Information Retrieval

CS 547/DS 547

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

Quiz2

Learning to Rank
(Machine Learned-Ranking)

Conventional Ranking Models

- Content relevance
  - Boolean, vector space, language model, ...
- Page importance
  - Link analysis: PageRank, HITS, ...
  - Query log mining, clickthroughs, …

Machine learning for IR ranking?

- There's a large body of work in machine learning
- Surely we can also use machine learning to rank the documents displayed in search results?
  - Sounds like a good idea
  - => "machine-learned relevance" or "learning to rank"

What is Machine Learning?

- Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence.
- Machine learning is a "Field of study that gives computers the ability to learn without being explicitly programmed".

Company Info

Follow

Get job updates from eBay Inc.

eBay Inc.

982 reviews

Founded in 1995 in San Jose, Calif., eBay (NASDAQ:EBAY) is where the world goes to shop, sell and give. Whether you're buying new or used,...

Data Scientist/Applied Researcher

eBay Inc. ★★★★☆ 982 reviews - San Jose, CA

Looking for a company that inspires passion, courage and imagination, where you can be part of the team shaping the future of global commerce? Want to shape how millions of people buy, sell, connect, and share around the world? If you're interested in joining a purpose driven community that is dedicated to creating an ambitious and inclusive workplace, join eBay – a company you can be proud to be a part of.

Looking for a company that inspires passion, courage and imagination, where you can be part of the team shaping the future of global commerce? Want to shape how millions of people buy, sell, connect, and share around the world? If you're interested in joining a purpose driven community that is dedicated to creating an ambitious and inclusive workplace, join eBay – a company you can be proud to be a part of.

Join the Search Science team at eBay!

Do you have what it takes to improve a world-class real-time search engine that serves millions of queries a day? Do you thrive on developing data mining techniques to pull insight
eCommerce re
transaction da

We are passio
online marketp
scientists

You will

- Be directly responsible for improving search recall and ranking via query understanding and machine learned ranking models that power the end-end search experience at eBay.
- Showcase the spectrum of value in the world's most diverse ecommerce inventory, by improving whole page relevance, query recommendations, spell corrections, synonym/acronym expansions, and query rewrites.
- Conceptualize, code, deploy, and iterate on designs from prototypes all the way through to production systems.
- Analyze petabytes of real-world behavioral data to understand patterns and trends.
- Transform data insights into actionable reports, targeting algorithms, and model features.
- Provide technical leadership in statistical analysis, data mining, machine learning, NLP, and information retrieval.

Company Info

Follow

Get job updates from Apple

Apple

★★★★☆ 5,066 reviews

This is where you can do the best work of your life. Where you'll join some of the world's smartest, most innovative people to create...

Siri - Data Scientist

Apple ★★★★☆ 5,066 reviews - San Francisco, CA 94114

The Siri Search team is building groundbreaking technology for algorithmic search, machine learning, natural language processing, and artificial intelligence. The features we build are redefining how hundreds of millions of people use their computers and mobile devices to search and find what they are looking for. Siri's universal search engine powers search features across a variety of Apple products, including Siri, Spotlight, Safari, Messages and Lookup. As part of this group, you will work with one of the most exciting high performance computing environments, with petabytes of data, millions of queries per second, and have an opportunity to imagine and build products that delight our customers every single day.

Key Qualifications

- 3+ years of experiences in a Machine Learning or a Data Science role
- You have excellent knowledge and good practical skills in major machine learning algorithms
- You have experience with machine learning tools and libraries such as Scikit-learn, TensorFlow, Spark
- You have excellent data analytical and problem solving skills
- Mastery of one of the following languages: Python, Go, Java, C++
- Your experience with large scale search and machine learning systems is helpful
- Good
- Good

Description

As a member of our fast-paced group, you'll have the unique and rewarding opportunity to shape upcoming products from Apple. We are looking for people with excellent applied machine learning experience and solid engineering skills in creating outstanding search service. This role will have the following responsibilities: Analyzing search ranking requirements, issues and opportunities Understanding product requirements, translate them into modeling tasks and engineering tasks Building machine learned models for search relevance, ranking and content understanding problems Integrating search functions into Apple products, such as Siri, Spotlight, Safari, Messages, Lookup, etc. Utilizing Spark, Hadoop MapReduce, HBase, Hive, Impala, Kafka, and RabbitMQ to perform distributed data processing Work alone or as part of small team to deliver complete systems Work project management and other engineering teams

Learning to rank algorithms

Least Square Retrieval Function (TOIS 1989)

Query refinement (WWW 2008)

SVM-MAP (SIGIR 2007)

ListNet (ICML 2007)

Pranking (NIPS 200

LambdaRank (NIPS 2006)

Frank (SIGIR

MHR (SIGIR 2007)

RankBoost (JMLR 2003)

LDM (SIGI

Large margin ranker (NIPS 2002)

Ranking SVM (ICANN 199

RankNet (ICML 2005)

Discriminative model for IR (SIGIR 2004)

OAP-BPM (ICML 2003)

GPRank (LR4IR 2007)

QBRank (NIPS 2007)

GBRar

Constraint Ordinal Regression (ICML 2005)

McRank (NIPS 2

AdaRank (SIGIR 2007)

CCA (SIGIR 2007)

RankCosine (IP&M 2007)

Supervised Rank

Year	Name
2008	ListMLE
2008	PermuRank
2008	SoftRank
2008	Ranking Refinement[24]
2008	SSRankBoost[25]
2008	SortNet[26]
2009	MPBoost
2009	BoltzRank
2009	BayesRank
2010	NDCG Boost[27]
2010	GBlend
2010	IntervalRank
2010	CRR
2014	LCR
2015	FaceNet
2016	XGBoost
2017	ES-Rank
2018	DLCM[28]
2018	PolyRank[29]
2018	FATE-Net/FETA-Net[30]
2019	FastAP[31]
2019	Mulberry
2019	DirectRanker
2019	GSF[32]
2020	PRM[33]
2020	SetRank[34]
2021	PiRank[35]

https://en.wikipedia.org/wiki/Learning_to_rank

Simple example: Using classification for ad hoc IR

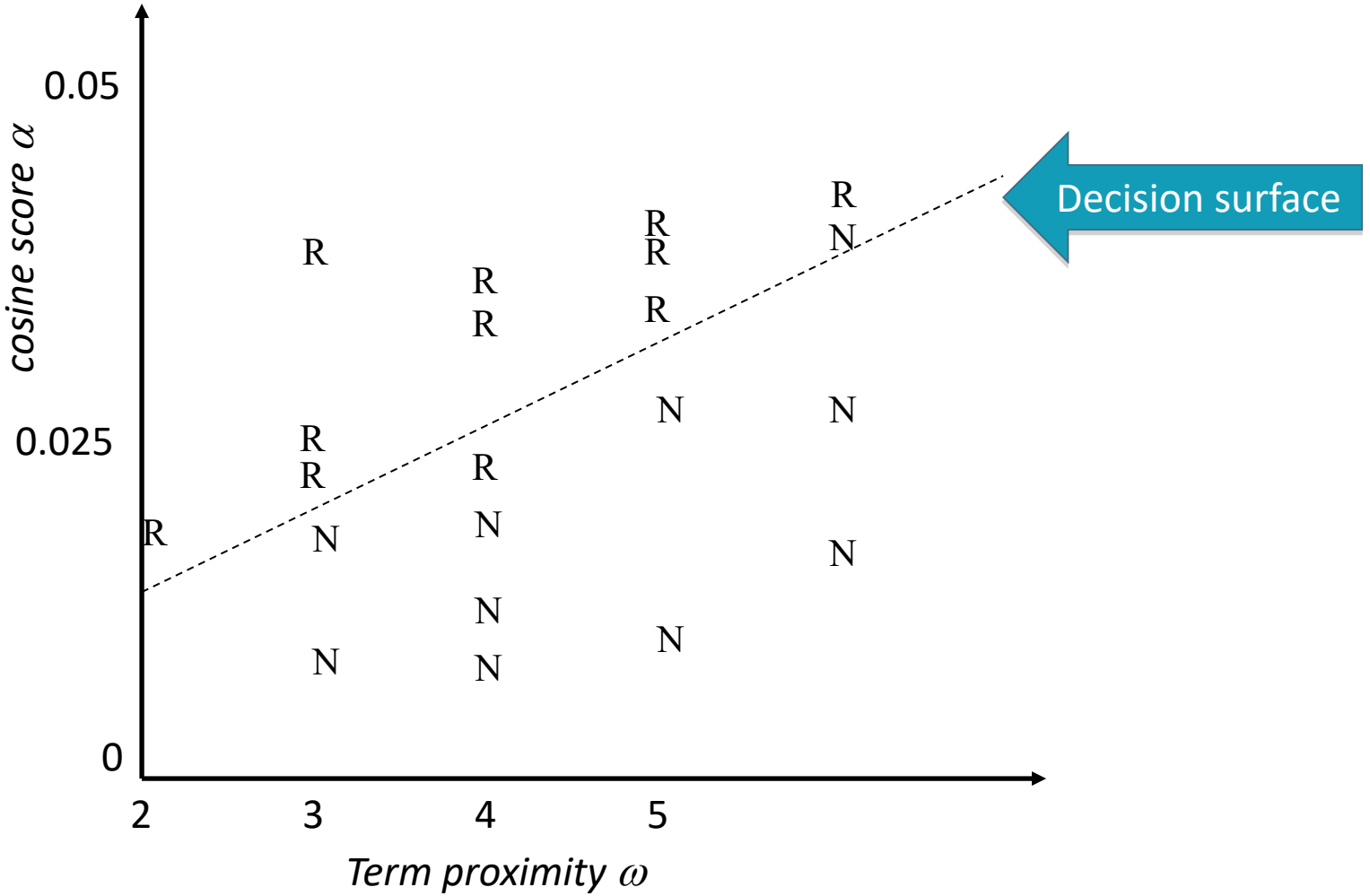
- Collect a training corpus of (*q, d, r*) triples
  - Relevance *r* is here binary
  - Document is represented by a feature vector
  - **x** = (α, ω): α is cosine similarity, ω is minimum query window size
    - Query term proximity is a **very important** weighting factor
- Train a machine learning model to predict the class *r* of a document-query pair

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*

Simple example: Using classification for ad hoc IR

- A linear score function is then
  - *Score(d, q) = Score(α, ω) = aα + bω + c*
- And the linear classifier is
  - Decide relevant if *Score(d, q) > threshold*
- … this is exactly like text classification

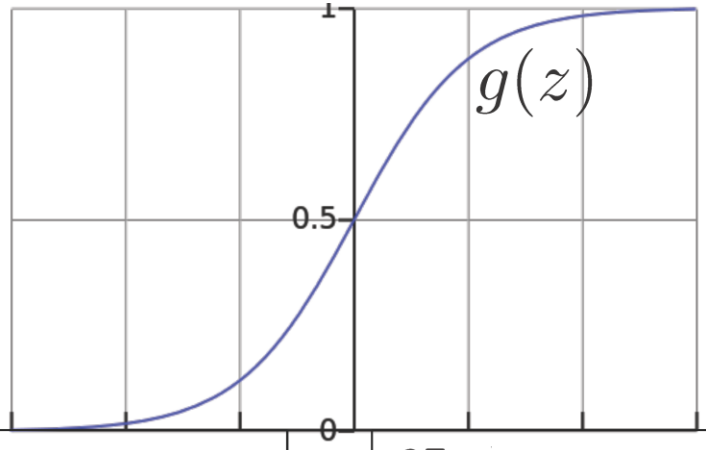
Simple example: Using classification for ad hoc IR



e.g., Logistic Regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}\right)$$

