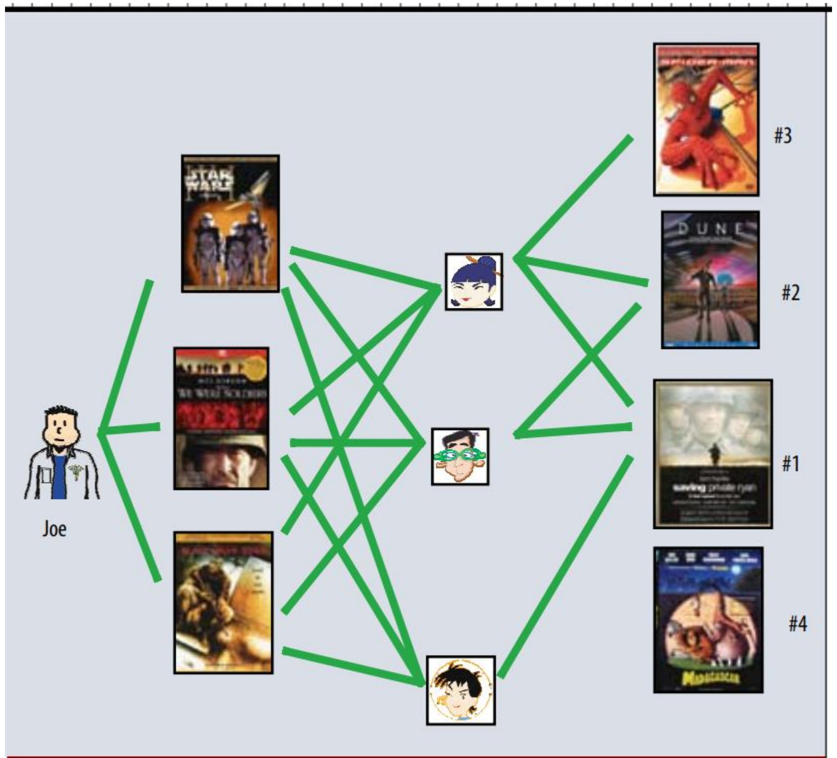


Figure 1. The user-oriented neighborhood method. Joe likes the three movies on the left. To make a prediction for him, the system finds similar users who also liked those movies, and then determines which other movies they liked. In this case, all three liked *Saving Private Ryan*, so that is the first recommendation. Two of them liked *Dune*, so that is next, and so on.

Y Koren et al, Matrix factorization techniques for recommendation systems, IEEE Computer 2009

Collaborative filtering (cont'd)

- Memory based method
  - Microsoft Smart Adaptive Recommendation (SAR) algorithm
- Model based methods
  - Matrix factorization methods
    - Singular Value Decomposition (SVD)
    - Spark ALS implementation
  - Neural network-based methods
    - Restricted Boltzmann Machine (RBM)
    - Neural Collaborative Filtering (NCF)

Collaborative Filtering

- Neighborhood-based methods - Memory-based
  - The neighborhood-based algorithm calculates the similarity between two users or items and produces a prediction for the user by taking the weighted average of all the ratings.
  - Two typical similarity measures:

Pearson correlation similarity:

$$s(x,y) = \frac{\sum_{\{i \in I_{xy}\}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt[2]{\sum_{\{i \in I_{xy}\}} (r_{x,i} - \bar{r}_x)^2} \sqrt[2]{\sum_{\{i \in I_{xy}\}} (r_{y,i} - \bar{r}_y)^2}}$$

Cosine similarity:

$$s(x,y) = \frac{\sum_{\{i \in I_{xy}\}} r_{x,i}\, r_{y,i}}{\sqrt[2]{\sum_{\{i \in I_{xy}\}} r_{x,i}^2} \sqrt{\sum_{\{i \in I_{xy}\}} r_{y,i}^2}}$$

  - Two paradigms:

UserCF:

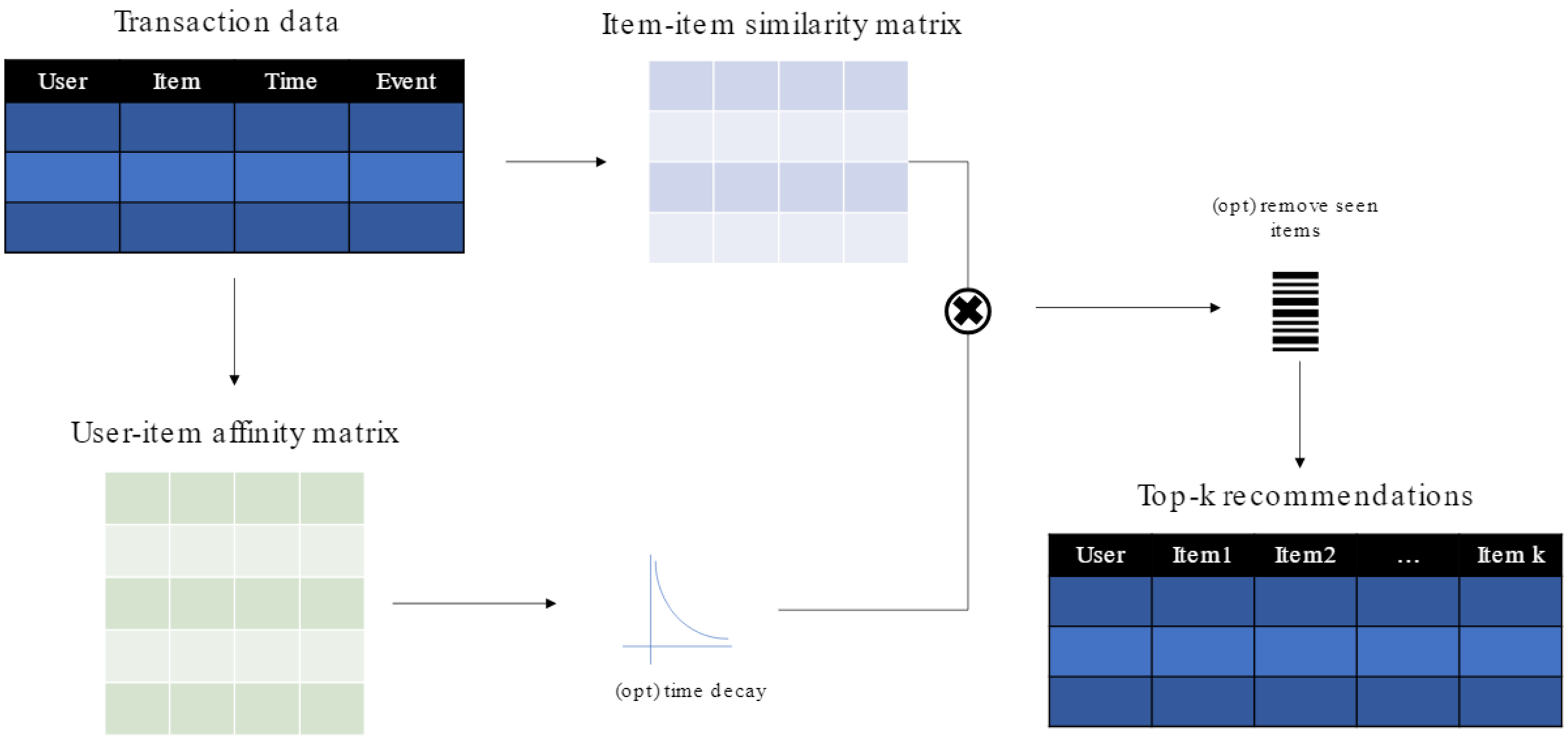
$$\hat{y}_{ui} = \sum_{v \in S(u,K) \cap I(i)} s(u,v) y_{vi}$$

ItemCF:

$$\hat{y}_{ui} = \sum_{j \in S(i,K) \cap I(u)} s(j,i) y_{uj}$$

Smart Adaptive Recommendation (SAR)