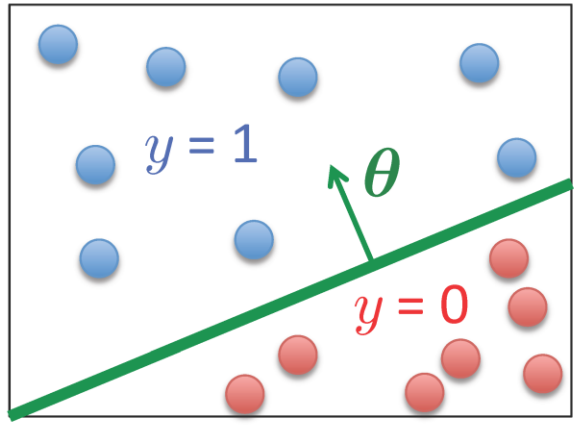
$$g(z) = \frac{1}{1+e^{-z}}$$



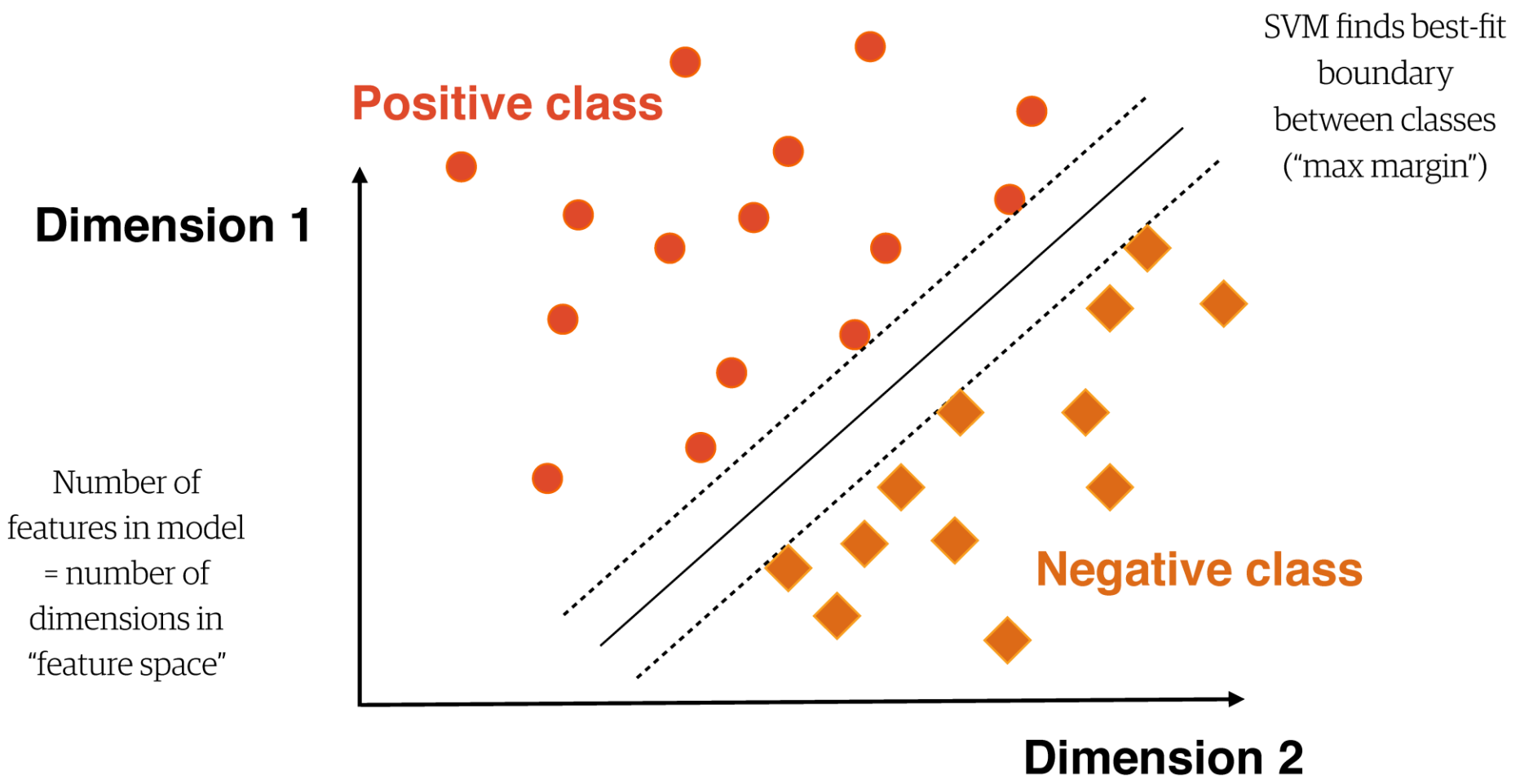
$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large negative values for negative instances

$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large positive values for positive instances

- Assume a threshold and...
  - Predict $y = 1$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0.5$
  - Predict $y = 0$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$



e.g., Support Vector Machine (SVM)



Extending the model

- We can generalize this to classifier functions over more features

- Machine learning for IR ranking has been actively researched – and actively deployed by major web search engines

Why is ML needed now?

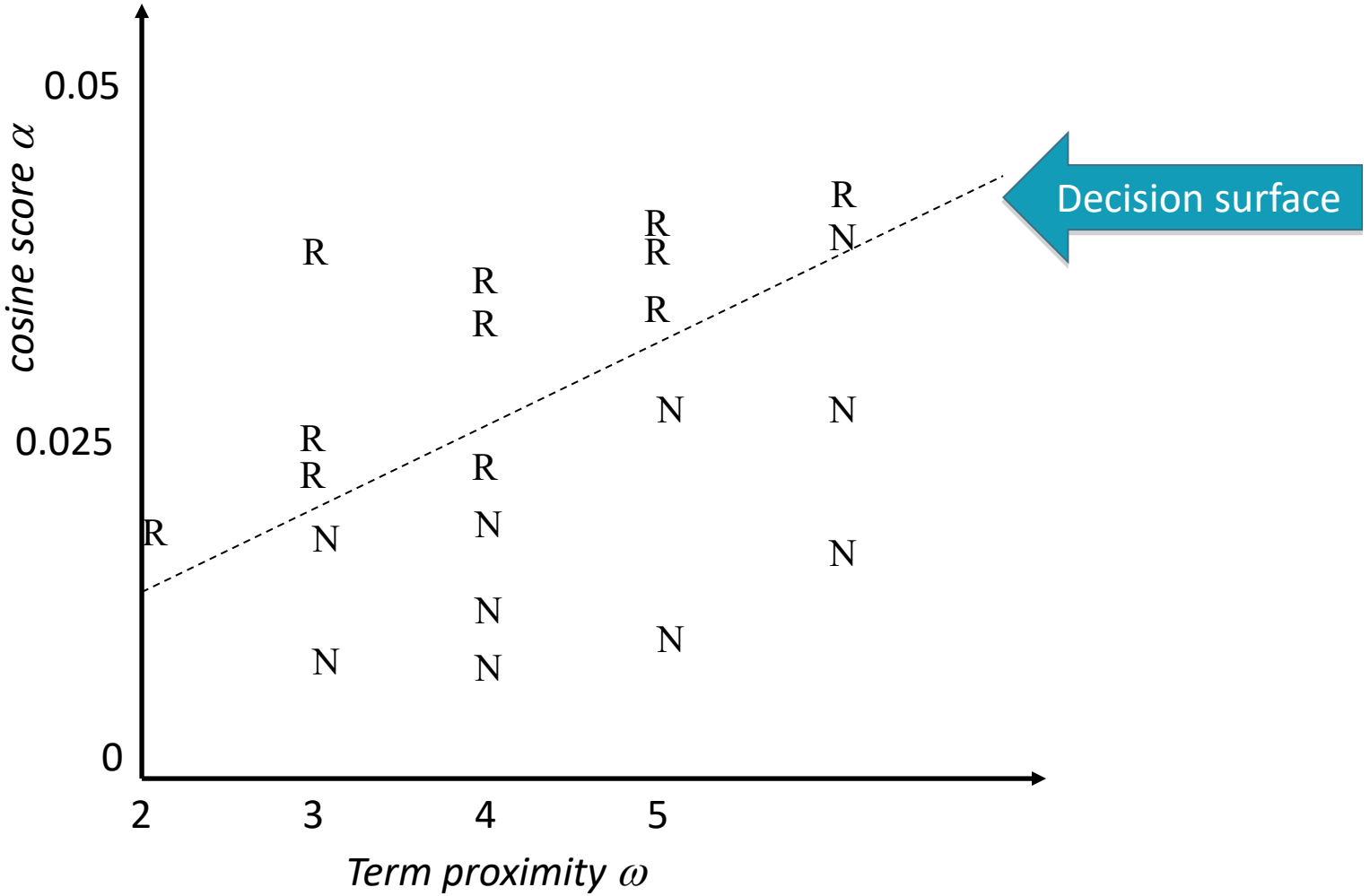
- Modern systems – especially on the Web – use a great number of features:
  - Arbitrary useful features – not a single unified model
  - Log frequency of query word in anchor text?
  - Query word in color on page?
  - # of images on page?
  - # of (out) links on page?
  - PageRank of page?
  - URL length?
  - URL contains "~"?
  - Page edit recency?
  - Page length?
- The *New York Times* reported that Google was using **over 200 such features** in 2008.

136 Features released from Microsoft Research in 2010

http://research.microsoft.com/en-us/projects/mslr/feature.aspx

Zones: body, anchor, title, url, whole document
Features: query term number, query term ratio, stream length, idf, sum of term frequency, min of term frequency, max of term frequency, mean of term frequency, variance of term frequency, sum of stream length normalized term frequency, min of stream length normalized term frequency, max of stream length normalized term frequency, mean of stream length normalized term frequency, variance of stream length normalized term frequency, sum of tf*idf, min of tf*idf, max of tf*idf, mean of tf*idf, variance of tf*idf, boolean model, vector space model, BM25, LMIR.ABS, LMIR.DIR, LMIR.JM, number of slash in url, length of url, inlink number, outlink number, PageRank, SiteRank, QualityScore, QualityScore2, query-url click count, url click count, url dwell time.

Simple example: Using classification for ad hoc IR



Result ranking by machine learning

- The above ideas can be readily generalized to functions of **many more than two** variables
- In addition to cosine similarity and query term window, there are lots of other indicators of relevance, e.g. PageRank-style measures, document age, zone contributions, document length, etc.
- If these measures can be calculated for a training document collection with relevance judgments, any number of such measures can be used to train a machine learning classifier

Learning to rank

- Purpose
  - Learn a function automatically to rank results (items) effectively
- Point-wise learning
  - The function is based on features of a single object
  - e.g., regress the relevance score, classify docs into R and NR
  - Classic retrieval models are also point-wise: score(q, D)
- Pair-wise learning
  - The function is based on a pair of items
  - e.g., given two documents, predict partial ranking
- List-wise learning
  - The function is based on a ranked list of items
  - e.g., given two ranked list of the same items, which is better? Or can we learn cross-document interactions and learn the best order of the documents.

https://en.wikipedia.org/wiki/Learning_to_rank

Learning to rank

- Classification probably isn't the right way to think about approaching ad hoc IR:
  - Classification problems: Map to a unordered set of classes
  - Regression problems: Map to a real value
  - Ordinal regression problems: Map to an *ordered* set of classes
    - A fairly obscure sub-branch of statistics, but what we want here
- This formulation gives extra power:
  - Relations between relevance levels are modeled
  - Documents are good versus other documents for query given collection; not an absolute scale of goodness