- An item-oriented memory-based algorithm from Microsoft



https://github.com/Microsoft/Recommenders/blob/master/notebooks/02_model/sar_deep_dive.ipynb

SAR (cont'd)

- SAR algorithm (the CF part)
  - It deals with implicit feedback
    - Item-to-item similarity matrix
      - Co-occurrence
      - Lift similarity
      - Jaccard similarity
    - User-to-item affinity matrix
      - Count of co-occurrence of user-item interactions
      - Weighted by interaction type and time decay
        - $a_{i,j} = \sum_1^k w_k (\frac{1}{2})^{\frac{t_0 - t_k}{T}}$
    - Recommendation
      - Product of affinity matrix and item similarity matrix
      - Rank of product matrix gives top-n recommendations

User ID	Item ID	Time	Event
User 1	Item 1	2015/06/20T10:00:00	Click
User 1	Item 1	2015/06/28T11:00:00	Click
User 1	Item 2	2015/08/28T11:01:00	Click
User 1	Item 2	2015/08/28T12:00:01	Purchase

Original feedback data

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	5.00	3.00	2.50		
User 2	2.00	2.50	5.00	2.00	
User 3	2.50			4.00	4.50
User 4	5.00		3.00	4.50	
User 5	4.00	3.00	2.00	4.00	3.50
User 6					2.00
User 7		1.00			

User affinity matrix

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

Item similarity matrix

User 1 recommendation score of item 4

rec(User 1, Item 4)
= sim(Item 4, Item 1) * aff(User 1, Item 1)
*+ sim(Item 4, Item 2) * aff(User 1, Item 2)*
*+ sim(Item 4, Item 3) * aff(User 1, Item 3)*
*+ sim(Item 4, Item 4) * aff(User 1, Item 4)*
*+ sim(Item 4, Item 5) * aff(User 1, Item 5)*
*= **3 * 5** + 2 * 3 + 3 * 2.5 + 4 * 0 + 2 * 0*
*= **15** + 6 + 7.5 + 0 + 0 = **28.5***

https://github.com/Microsoft/Product-Recommendations/blob/master/doc/sar.md
https://github.com/Microsoft/Recommenders/blob/master/notebooks/02_model/sar_deep_dive.ipynb

SAR Properties

- Advantages
  - Free from machine learning
  - Free from feature collection
  - Explainable results

- Disadvantages
  - Sparsity of affinity matrix
    - User-item interaction is usually sparse
  - Scalability of matrix multiplication
    - User-item matrix size grows with number of users and items
    - Matrix multiplication can be a challenge

SAR practice with Microsoft/Recommenders

- Import packages

In [1]:
```
# set the environment path to find Recommenders
import sys
sys.path.append("../../")

import itertools
import logging
import os

import numpy as np
import pandas as pd
import papermill as pm

from reco_utils.dataset import movielens
from reco_utils.dataset.python_splitters import python_stratified_split
from reco_utils.evaluation.python_evaluation import map_at_k, ndcg_at_k, precision_at_k, recall_at_k
from reco_utils.recommender.sar.sar_singlenode import SARSingleNode

print("System version: {}".format(sys.version))
print("Pandas version: {}".format(pd.__version__))
```

```
System version: 3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 01:22:34)
[GCC 7.3.0]
Pandas version: 0.24.2
```

Source code: https://github.com/microsoft/recommenders/blob/master/notebooks/02_model/sar_deep_dive.ipynb

SAR practice with Microsoft/Recommenders

- Prepare dataset

In [3]:
```
data = movielens.load_pandas_df(
    size=MOVIELENS_DATA_SIZE,
    header=['UserId', 'MovieId', 'Rating', 'Timestamp'],
    title_col='Title'
)

# Convert the float precision to 32-bit in order to reduce memory consumption
data.loc[:, 'Rating'] = data['Rating'].astype(np.float32)

data.head()
```

```
4.93MB [00:02, 2.36MB/s]
```

Out[3]:

	UserId	MovieId	Rating	Timestamp	Title
0	196	242	3.0	881250949	Kolya (1996)
1	63	242	3.0	875747190	Kolya (1996)
2	226	242	5.0	883888671	Kolya (1996)
3	154	242	3.0	879138235	Kolya (1996)
4	306	242	5.0	876503793	Kolya (1996)

In [5]:
```
train, test = python_stratified_split(data, ratio=0.75, col_user=header["col_user"], col_item=header["col_
item"], seed=42)
```

Source code: https://github.com/microsoft/recommenders/blob/master/notebooks/02_model/sar_deep_dive.ipynb

SAR practice with Microsoft/Recommenders

- Fit a SAR model

In [6]:
```
# set log level to INFO
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s %(levelname)-8s %(message)s')

model = SARSingleNode(
    similarity_type="jaccard",
    time_decay_coefficient=30,
    time_now=None,
    timedecay_formula=True,
    **header
)
```

In [7]:
```
model.fit(train)
```

```
2019-05-28 22:40:09,133 INFO     Collecting user affinity matrix
2019-05-28 22:40:09,137 INFO     Calculating time-decayed affinities
2019-05-28 22:40:09,178 INFO     Creating index columns
2019-05-28 22:40:09,188 INFO     Building user affinity sparse matrix
2019-05-28 22:40:09,194 INFO     Calculating item co-occurrence
2019-05-28 22:40:09,412 INFO     Calculating item similarity
2019-05-28 22:40:09,413 INFO     Using jaccard based similarity
2019-05-28 22:40:09,534 INFO     Done training
```

Source code: https://github.com/microsoft/recommenders/blob/master/notebooks/02_model/sar_deep_dive.ipynb

SAR practice with Microsoft/Recommenders

- Get the top k recommendations

```
In [8]: top_k = model.recommend_k_items(test, remove_seen=True)
```

```
In [10]: # all ranking metrics have the same arguments
args = [test, top_k]
kwargs = dict(col_user='UserId',
              col_item='MovieId',
              col_rating='Rating',
              col_prediction='Prediction',
              relevancy_method='top_k',
              k=TOP_K)

eval_map = map_at_k(*args, **kwargs)
eval_ndcg = ndcg_at_k(*args, **kwargs)
eval_precision = precision_at_k(*args, **kwargs)
eval_recall = recall_at_k(*args, **kwargs)
```

```
In [11]: print(f"Model:",
      f"Top K:\t\t {TOP_K}",
      f"MAP:\t\t {eval_map:f}",
      f"NDCG:\t\t {eval_ndcg:f}",
      f"Precision@K:\t {eval_precision:f}",
      f"Recall@K:\t {eval_recall:f}", sep='\n')
```

```
Model:
Top K:          10
MAP:            0.095544
NDCG:           0.350232
Precision@K:    0.305726
Recall@K:       0.164690
```

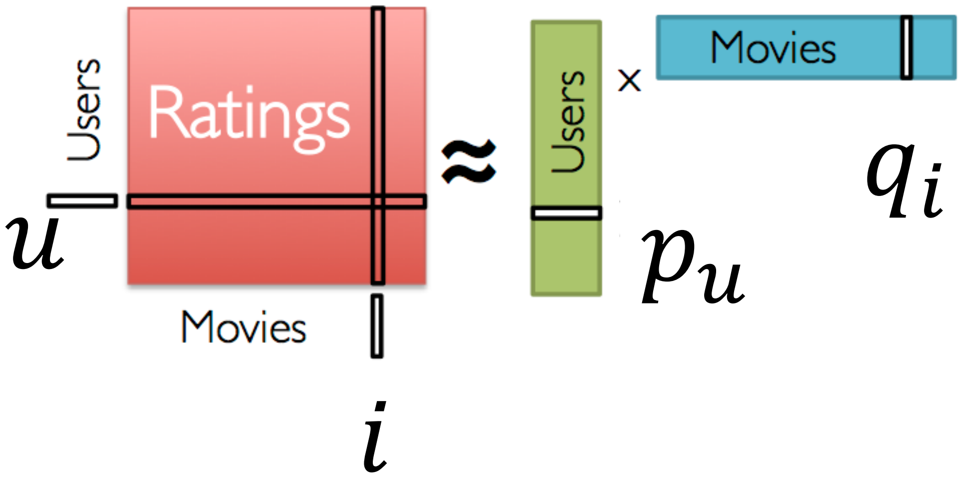
Source code: https://github.com/microsoft/recommenders/blob/master/notebooks/02_model/sar_deep_dive.ipynb

Matrix factorization

- The simplest way to model latent factors is as *user & item vectors* that multiply (as inner products)
- Learn these factors from the data and use as model, and predict an unseen rating of user-item by multiplying user factor with item factor
  - The matrix factors *U, V* have *f* columns, rows resp.
  - The number of factors *f* is also called the *rank* of the model

Stochastic Gradient Descent (SGD)
Parameters are updated in the opposite direction of gradient:

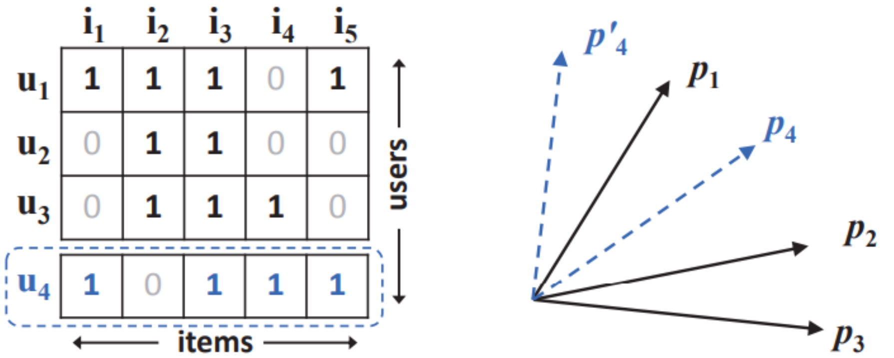
$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$



https://www.datacamp.com/community/tutorials/matrix-factorization-names

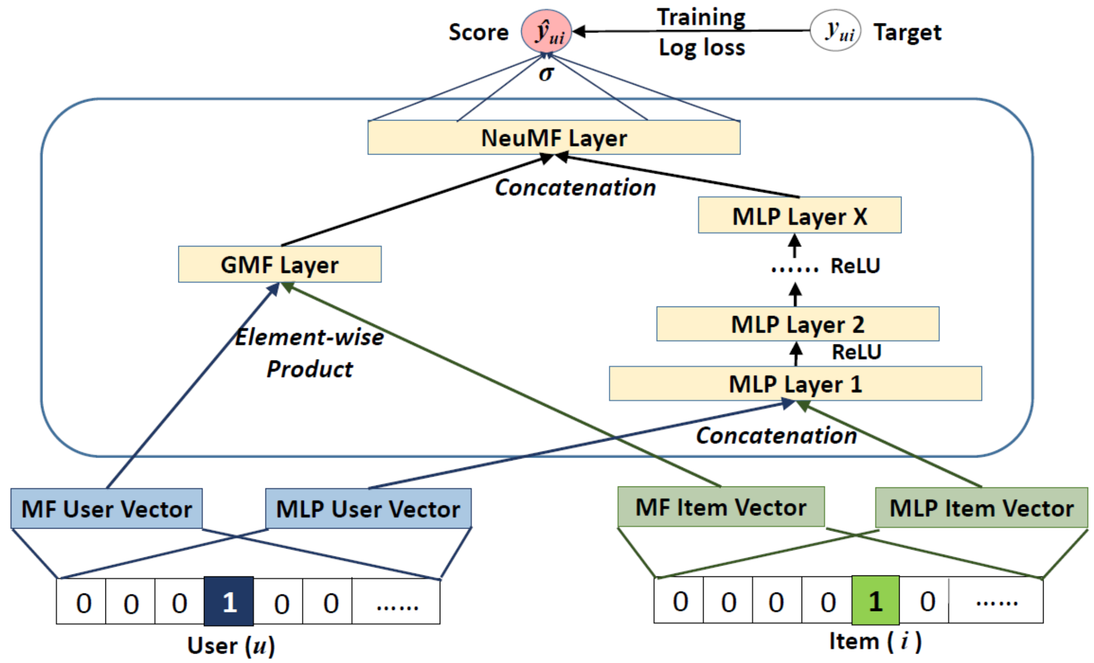
Neural collaborative filtering (NCF)

- Neural collaborative filtering
  - Neural network-based architecture to model latent features
  - Generalization of MF based method
  - Multi-Layer Perceptron (MLP) can be incorporated for dealing with non-linearities



(a) user–item matrix (b) user latent space

Figure 1: An example illustrates MF's limitation. From data matrix (a), u_4 is most similar to u_1, followed by u_3, and lastly u_2. However in the latent space (b), placing p_4 closest to p_1 makes p_4 closer to p_2 than p_3, incurring a large ranking loss.



X He et al, Neural collaborative filtering, WWW 2017

Content-based filtering

- Content-based filtering methods
  - “Content” can be user/item features, review comments, knowledge graph, multi-domain information, contextual information, etc.
  - Mitigate the cold-start issues in collaborative filtering typed algorithms
  - Personalized recommendation
    - Location, device, age, etc.

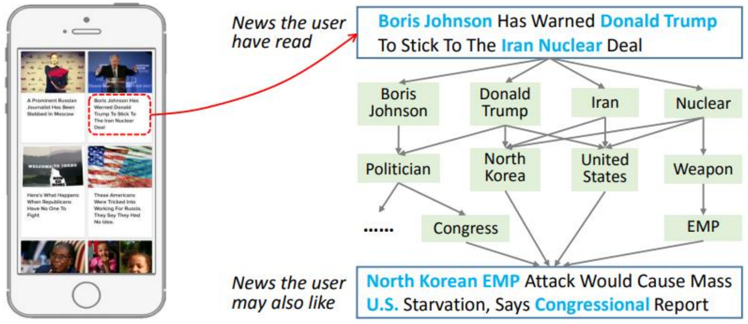


Figure 1: Illustration of two pieces of news connected through knowledge entities.

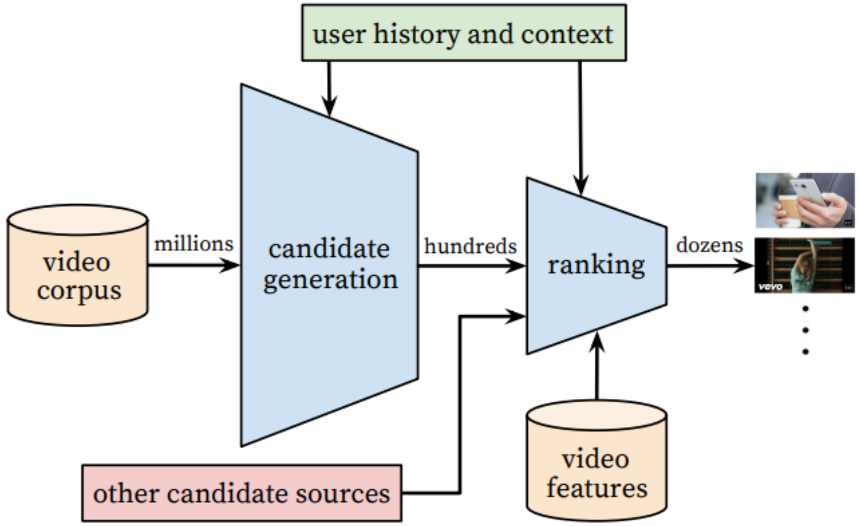


Figure 2: Recommendation system architecture demonstrating the “funnel” where candidate videos are retrieved and ranked before presenting only a few to the user.