Learning to rank

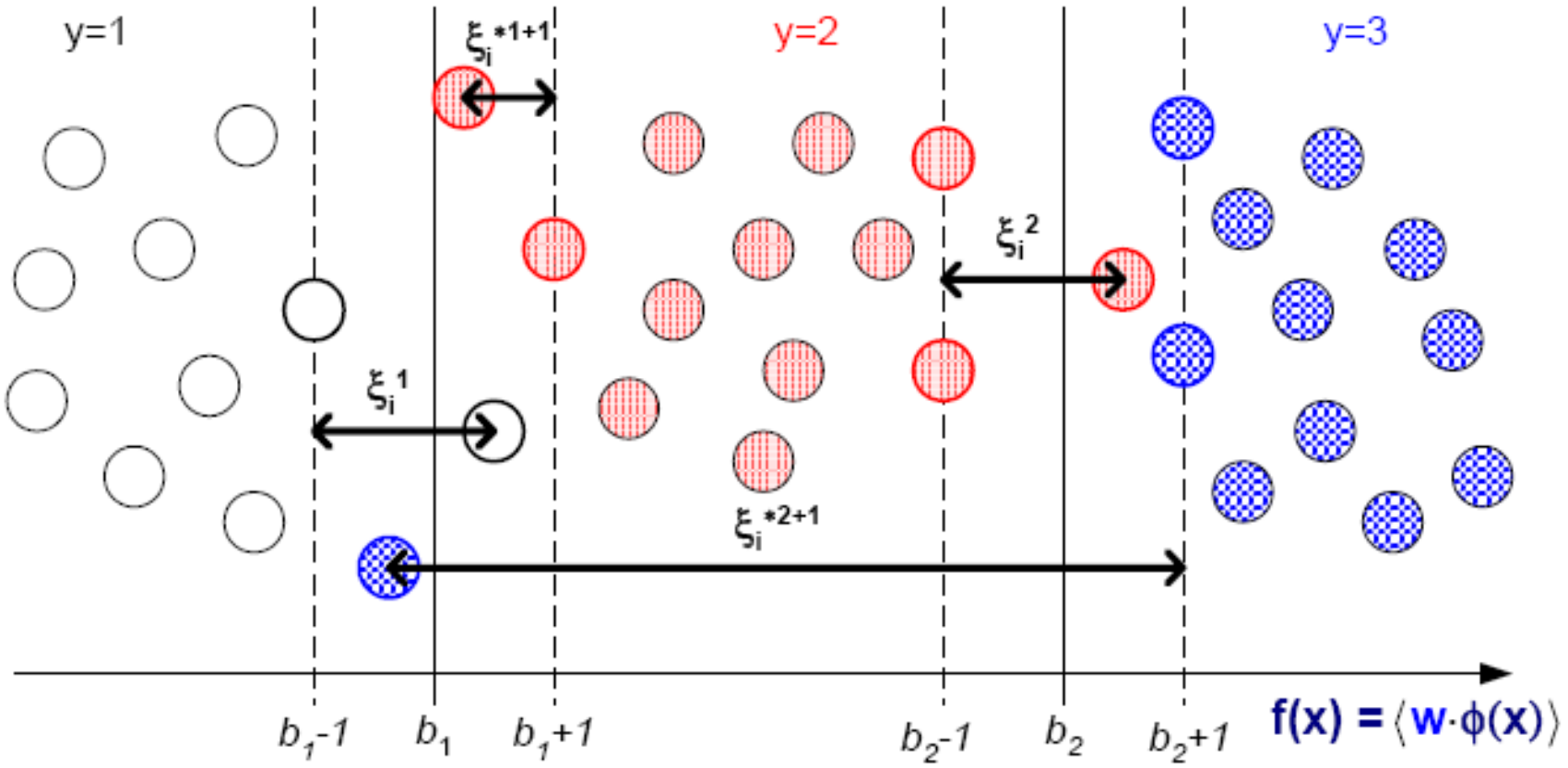
- Assume a number of categories ***C*** of relevance exist
  - These are totally ordered: $c_1 < c_2 < \ldots < c_J$
  - This is the ordinal regression setup

- Assume training data is available consisting of document-query pairs represented as feature vectors ψ_i and relevance ranking c_i

- We could do ***point-wise*** **learning**, where we try to map items of a certain relevance rank to a subinterval (e.g, Crammer et al. 2002 Prank)

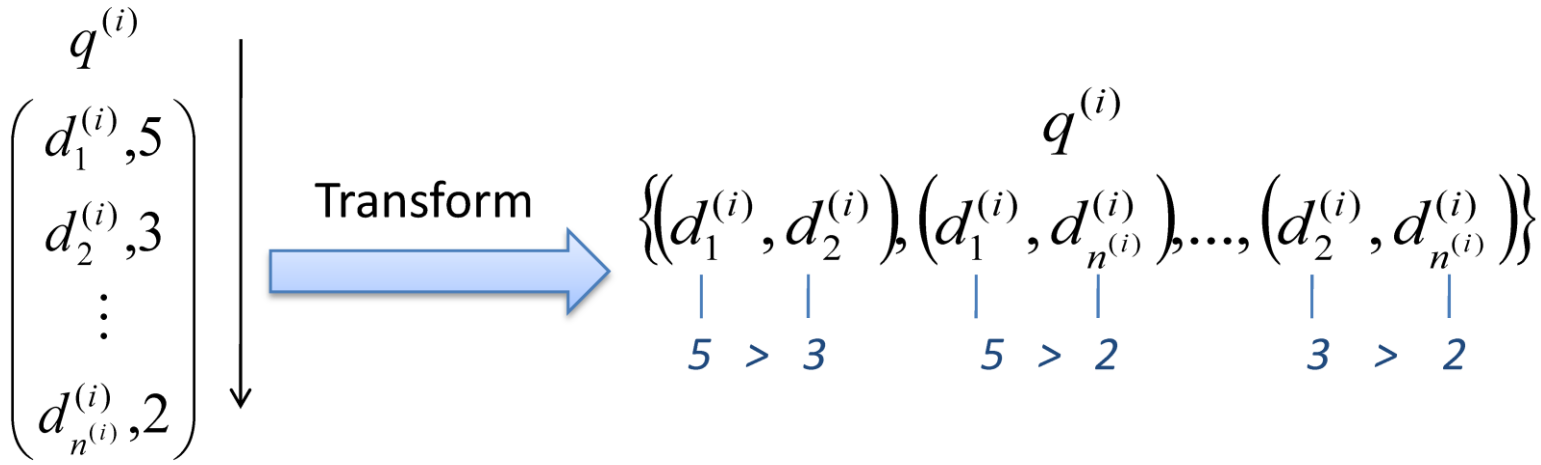
Point-wise learning

- Goal is to learn a threshold to separate each rank



Learning to rank

- But most work does ***pair-wise* learning**, where the input is a pair of results for a query, and the class is the relevance ordering relationship between them



https://en.wikipedia.org/wiki/Learning_to_rank

An Example of the Pairwise approach: The Ranking SVM

- We begin with a set of judged queries
- For each training query q, we have a set of documents returned in response to the query, which have been totally ordered by a person for relevance to the query
- We construct a vector of features $\psi_j = \psi(d_j , q)$ for each document/query pair, using features such as those discussed, and many more
- For two documents d_i and d_j , we then form the vector of feature differences:

$$\Phi(d_i , d_j , q) = \psi(d_i , q) - \psi(d_j , q)$$

The construction of a ranking SVM

- By hypothesis, one of d_i and d_j has been judged more relevant
- If d_i is judged more relevant than dj , denoted $d_i \prec d_j$ (d_i should precede d_j in the results ordering), then we will assign the vector $\Phi(d_i , d_j , q)$ the class $y_{ijq} = +1$; otherwise -1
- The goal then is to build a classifier which will return

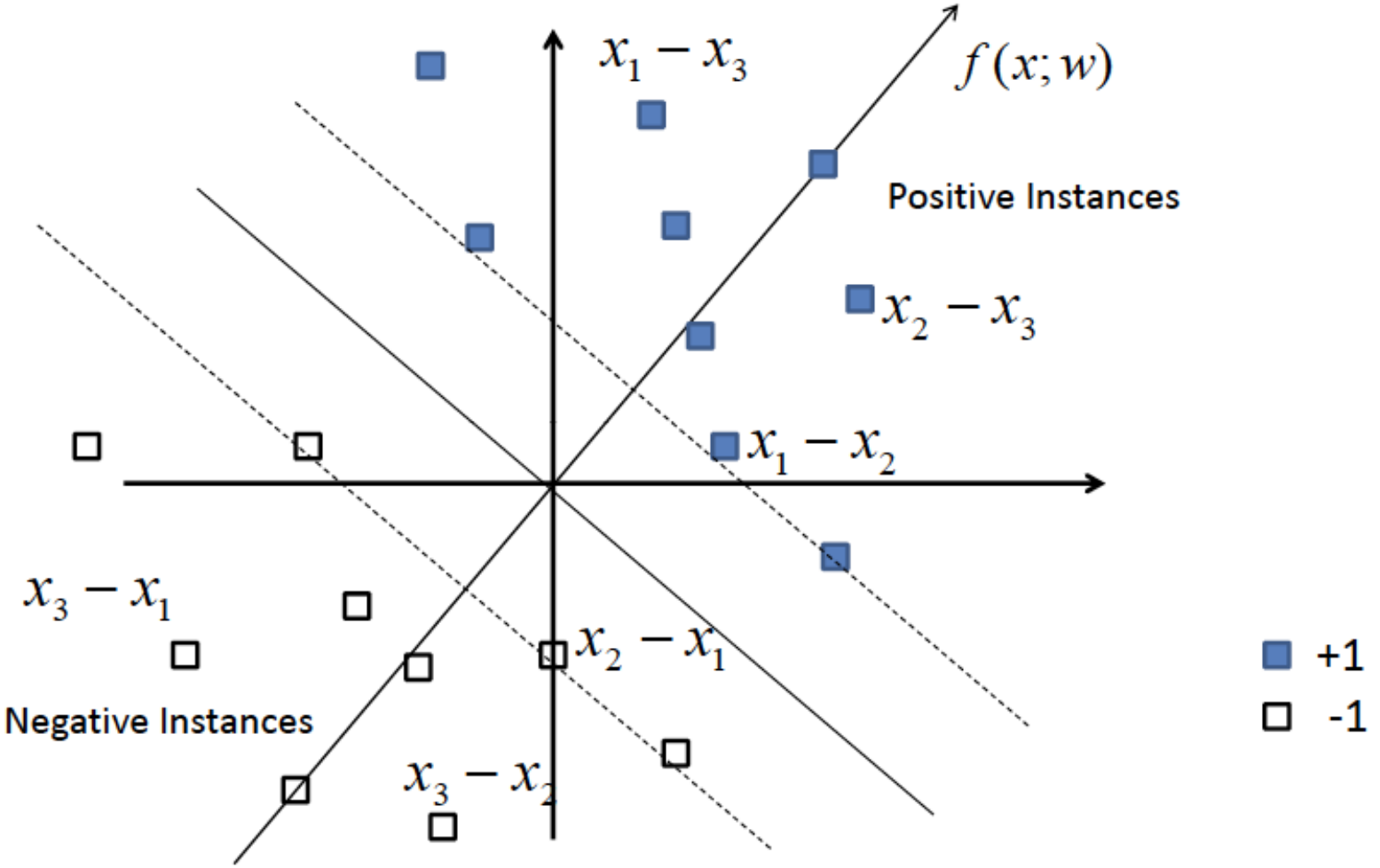
$$\mathbf{w} \bullet \Phi(d_i , d_j , q) > 0 \text{ iff } d_i \prec d_j$$

Testing the Ranking SVM

- Input: a pair of documents Di and Dj
  - Characterized by feature vectors ψ_i *and* ψ_j

- Classify:
  - *rank i before j* iff $\mathbf{w} \bullet (\psi_i - \psi_j) > 0$

An Example of the Pairwise approach: The Ranking SVM



Ranking SVM

- This approach has been used to build ranking functions which outperform standard hand-built ranking functions in IR evaluations on standard data sets

The ranking classifier fails to model the IR problem well …

- Correctly ordering the most relevant documents is crucial to the success of an IR system, while misordering less relevant results matters little
  - The ranking SVM considers all ordering violations as the same
- Some queries have many (somewhat) relevant documents, and other queries few. If we treat all pairs of results for a query equally, queries with many results will dominate the learning
  - But actually queries with few relevant results are at least as important to do well on

https://en.wikipedia.org/wiki/Learning_to_rank

SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval -- An example of list-wise learning

- Given a candidate documents per query by a traditional ranking method (e.g., BM25 and LambdaMART), rerank the candidate documents.
- Consider two requirements:
  - (1) Model cross-document interactions so as to capture local context information in a query
  - (2) Be permutation invariant, which means that any permutation of the inputted documents would not change the output ranking.