H Wang et al, Deep knowledge aware network for news recommendation, WWW’18
Paul Convington, et al, Deep Neural Networks for YouTube Recommendations. RecSys’16

Content-based algorithms

- A content-based machine learning perspective
  - $\hat{y}(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x})$
  - Logistic regression, factorization machine, GBDT, …
- Feature vector is highly sparse
  - $\boldsymbol{x} = [0,0, \dots, 1,0,0, \dots, 1, \dots 0,0, \dots] \in R^D$, where D is a large number
- The interaction between features
  - Cross-product transformation of raw features
    - In matrix factorization: $<user_i,\ item_j>$
    - A 3-way cross feature: AND(*gender=f, time=Sunday, category=makeup*)

Factorization Machines (FM)

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i \, x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i \, x_j$$

Feature vector **x**

	User				Movie					Other Movies rated					Time	Last Movie rated					Target y	
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...		TI	NH	SW	ST	...		
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

Rendle, Steffen. "Factorization machines." ICDM 2010

Factorization machine (FM)

- Advantages of FM
  - Parameter estimation of sparse data – independence of interaction parameters are broken because of factorization
  - Linear complexity of computation, i.e., O(kn)
  - General predictor that works for any kind of feature vectors
- Formulation

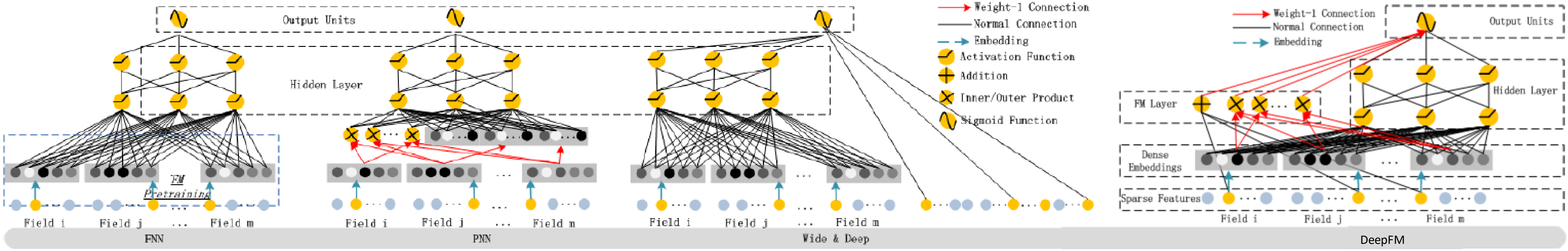
$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i \, x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i \, x_j$$

  - The weights w0, wi, and the dot product of vectors are the estimated parameters
  - It can be learnt by using SGD with a variety of loss functions, as it has closed-form equation can be computed in linear time complexity

S Rendle, Factorization Machines, ICDM 2010

Extending FM to Higher-order Feature Interactions

- Leveraging the power of deep neural networks



Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems.“ DLRS 2016.
Guo, Huifeng, et al. "DeepFM: A factorization-machine based neural network for CTR prediction." IJCAI 2017

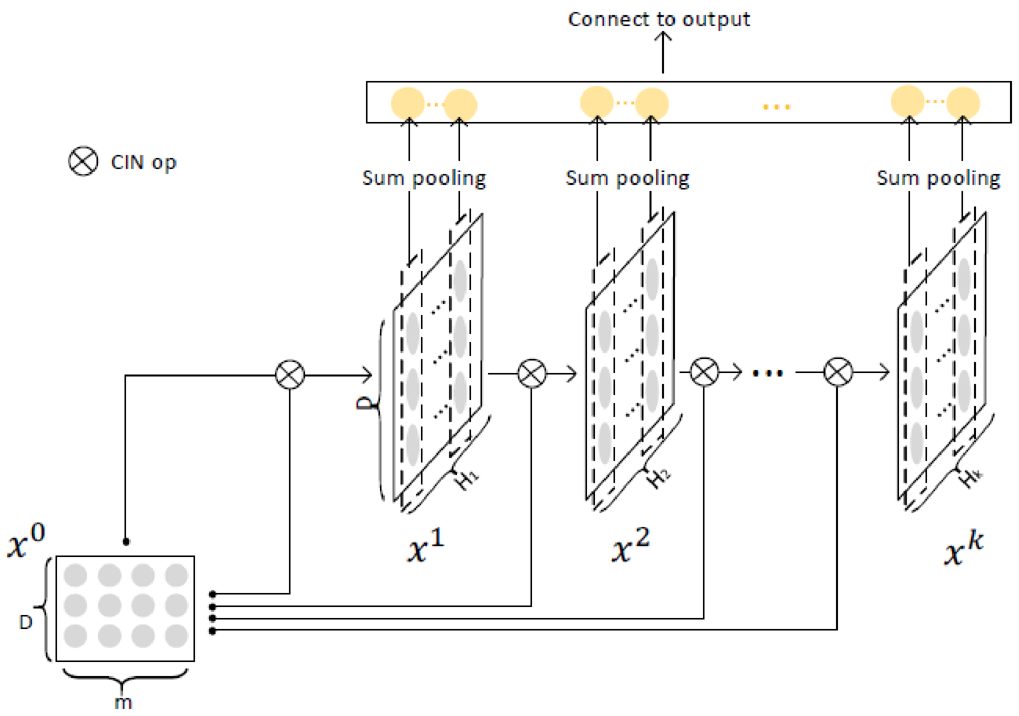
Extreme deep factorization machine (xDeepFM)

➢Compressed Interaction Network (CIN)

- Hidden units at the k-th layer:

$$\mathbf{X}_{h,*}^{k} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} \mathbf{W}_{ij}^{k,h} (\mathbf{X}_{i,*}^{k-1} \circ \mathbf{X}_{j,*}^{0})$$

m: # fields in raw data
D: dimension of latent space
H_k: # feature maps in the k-th hidden layer
x^0 : input data
x^k: states of the k-th hidden layer



➢Properties

- Compression: reduce interaction space from $O(mH_{k-1})$ down to $O(H_k)$
- Keep the form of vectors
  - Hidden layers are matrices, rather than vectors
- Degree of feature interactions increases with the depth of layers (explicit)

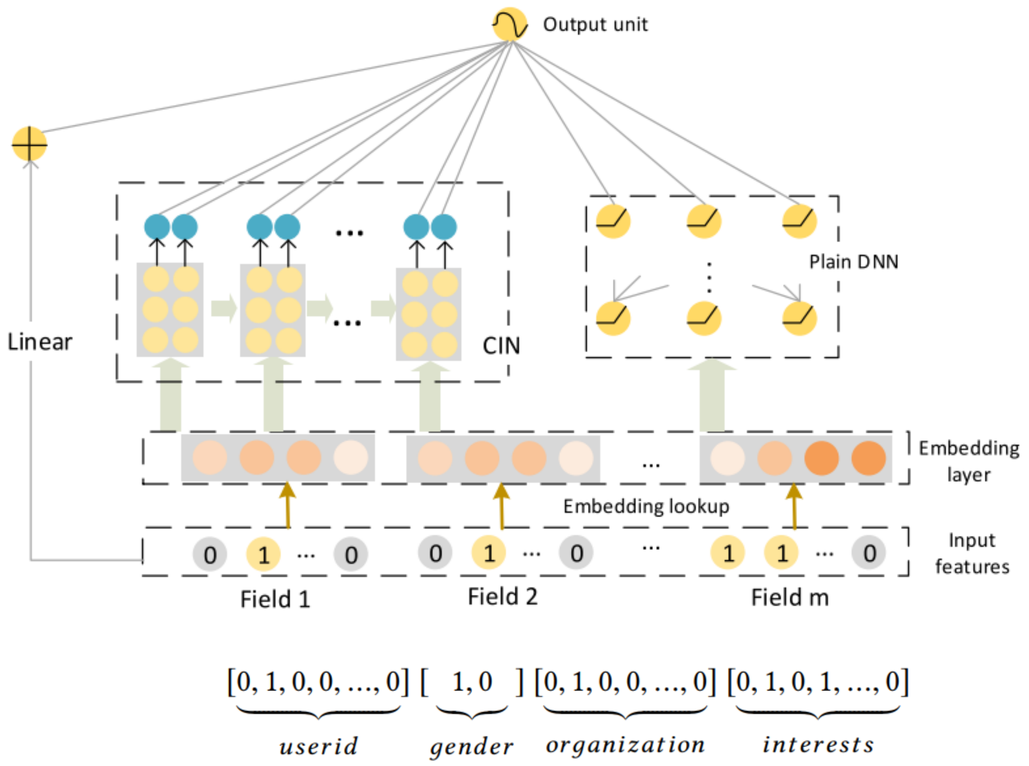
Jianxun Lian et al, Combining explicit and implicit feature interactions for recommender systems, KDD 2018

Extreme deep factorization machine (xDeepFM)

- Proposed for CTR prediction

$$\hat{y} = \sigma(\mathbf{w}_{linear}^T \mathbf{a} + \mathbf{w}_{dnn}^T \mathbf{x}_{dnn}^k + \mathbf{w}_{cin}^T \mathbf{p}^+ + b)$$

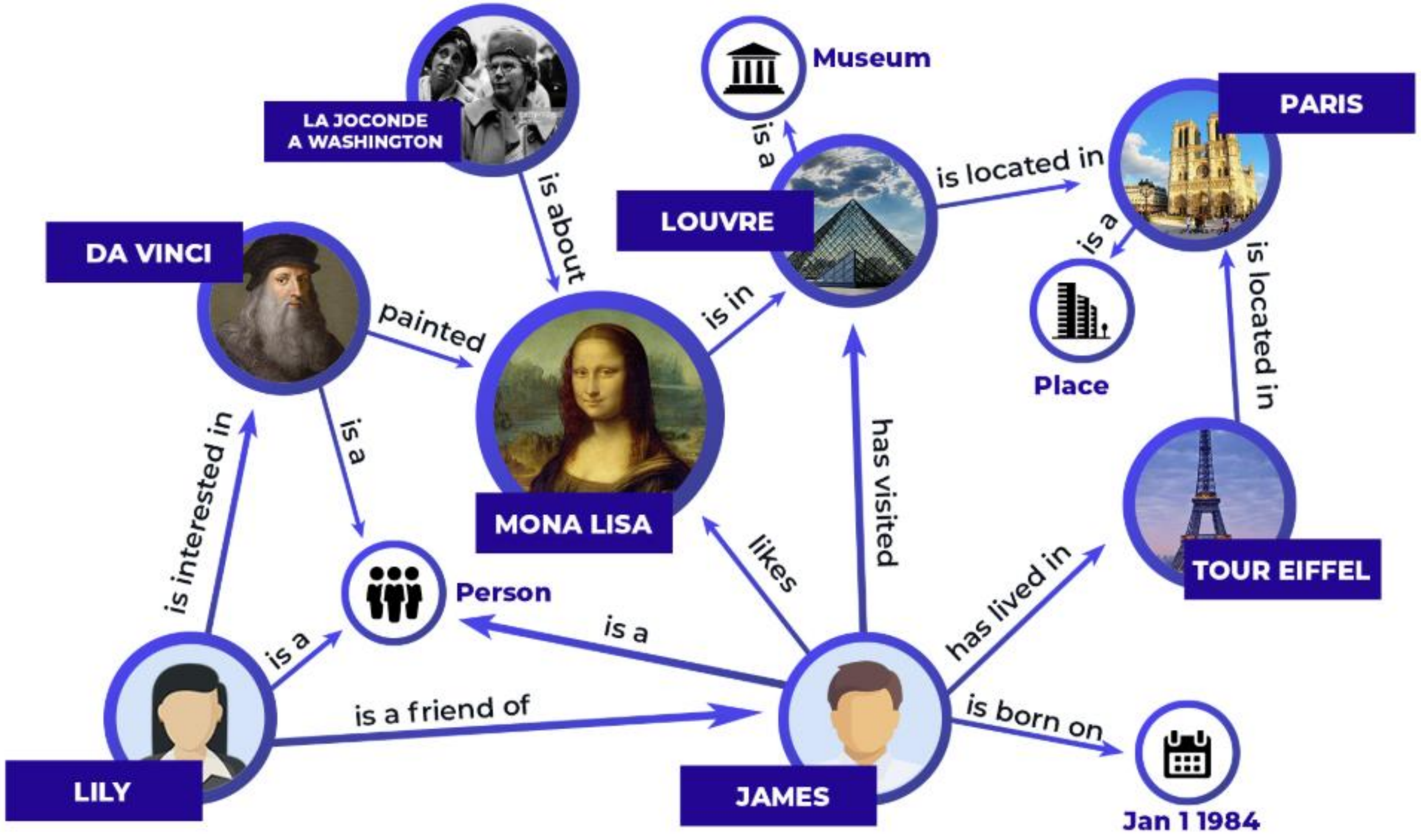
  - Low-order and high-order feature interactions:
    - Linear: linear and quadratic interactions (low order)
    - DNN higher order implicit interactions (black-box, no theoretical understanding, noise effects)
    - Compressed Interaction Network (CIN)
      - Compresses embeddings
      - High-order explicit interactions
      - Vector-wise instead of bit-wise



Jianxun Lian et al, Combining explicit and implicit feature interactions for recommender systems, KDD 2018

Recommender Systems Meet Knowledge Graph

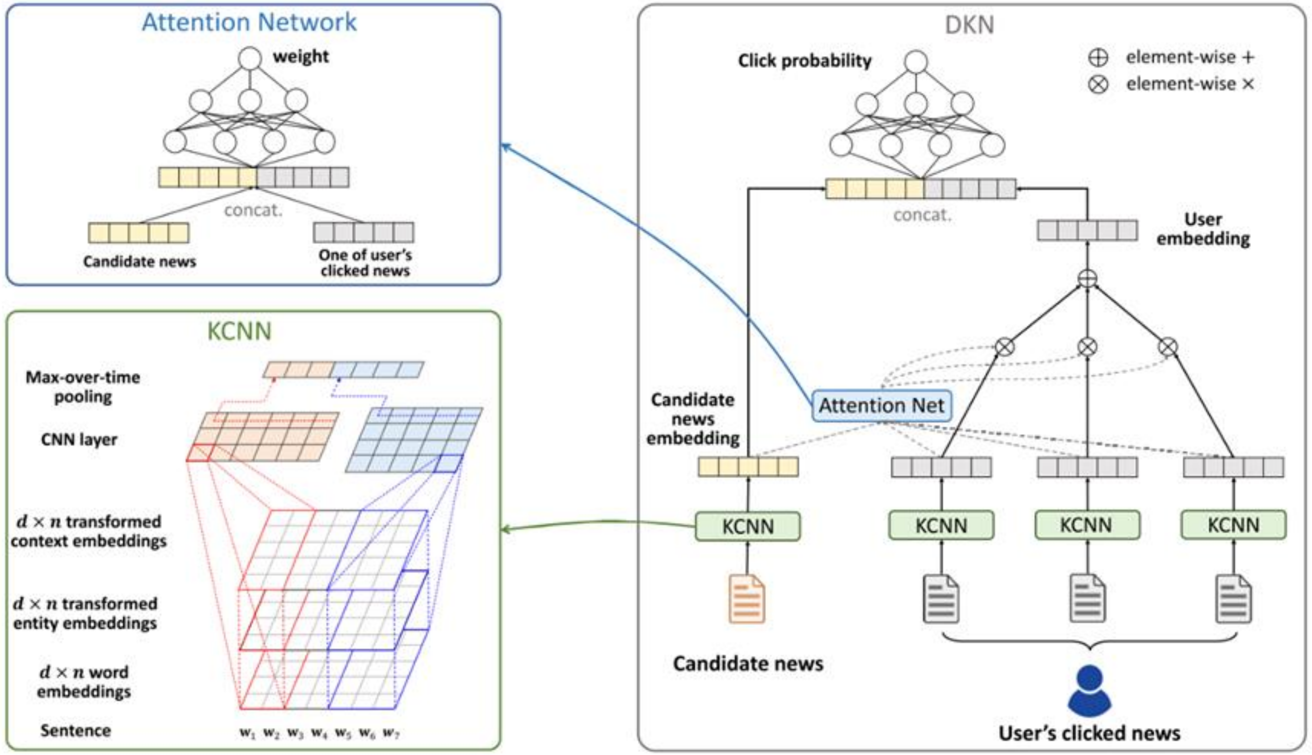
- Items are not isolated
- KG meets RSs:
  - more accurate predictions
  - generate more diverse candidates
  - Provide high-quality explanations