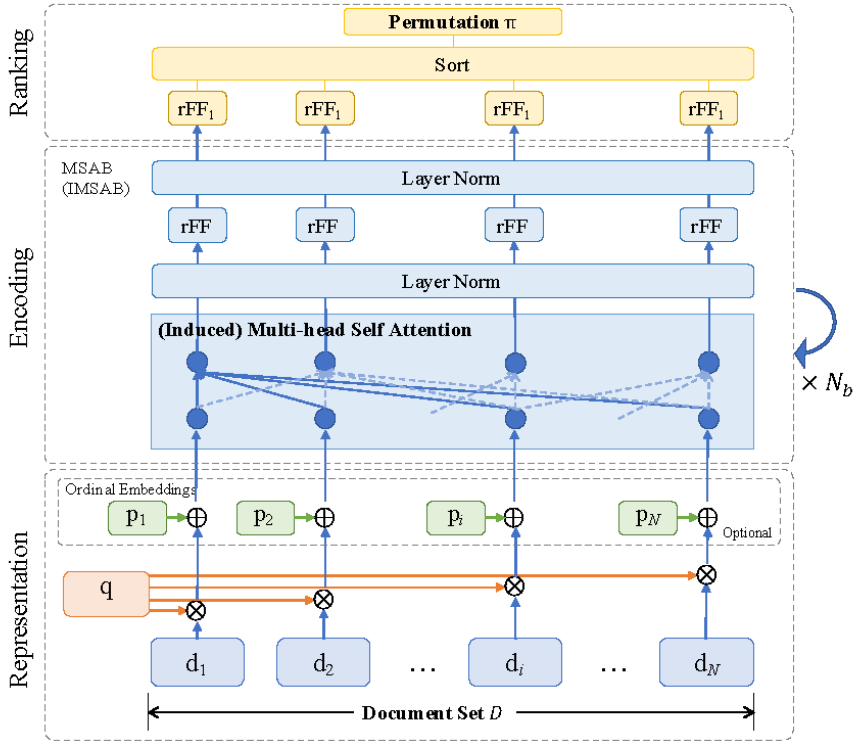


Figure 1: Architecture of SetRank. The representation layer generates representation of query-document pairs separately; the encoding layer jointly process the documents with multiple sub-layers of MSAB (or IMSAB), and output internal document representations; the ranking layer calculates the scores and sorts the documents.

https://arxiv.org/abs/1912.05891

1. Document Representation

Given a query q and its associated document set $D = [d_1, d_2, \cdots, d_N]$, each of the document in D can be represented as a feature vector

$$\mathbf{d}_i = \phi(q, d_i), \text{ where } \mathbf{d}_i \in \mathbb{R}^E,$$

where ϕ is the function for feature extraction and E is the dimension of the vector. The features extracted in traditional learning-to-rank are used here, including document only feature of PageRank, query-document matching features of TF-IDF, BM25 etc. In the experiments of this paper, we used the features provided by the benchmark datasets.

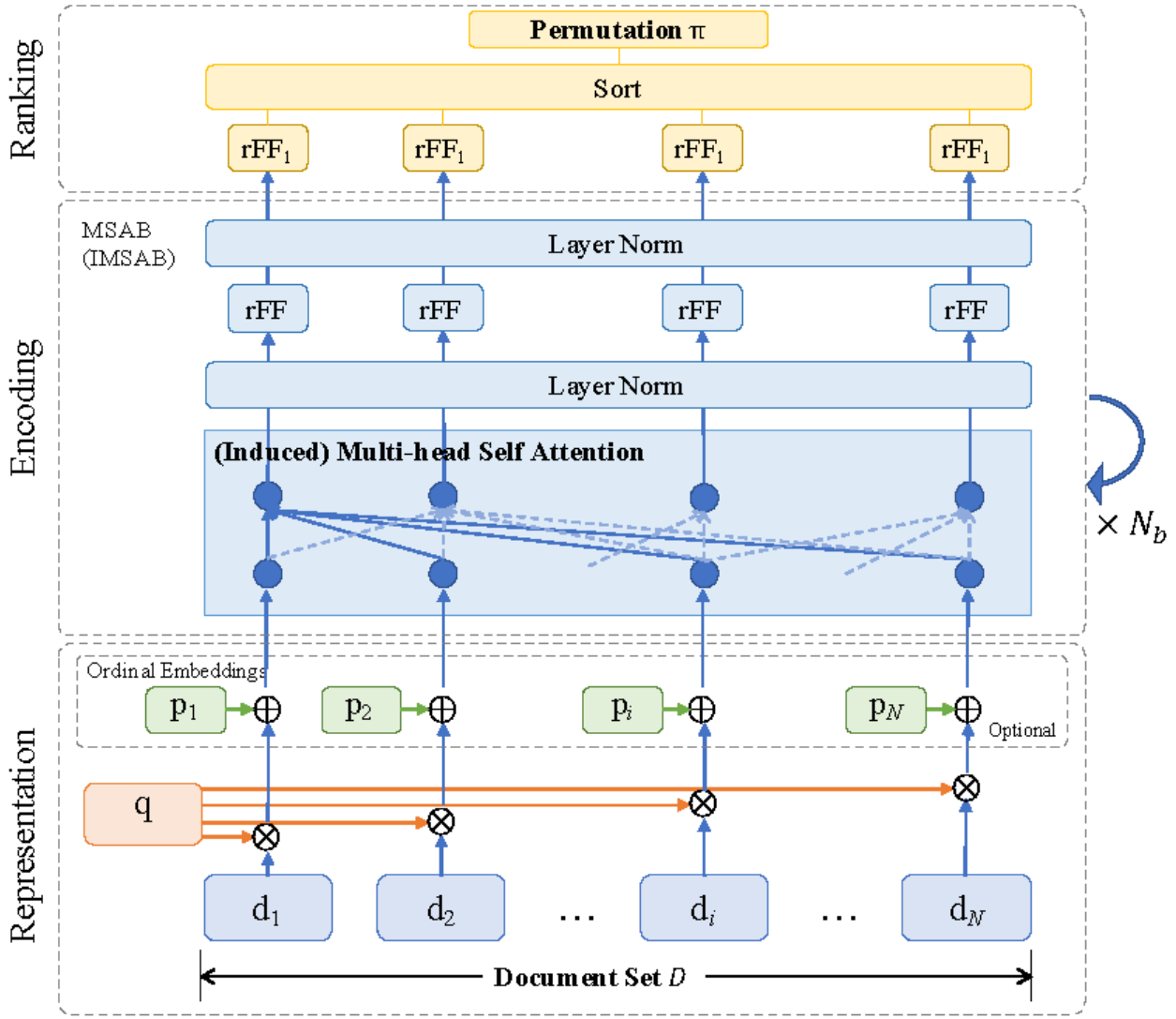
$$\mathbf{p}_i = P(\text{rank}(d_i)), \text{ where } \mathbf{p}_i \in \mathbb{R}^E,$$

where $\text{rank}(d_i)$ denotes the absolute rank position of d_i in the initial ranking generated by models such as BM25, LambdaMART etc.

Output of document representation

$$\mathbf{X} = [\mathbf{d}_1 + \mathbf{p}_1, \mathbf{d}_2 + \mathbf{p}_2, \cdots, \mathbf{d}_N + \mathbf{p}_N]^T.$$

$$\mathbf{X} \in \mathbb{R}^{N \times E}$$



2. Document Encoding with (Induced) Multi-head Self Attention Block

Multi-head attention block (MAB)

$$\begin{aligned} \mathrm{MAB}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= \mathrm{LayerNorm}(\mathbf{B} + \mathrm{rFF}(\mathbf{B})), \\ \text{where } \mathbf{B} &= \mathrm{LayerNorm}(\mathbf{Q} + \mathrm{Multihead}(\mathbf{Q},\mathbf{K},\mathbf{V})) \end{aligned} \tag{5}$$

where $\mathrm{rFF}(\cdot)$ is a row-wise feedforward layer, and $\mathrm{LayerNorm}(\cdot)$ is layer normalization [3].

Two approaches:

a) Multi-head self attention block (MSAB)

$$\mathrm{MSAB}(\mathbf{X}) = \mathrm{MAB}(\mathbf{X},\mathbf{X},\mathbf{X}).$$

b) Induced multi-head self attention block (IMSAB)

First, construct M fake attention query vectors, denoted as, $\mathbf{I} \in \mathbb{R}^{M \times E}$, to extract information from original keys/values. M is less than N (# of candidate docs)

$$\mathrm{IMSAB}_M(\mathbf{X}) = \mathrm{MAB}(\mathbf{X},\mathbf{H},\mathbf{H}), \text{ where } \mathbf{H} = \mathrm{MAB}(\mathbf{I},\mathbf{X},\mathbf{X})$$

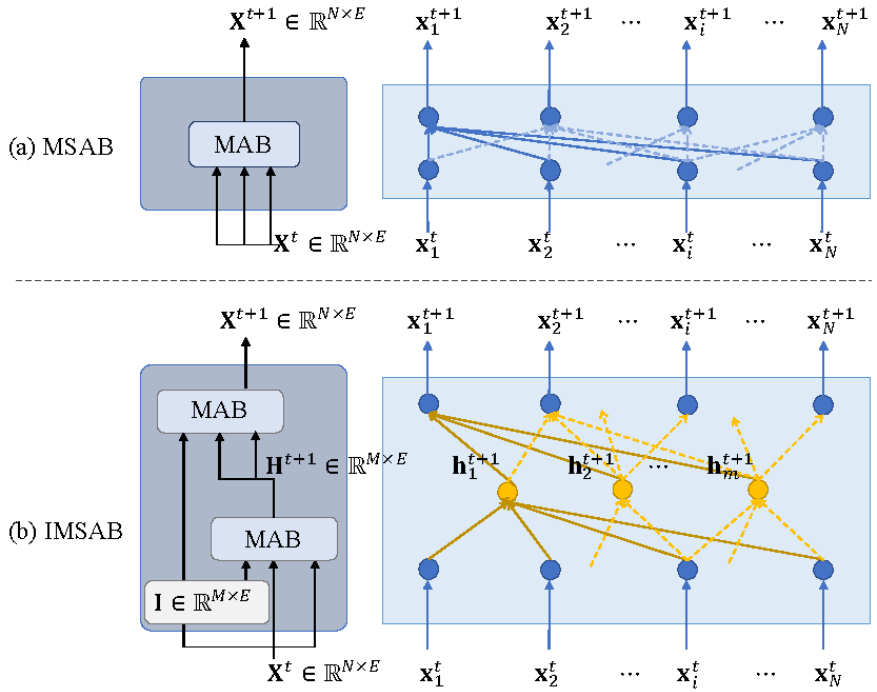


Figure 2: (a) The structure of multi-head self attention block (MSAB). MSAB encodes an arbitrary set $\mathbf{X}^t$ of size N and output an set $\mathbf{X}^{t+1}$ of size N. (b) The structure of induced multi-head self attention block (IMSAB) with induced size of M. IMSAB first encodes an arbitrary set $\mathbf{X}^t$ of size N to a fixed set $\mathbf{H}^{t+1}$ of size M, and then encodes $\mathbf{H}^{t+1}$ to an N-size output $\mathbf{X}^{t+1}$. IMSAB can be interpreted as first creating M cluster centers and then encoding the inputted N documents using these M cluster centers.