https://kpi6.com/blog/interest-detection-from-social-media/knowledge-graph/

Deep knowledge-aware network

- Features of DKN
  - Multi-channel word-entity aligned knowledge aware CNN
    - Similar to RGB in images
    - Alignment to eliminate heterogeneity of word, entity, etc.
  - Semantic level and knowledge level
    - Knowledge graph (distillation: entity linking, kg construction, kg embedding)
    - Translation-based embedding methods (TransE, TransH, etc.)
  - Attention mechanism to capture diversity of user preferences



A combination of two parts in the KCNN model – news vectors (from entities and words) and user vectors (clicked news items)

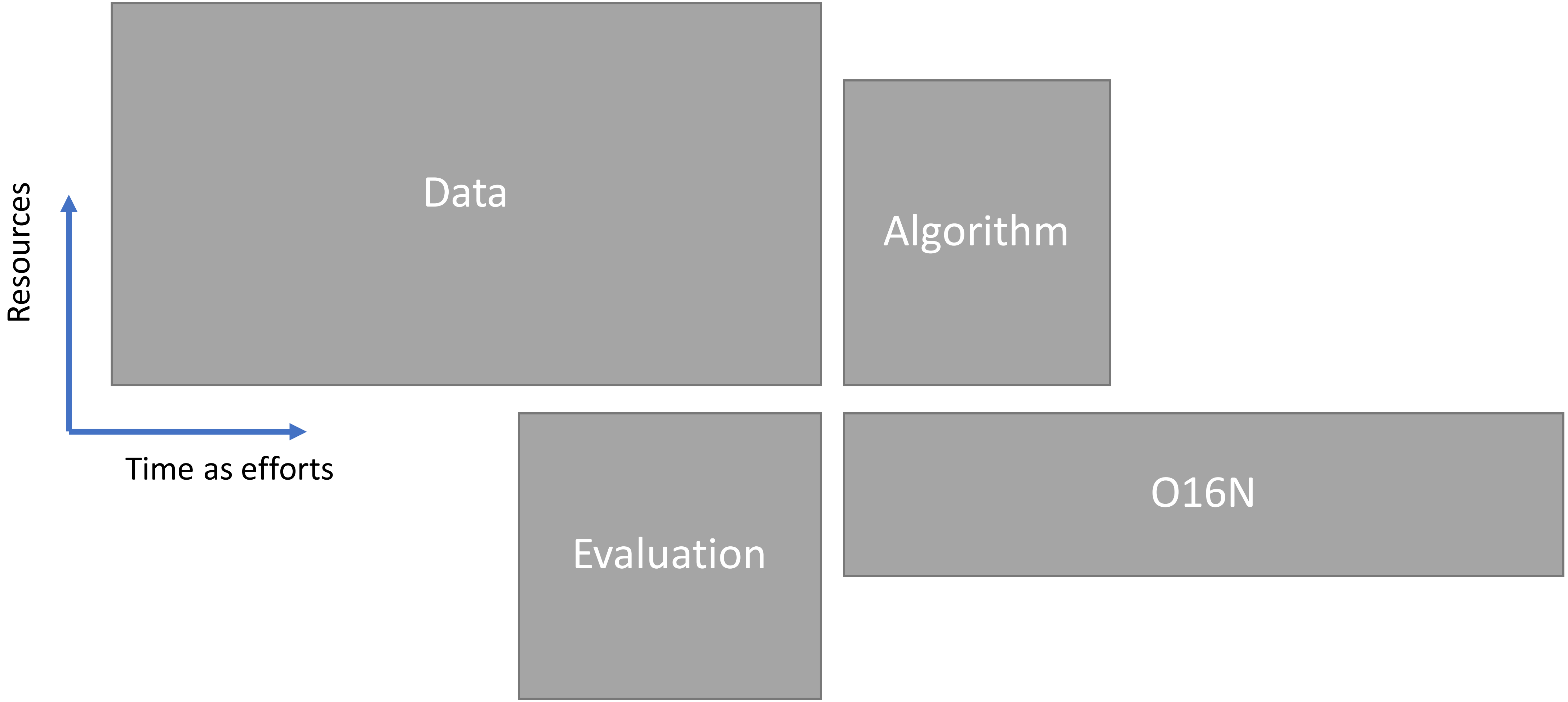
H Wang et al, Deep knowledge aware network for news recommendation, WWW 2018

End-to-end example

“The best way to predict the future is to invent it.”

Alan Kay

Operationalization challenge



https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Operationalization

- End to end operationalization
  - Data collection front end
  - Data preparation pipeline
  - Data storage (i.e., graph database, distributed database, etc.)
  - Model building pipeline
    - Hyperparameter tuning
    - Cross-validation
  - Model deployment
    - Scoring (real-time or in batch) by using the model
    - Frond end web/app service
  - DevOps
    - Model versioning, testing, maintaining, etc.

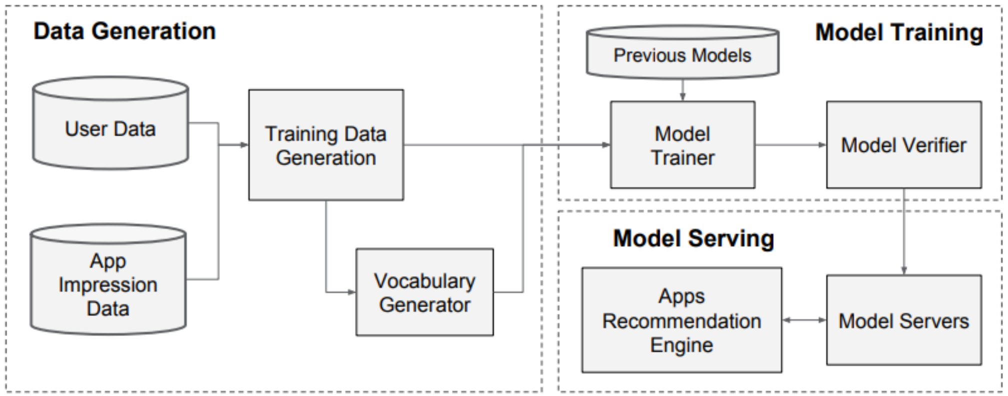
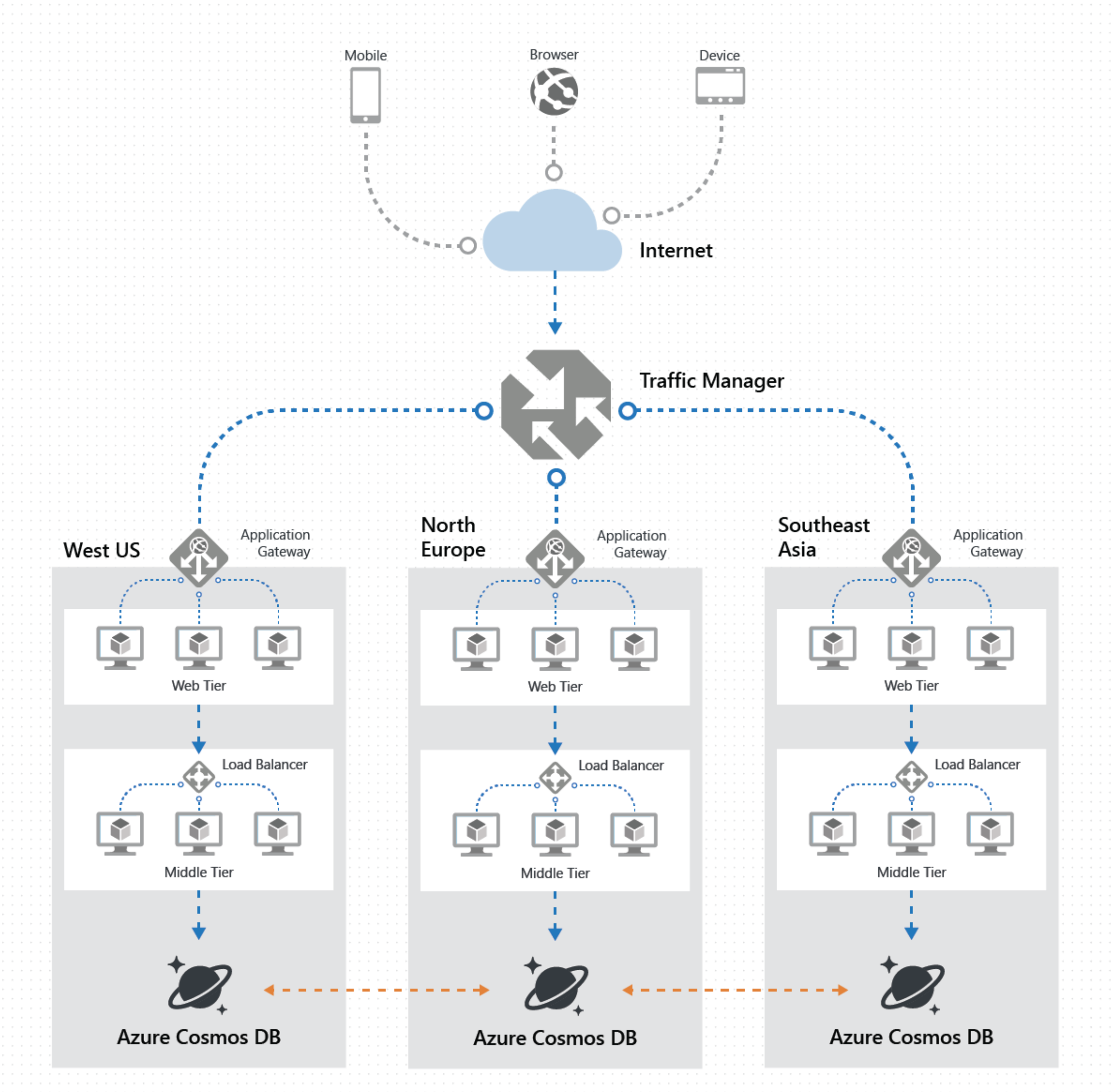


Figure 3: Apps recommendation pipeline overview.

https://arxiv.org/pdf/1606.07792.pdf

Operationalize a real-time recommender

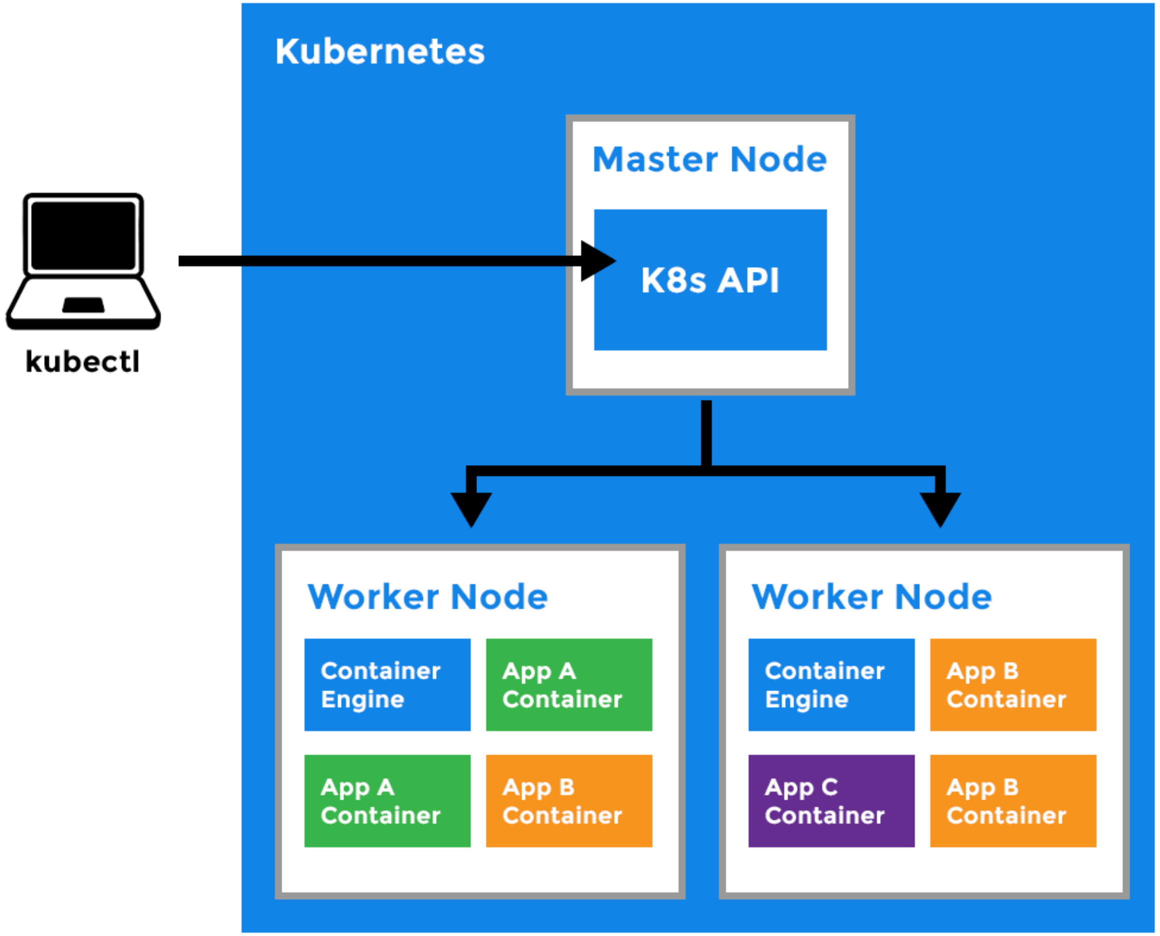
- Caching recommendations
  - Recommendation results are put into database for serving
  - Recommendations from a CF model can be served in a batch mode
  - Globally distributed database with high-throughput support is needed
    - Global active-active apps
    - Highly responsive apps
    - Highly available apps
    - Continuity during regional outrages
    - Scale read and write globally
    - Consistency flexibility