3. Document Ranking

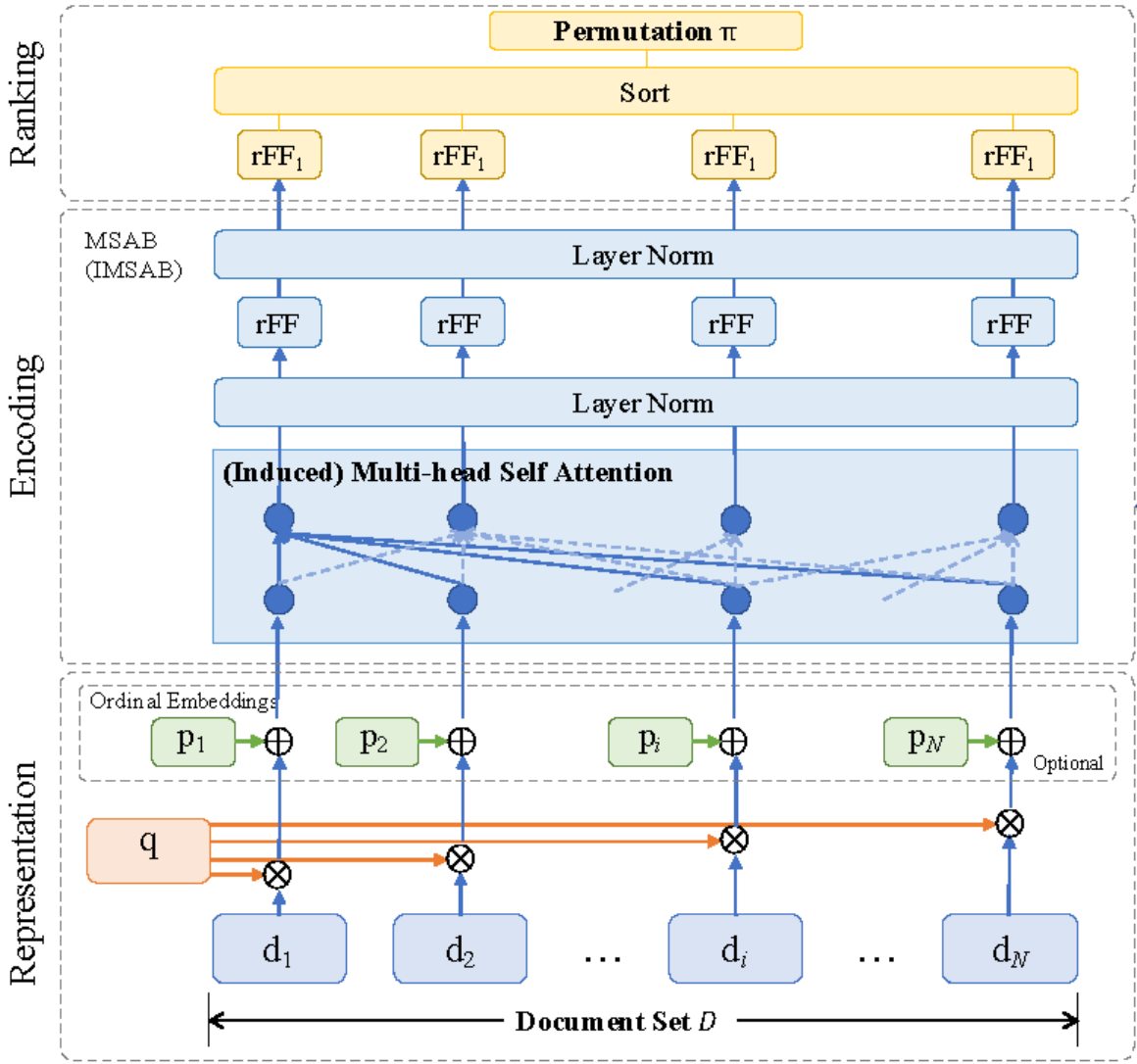
Stacking aforementioned MSAB or IMSABM blocks and passing the results to row-wise feedforward neural networks, we achieve two versions of SetRank scoring functions, respectively named as SetRankMSAB and SetRankIMSAB:

$$\mathbf{X}_{\mathrm{MSAB}}^{N_b} = \underbrace{\mathrm{MSAB}(\mathrm{MSAB}\ldots(\mathrm{MSAB}}_{N_b}(\mathbf{X}^0))),$$

$$\mathrm{SetRank}_{\mathrm{MSAB}}(D) = \mathrm{rFF}_1\left(\mathbf{X}_{\mathrm{MSAB}}^{N_b}\right),$$

Finally, sort each doc's score

$$\hat{\pi}_{\mathrm{MSAB}} = sort \circ \mathrm{SetRank}_{\mathrm{MSAB}}(D).$$



Training and Datasets

- Model Training

Given a labeled query $\psi_q = \{D = \{d_i\}, \mathbf{y} = \{y_i\} | 1 \le i \le N\}$, where y_i is the relevance label and denotes the information gain of document d_i for query q. The optimal attention allocation strategy in terms of relevance labels for document $d_i (i = 1, \cdots, N)$ is:

$$a_i^y = \frac{\tau(y_i)}{\sum_{d_k \in D} \tau(y_k)}, \quad \tau(x) = \begin{cases} \exp(x) & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Similarly, given the predicted ranking scores $\{s_1, \cdots, s_N\} = \text{SetRank}(D)$, the attention w.r.t. the predicted scores for document $d_i (i = 1, \cdots, N)$ is:

$$a_i^s = \exp(s_i) / \sum_{d_k \in D} \exp(s_k).$$

Therefore, the list-wise cross entropy loss function can be constructed on the basis of these two attention distributions:

$$\mathcal{L} = \sum_{d_i \in D} a_i^y \log a_i^s + (1 - a_i^y) \log(1 - a_i^s).$$

Table 1: Dataset statistics.

Property	Istella	MSLR 30K	Yahoo!
#features	220	136	700
#queries in training	20,317	18,919	19,944
#queries in validation	2,902	6,306	2,994
Total #docs in train	7,325,625	2,270,296	473,134
#queries in test	9,799	6,306	6,983
Total #docs in test	3,129,004	753,611	165,660
Avg. #docs per query in test	319.31	119.51	23.72

Results

Table 2: Performance comparison of different models on Istella, MSLR30K and Yahoo datasets. Significant performance improvements (paired t-test with p-value ≤ 0.05) over LambdaMart and DLCM are denoted as '+' and '†', respectively. Boldface indicates the best performed results.

(a) Ranking accuracies on Istella LETOR dataset

	NDCG			
Model	@1	@3	@5	@10
RankSVM	0.5269	0.4867	0.5041	0.5529
RankBoost	0.4457	0.3977	0.4097	0.4511
Mart	0.6185	0.5633	0.5801	0.6285
LambdaMart	0.6571	0.5982	0.6118	0.6591
Without initial rankings				
DLCM$^{w/o\ init}$	0.6272	0.5717	0.5848	0.6310
GSF	0.6224	0.5796	0.5968	0.6508
SetRank$_{\text{MSAB}}$	0.6702$^{+\dagger}$	0.6150$^{+\dagger}$	0.6282$^{+\dagger}$	0.6766$^{+\dagger}$
SetRank$_{\text{IMSAB}}$	0.6733$^{+\dagger}$	0.6136$^{+\dagger}$	0.6278$^{+\dagger}$	0.6737$^{+\dagger}$
With initial rankings generated by LambdaMart				
DLCM	0.6558	0.6030^{+}	0.6194^{+}	0.6680^{+}
SetRank$^{init}_{\text{MSAB}}$	0.6745$^{+\dagger}$	0.6201$^{+\dagger}$	**0.6350**$^{+\dagger}$	0.6819$^{+\dagger}$
SetRank$^{init}_{\text{IMSAB}}$	**0.6760**$^{+\dagger}$	**0.6202**$^{+\dagger}$	0.6345$^{+\dagger}$	**0.6834**$^{+\dagger}$

(b) Ranking accuracies on Microsoft LETOR 30K dataset

	NDCG			
Model	@1	@3	@5	@10
RankSVM	0.3010	0.3180	0.3350	0.3650
RankBoost	0.2788	0.2897	0.3043	0.3339
Mart	0.4436	0.4344	0.4414	0.4633
LambdaMart	0.4570	0.4420	0.4450	0.4640
Without initial rankings				
DLCM$^{w/o\ init}$	0.3985	0.3919	0.4001	0.4245
GSF	0.4129	0.4073	0.4151	0.4374
SetRank$_{\text{MSAB}}$	0.4243	0.4116	0.4177	0.4403
SetRank$_{\text{IMSAB}}$	0.4290	0.4166	0.4220	0.4428
With initial rankings generated by LambdaMart				
DLCM	**0.4630**$^{+}$	0.4450^{+}	0.4500^{+}	0.4690^{+}
SetRank$^{init}_{\text{MSAB}}$	0.4572	0.4452^{+}	0.4499^{+}	0.4692^{+}
SetRank$^{init}_{\text{IMSAB}}$	0.4591^{+}	**0.4469**$^{+\dagger}$	**0.4515**$^{+}$	**0.4696**$^{+}$

(c) Ranking accuracies on Yahoo! LETOR challenge set1 dataset

	NDCG			
Model	@1	@3	@5	@10
RankSVM	0.6370	0.6500	0.6740	0.7260
RankBoost	0.6293	0.6409	0.6661	0.7159
Mart	0.6830	0.6827	**0.7034**	**0.7469**
LambdaMart	0.6770	0.6760	0.6960	0.7380
Without initial rankings				
DLCM$^{w/o\ init}$	0.6693	0.6751	0.6958	0.7391
GSF	0.6429	0.6604	0.6838	0.7316
SetRank$_{\text{MSAB}}$	0.6623	0.6698	0.6911	0.7369
SetRank$_{\text{IMSAB}}$	0.6711	0.6760	0.6960	0.7398
With initial rankings generated by LambdaMart				
DLCM	0.6760	0.6810^{+}	0.6990^{+}	0.7430^{+}
SetRank$^{init}_{\text{MSAB}}$	**0.6837**$^{+\dagger}$	0.6820^{+}	0.7009^{+}	0.7443^{+}
SetRank$^{init}_{\text{IMSAB}}$	0.6822$^{+\dagger}$	**0.6835**$^{+\dagger}$	0.7029^{+}	0.7453$^{+\dagger}$

Without initial rankings mean without incorporating ordinal embedding. Simple $\mathbf{X} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \ldots, \mathbf{d}_N]$

Distributed Representation and Neural IR

Some slides were adapted from Chris Manning's notes

How do we represent the meaning of a word?

Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

signifier (symbol) $\Leftrightarrow$ signified (idea or thing)

How do we have usable meaning in a computer?

Common solution: Use e.g. WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
        print("{}: {}".format(poses[synset.pos()],
                        ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good  adj:
good
adj (sat): estimable, good, honorable, respectable  adj (sat):
beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")  hyper =
lambda s: s.hypernyms()
list(panda.closure(hyper))
```

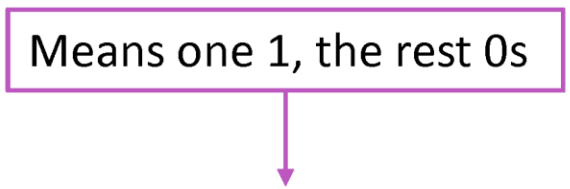
```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Problems with resources like WordNet

- Great as a resource but missing nuance
  - e.g. "proficient" is listed as a synonym for "good". This is only correct in some contexts.
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can't compute accurate word similarity

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation



Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

Problem with words as discrete symbols

Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel".

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal.

There is no natural notion of **similarity** for one-hot vectors!

- **Instead: learn to encode similarity in the vectors themselves**

Representing words by their context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

Word meaning as a neural word vector – visualization

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word2vec: Overview

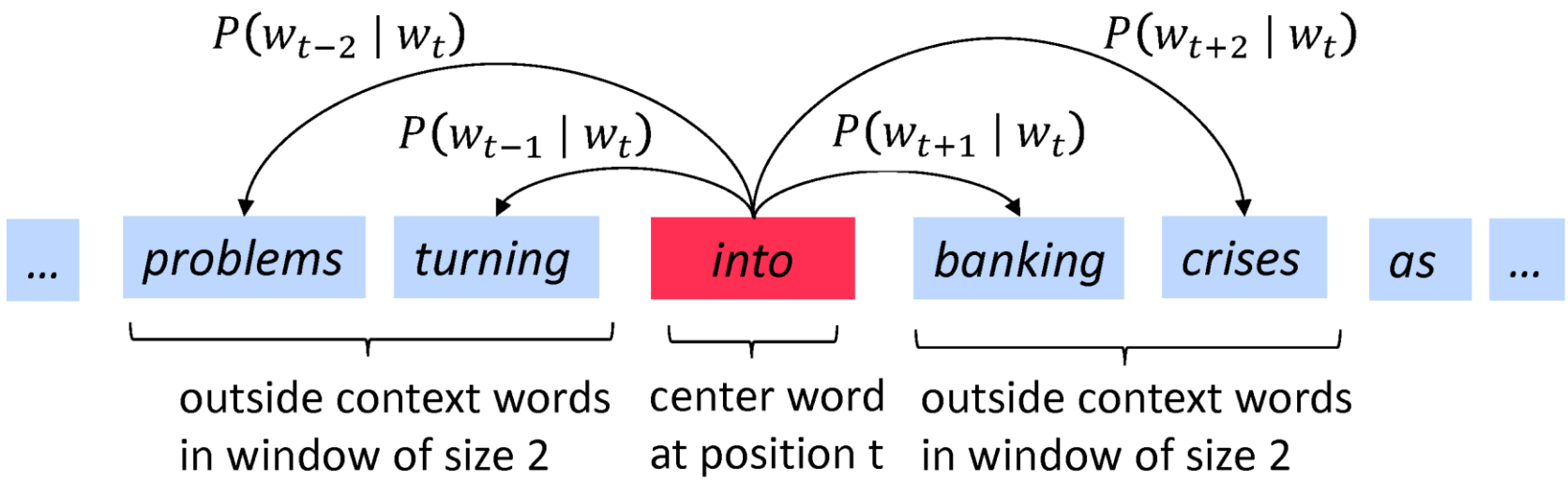
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