https://docs.microsoft.com/en-us/azure/cosmos-db/distribute-data-globally

Operationalize a real-time recommender

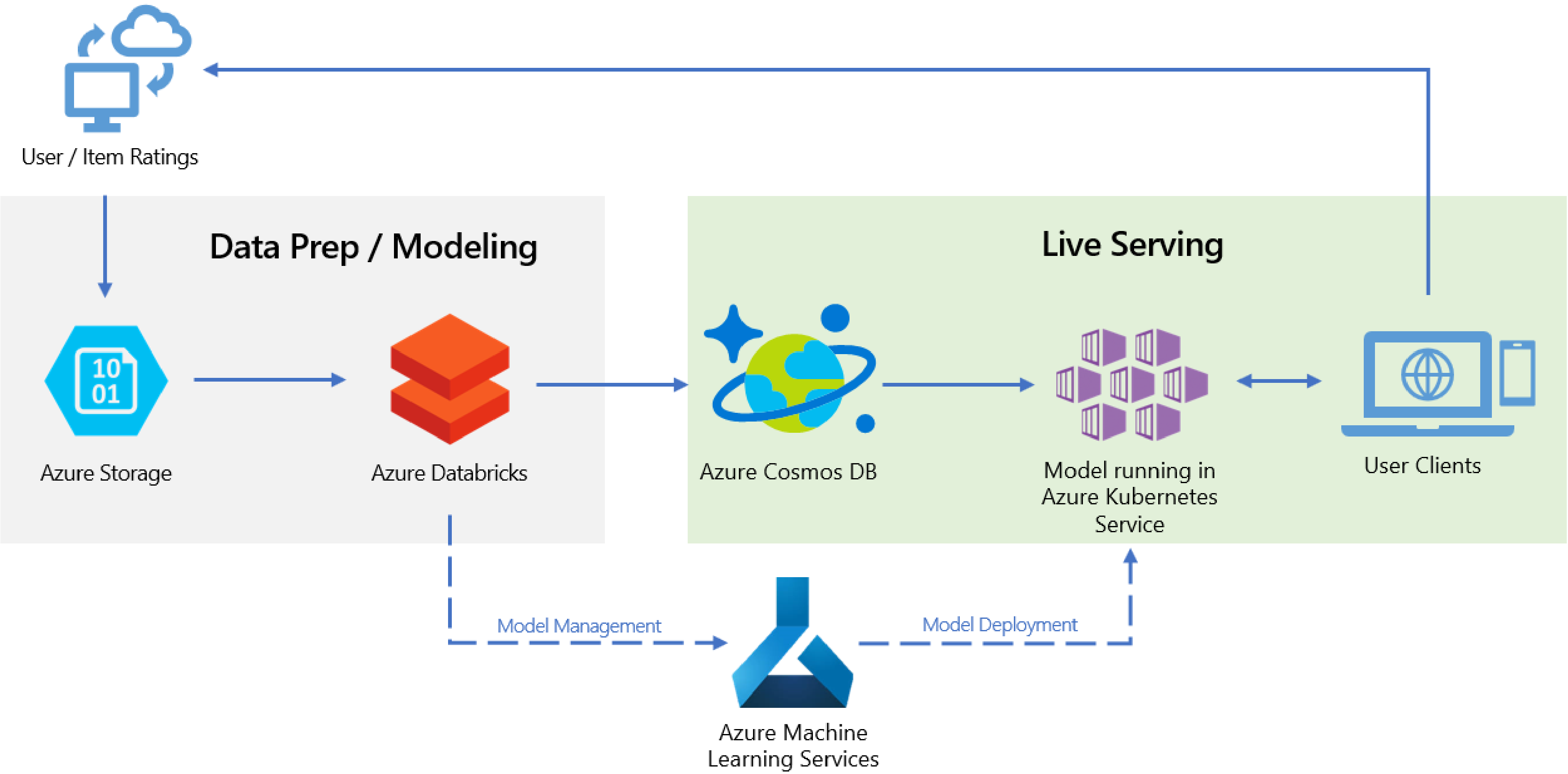
- Serving the results
  - Containerize the model serving pipeline
    - Docker container
    - Modularization
  - Kubernetes is used for scalability benefits
    - K8S manages networking across containers
    - Cluster can be sized properly according to the traffic characteristics



https://docs.microsoft.com/en-us/azure/cosmos-db/distribute-data-globally

Operationalize a real-time recommender

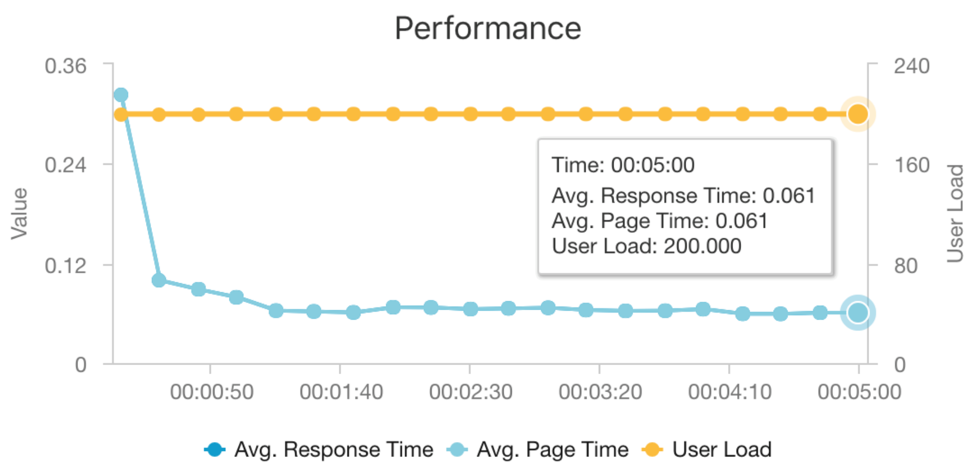
- The whole end-to-end architecture

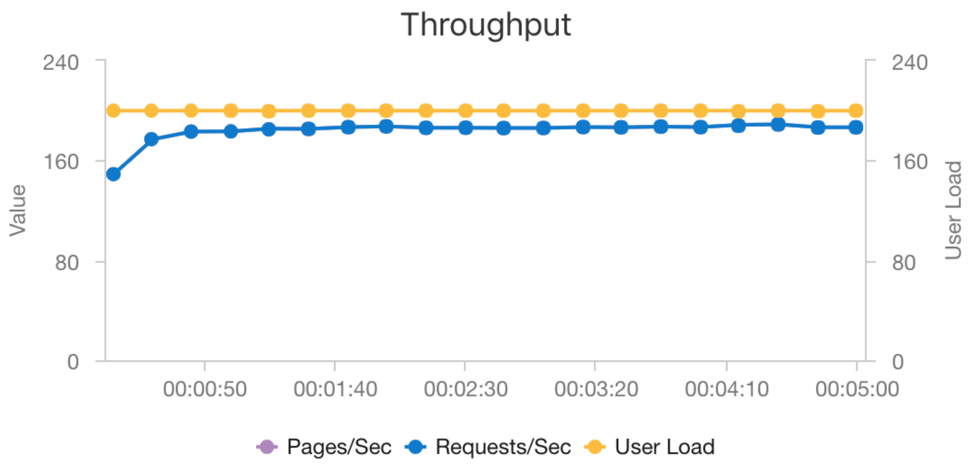


https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/ai/real-time-recommendation

Operationalize a real-time recommender

- Performance measurement
  - A simulated load-test with 200 concurrent users
  - K8S cluster design consideration
    - Optimize throughput of database query
    - Sizing of computing nodes in Kubernetes cluster
    - Example
      - Kubernetes cluster with 12 CPU cores, 42 GB memory, and 11000 “request units” for Azure Cosmos DB
      - Median latency of 60ms at a throughput of 180 requests per second





https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/ai/real-time-recommendation

Summary

- The ultimate goal of a recommender system is to predict user preferences instead of to optimize root mean squared error
- Building a recommender system for industry-grade applications requires in-depth understanding of data preparation, evaluation, recommending algorithm, and model operationalization
- A deployed recommender system should always be up-to-date along with the change of data (characteristics), business scenarios, operationalization pipeline, etc.
- Recommender system is built by using a blend of many technologies, e.g., deep learning, parallel computing, distributed database, etc.

Q & A