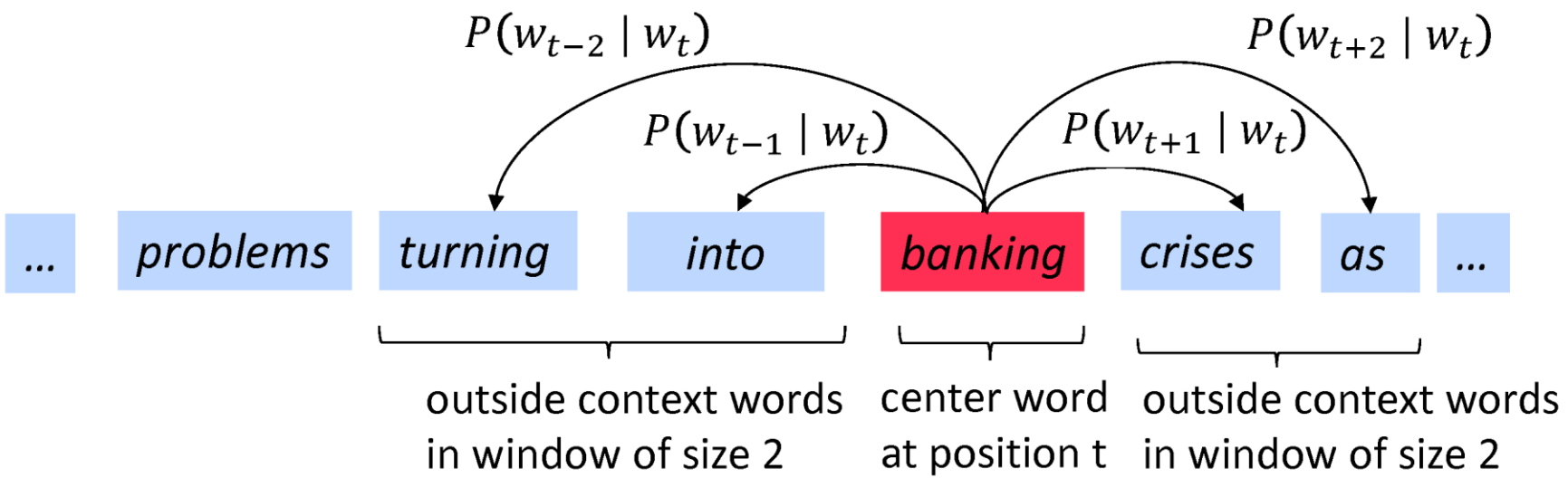
Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

Likelihood = $$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- Answer: We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2vec: prediction function

Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

Normalize over entire vocabulary
to give probability distribution

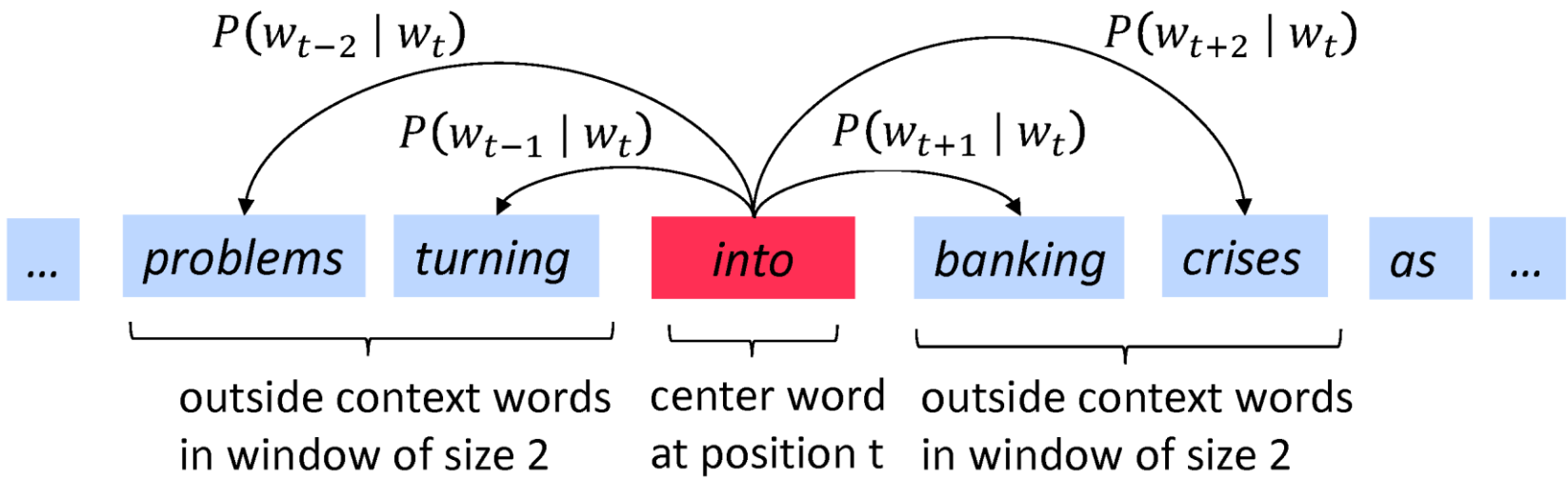
- This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

Main idea of word2vec

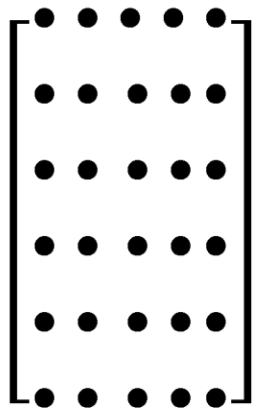
- Iterate through each word of the whole corpus
- Predict surrounding words using word vectors



- $P(o|c) = \dfrac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

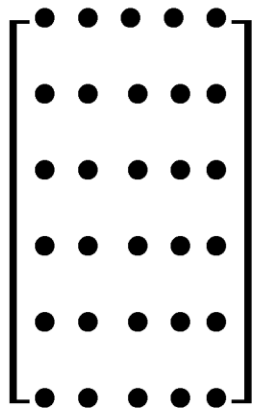
- Update vectors so you can predict well

Word2vec parameters and computations



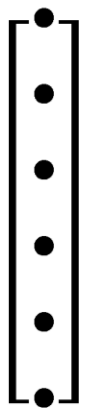
U

outside



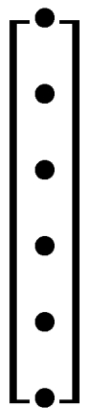
V

center



$U.{v_4}^T$

dot product



$\mathrm{softmax}(U.{v_4}^T)$

probabilities

Same predictions at each position

Word2vec: More details

Why two vectors? → Easier optimization. Average both at the end

- But can do it with just one vector per word

Two model variants:

1. Skip-grams (SG)

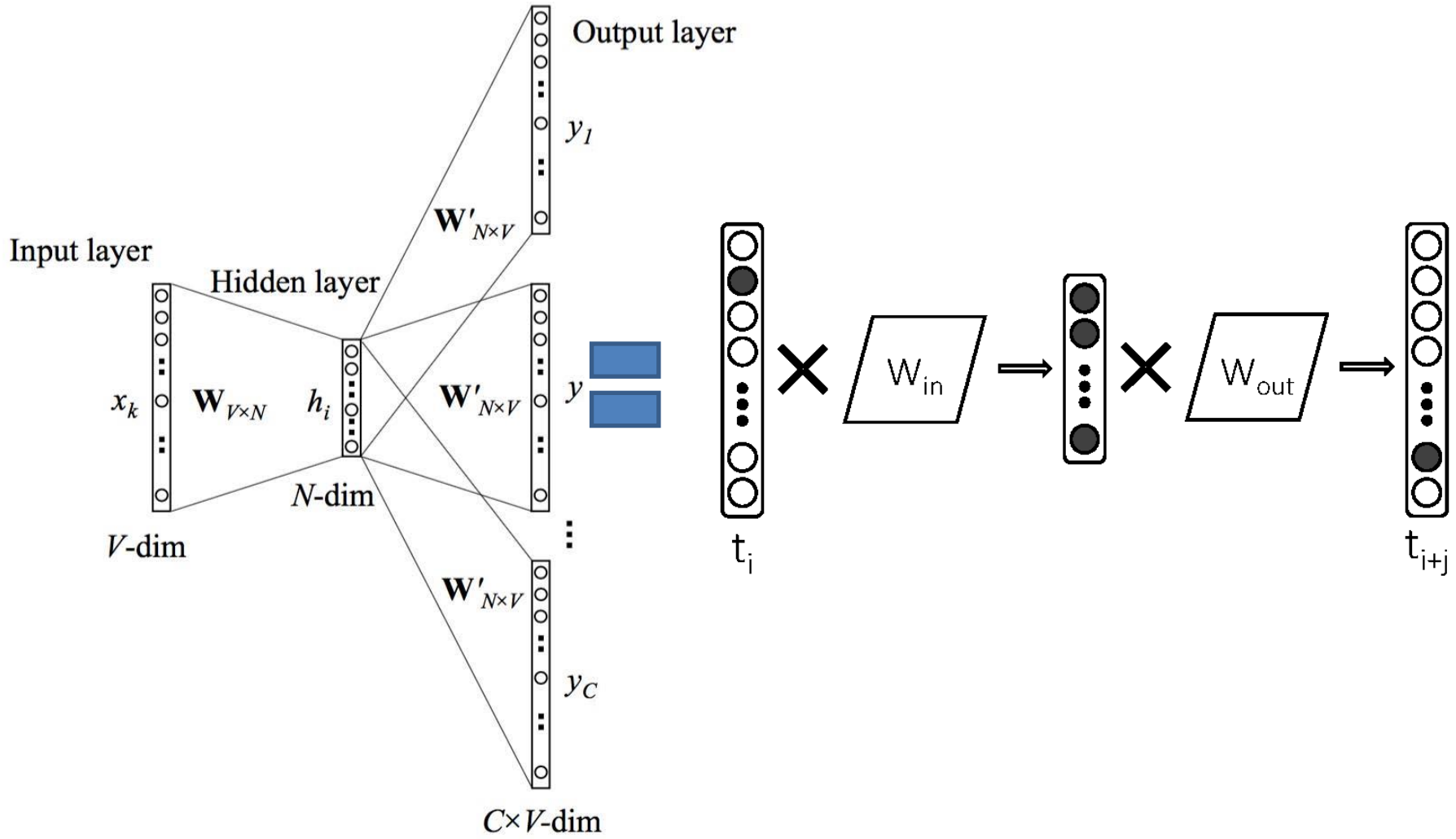
   Predict context ("outside") words (position independent) given center word

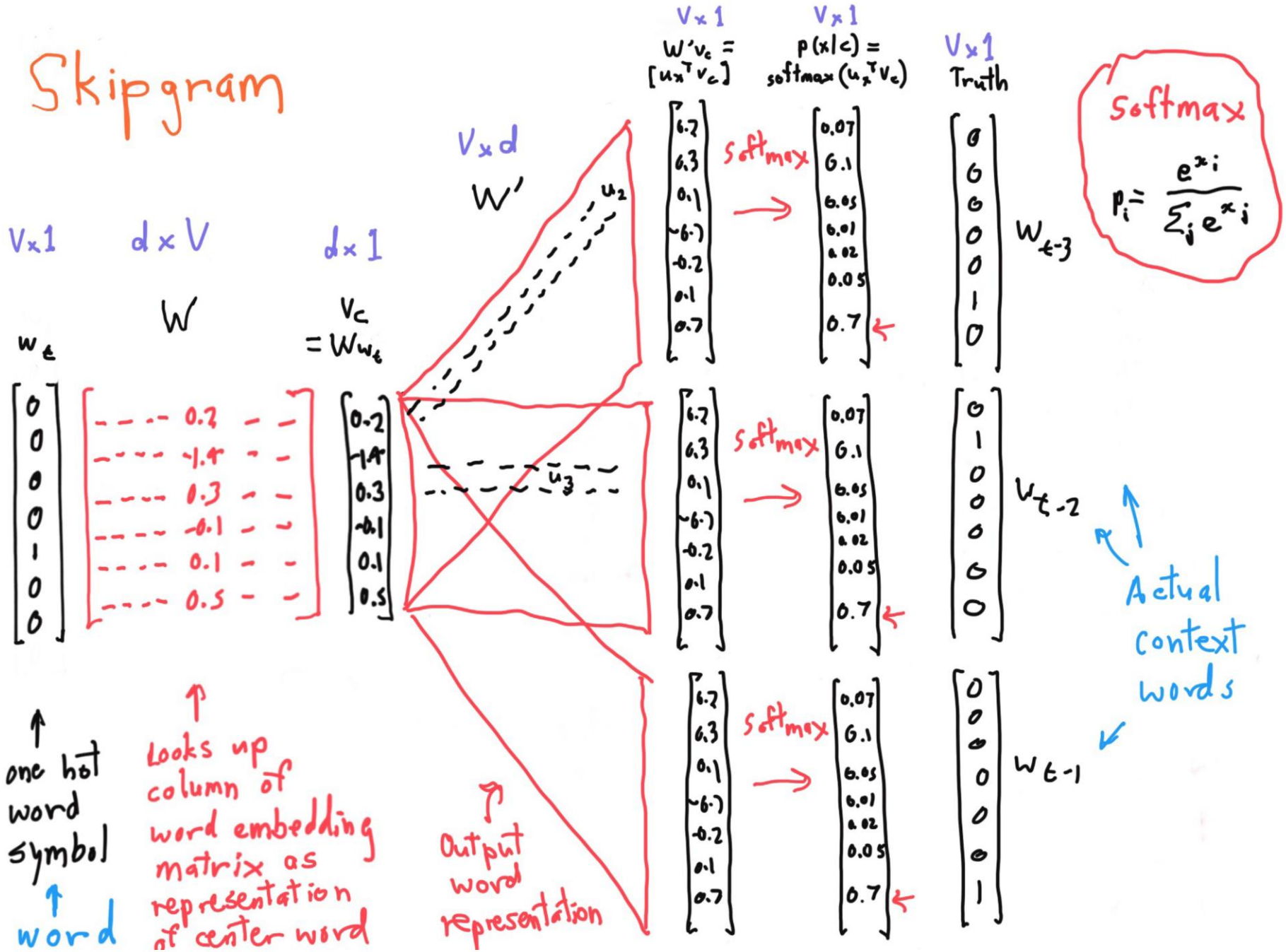
2. Continuous Bag of Words (CBOW)

   Predict center word from (bag of) context words

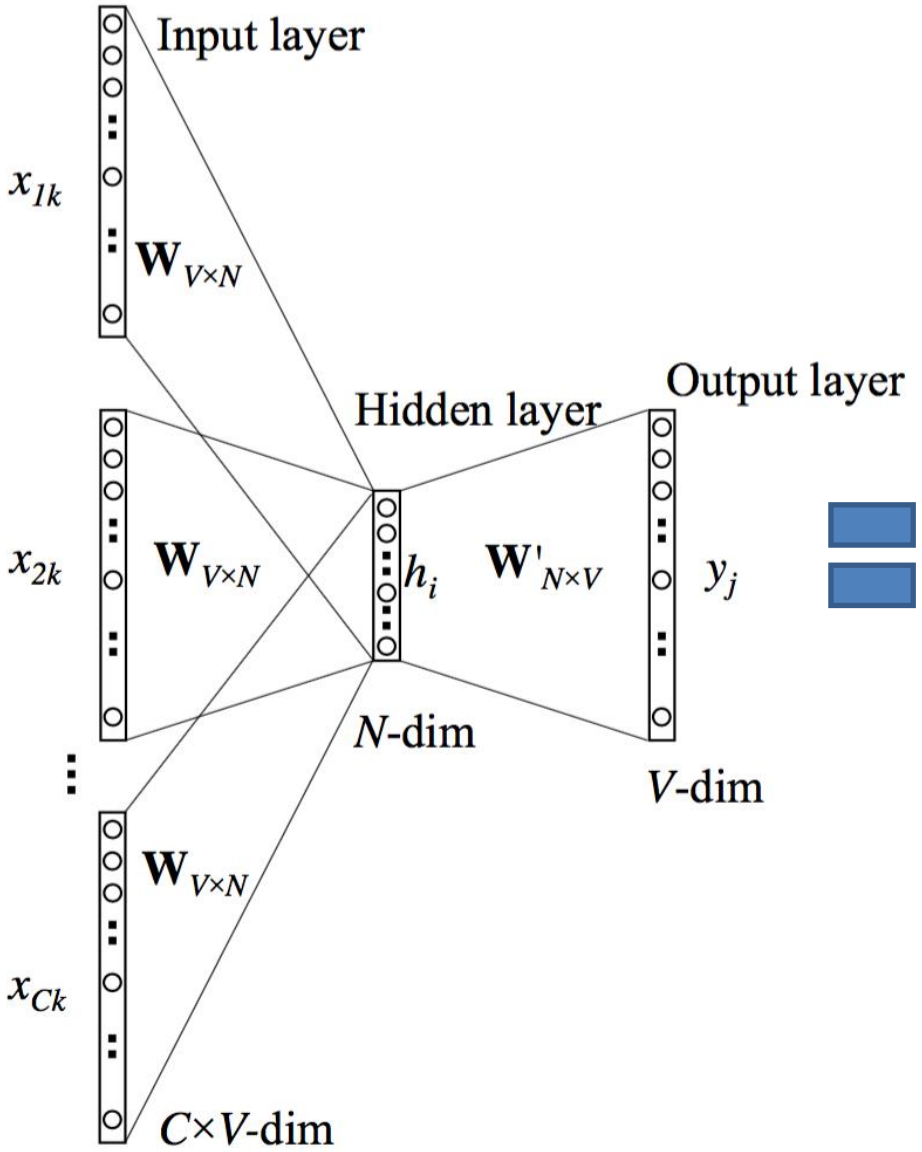
We presented: **Skip-gram model**

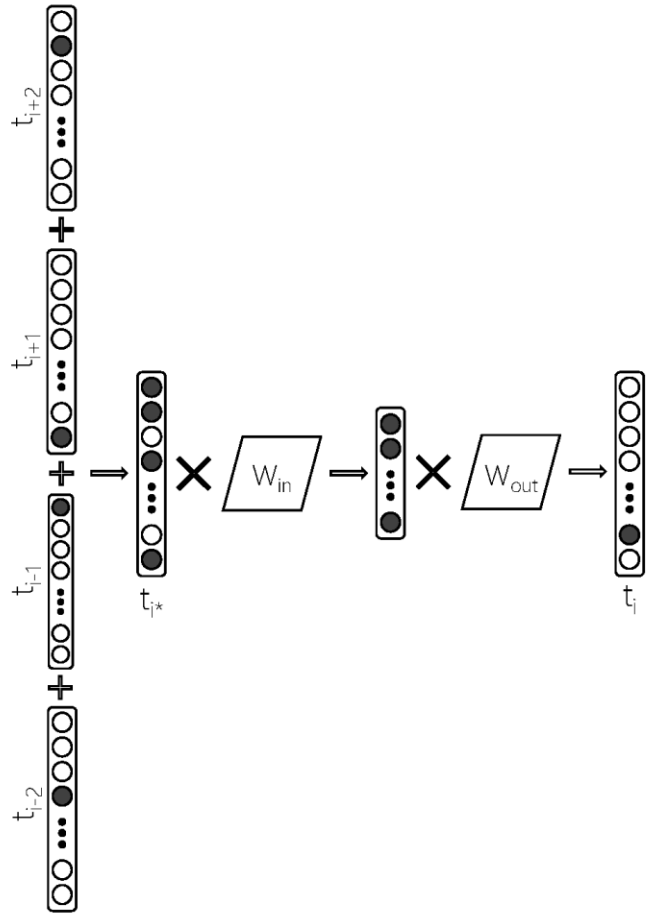
Skip-grams (SG)



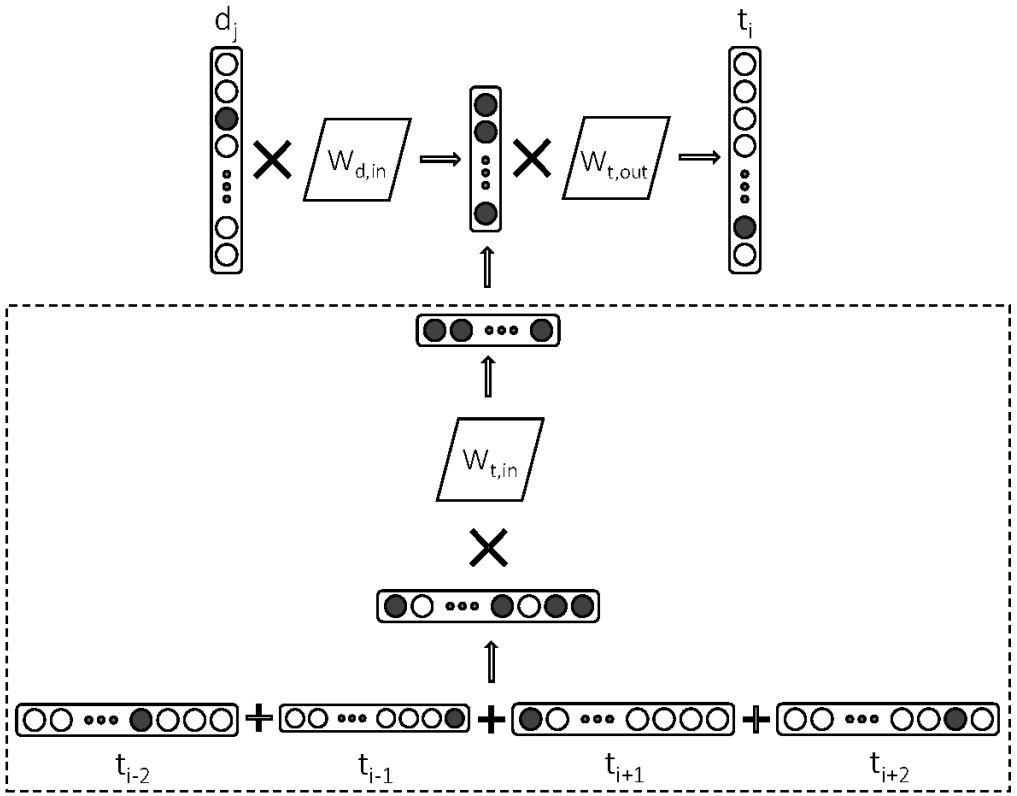

Skipgram
Softmax
$p_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$
w_t
V x 1
one hot word symbol
word
W
d x V
Looks up column of word embedding matrix as representation of center word
$v_c = Wx_t$
d x 1
W'
V x d
u_2
u_3
Output word representation
$W'v_c = [u_x^T v_c]$
V x 1
$p(x|c) = \text{softmax}(u_x^T v_c)$
V x 1
softmax
Truth
V x 1
W_{t-3}
W_{t-2}
W_{t-1}
Actual context words

Continuous Bag of Words (CBOW)





Paragraph2Vec



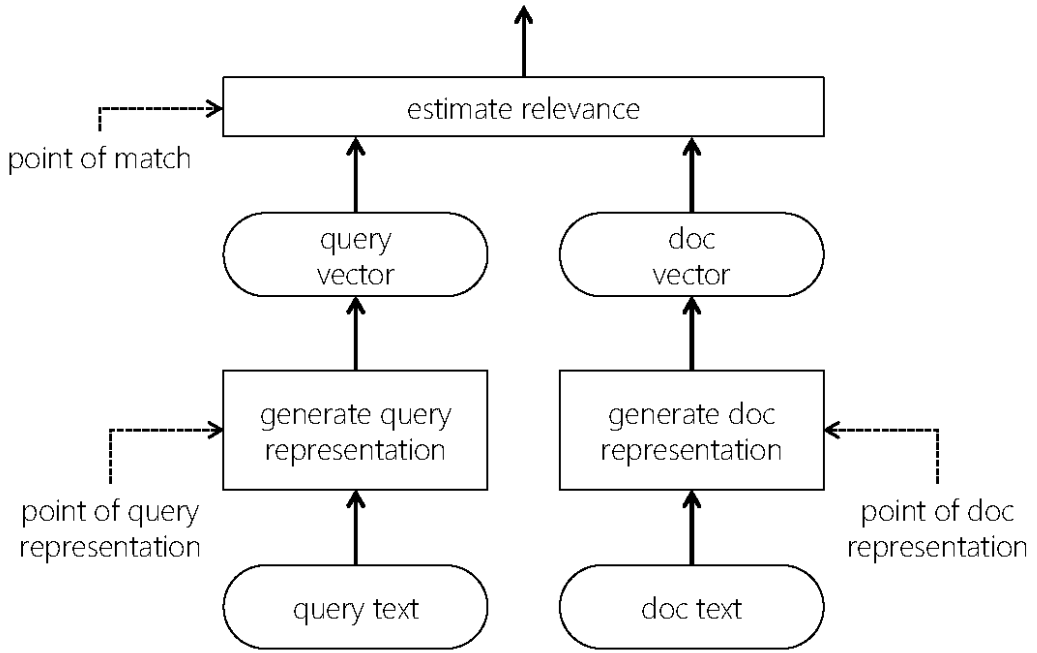
- The paragraph2vec architecture trains by predicting a term given a document (or passage) ID containing the term.
- By trying to minimize the prediction error, the model learns an embedding for the
- term as well as for the document.
- In some variants of the architecture, optionally the neighboring terms are also provided as input—as shown in the dotted box.

Distributed Representations of Sentences and Documents: https://arxiv.org/pdf/1405.4053.pdf

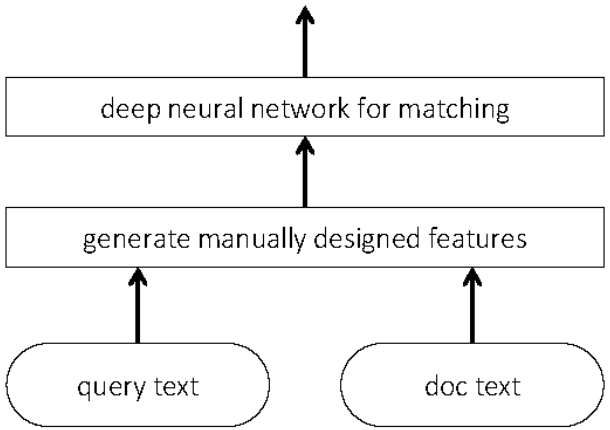
https://radimrehurek.com/gensim/models/doc2vec.html

Neural IR

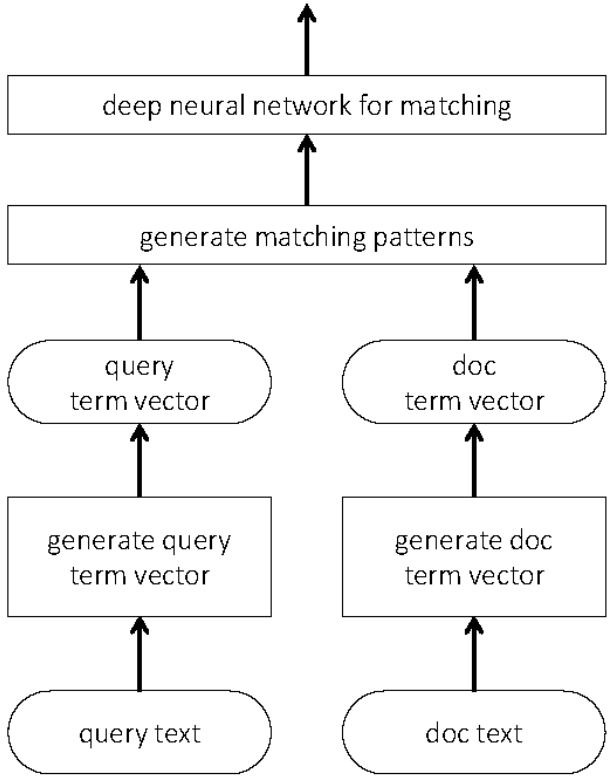
- Three components of IR:
  - the query representation
  - the document representation
  - estimating relevance between them
- Neural approaches can be applied to any of the three components.



Neural IR



(a) Learning to rank using manually designed features (*e.g.*, Liu (2009))



(b) Estimating relevance from patterns of exact matches (*e.g.*, (Guo *et al.*, 2016a; Mitra *et al.*, 2017a))

- In (a) and (b) the neural network is only used at the point of matching

A Deep Relevance Matching Model for Ad-hoc Retrieval

– example of Neural IR (b)

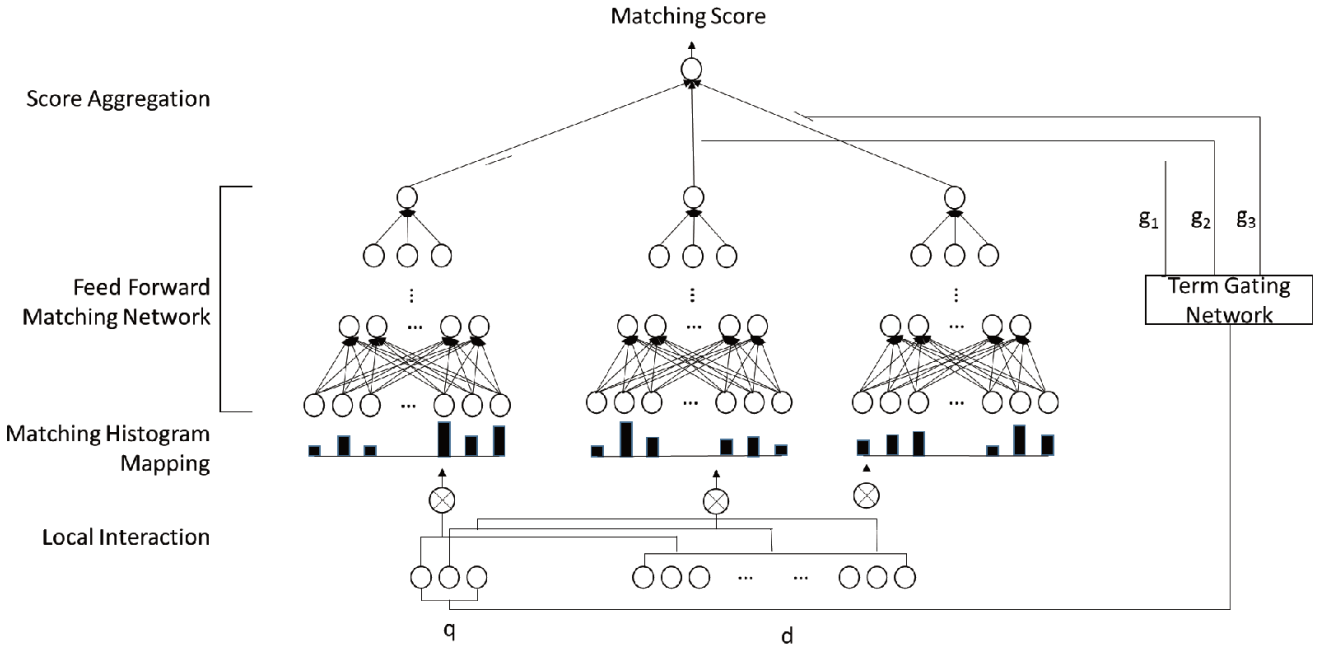


Figure 2: Architecture of the Deep Relevance Matching Model.

More formally, suppose both query and document are represented as a set of term vectors denoted by $q=\{w_1^{(q)}, \ldots, w_M^{(q)}\}$ and $d = \{w_1^{(d)}, \ldots, w_N^{(d)}\}$, where $w_i^{(q)}, i = 1, \ldots, M$ and $w_j^{(d)}, j = 1, \ldots, N$ denotes a query term vector and a document term vector, respectively, and s denotes the final rel-

$$\boldsymbol{z}_i^{(0)} = h(w_i^{(q)} \otimes d), \qquad i = 1, \ldots, M$$

$$\boldsymbol{z}_i^{(l)} = tanh(\boldsymbol{W}^{(l)} \boldsymbol{z}_i^{(l-1)} + \boldsymbol{b}^{(l)}), \qquad i = 1, \ldots, M, l = 1, \ldots, L$$

$$s = \sum_{i=1}^{M} g_i z_i^{(L)}$$

where $\otimes$ denotes the interaction operator between a query term and the document terms, h denotes the mapping function from local interactions to matching histogram, $\boldsymbol{z}_i^{(l)}, l = 0, \ldots, L$ denotes the intermediate hidden layers for the i-th query term, and $g_i, i = 1, \ldots, M$ denotes the aggregation weight produced by the term gating network. $\boldsymbol{W}^{(l)}$ denotes the l-th weight matrix and $\boldsymbol{b}^{(l)}$ denotes the l-th bias term, which are shared across different query terms. Note that we adopt cosine similarity, a widely used measure for semantic closeness in neural embeddings [18, 20], as the interaction operator between each pair of term vectors from a query and a document. In our work, we assume the term vectors are learned a priori using existing neural embedding models such as Word2Vec [18].

https://arxiv.org/abs/1711.08611

Term Gate Network $g_i = \dfrac{\exp(\boldsymbol{w}_g \boldsymbol{x}_i^{(q)})}{\sum_{j=1}^{M} \exp(\boldsymbol{w}_g \boldsymbol{x}_j^{(q)})}, \quad i = 1, \ldots, M,$

A Deep Relevance Matching Model for Ad-hoc Retrieval

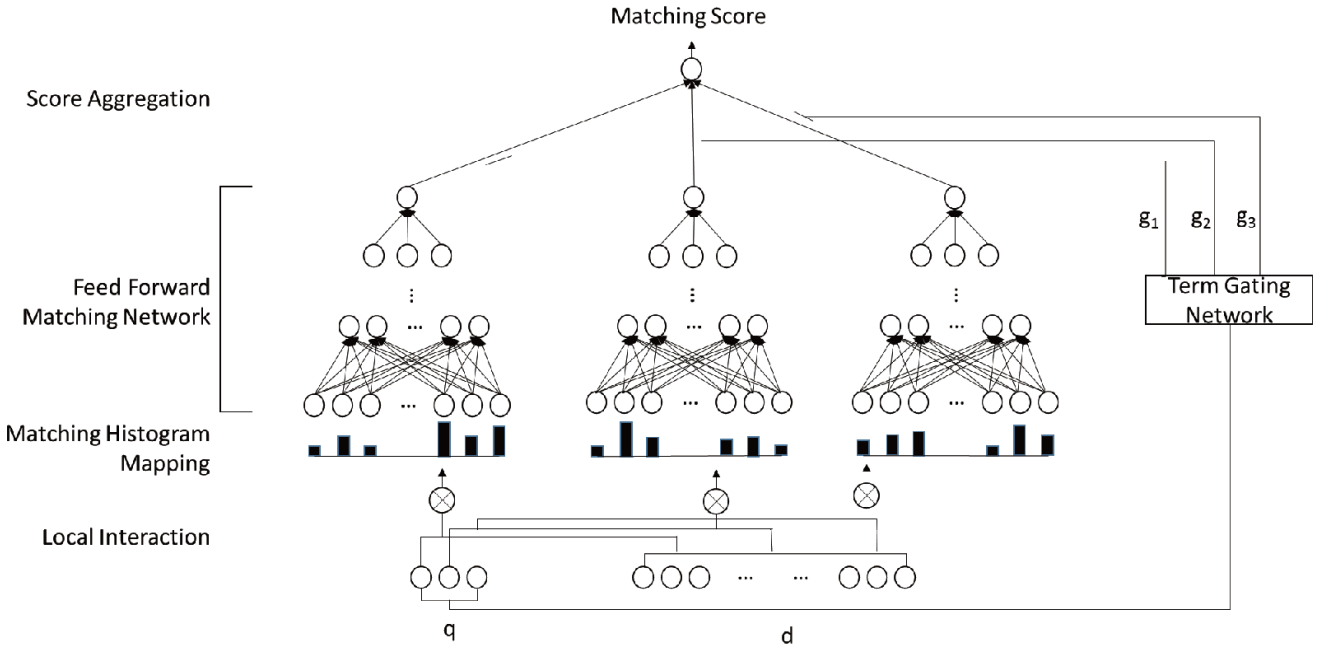


Figure 2: Architecture of the Deep Relevance Matching Model.

For example, suppose the bin size is set as 0.5, we will obtain five bins $\{[-1, -0.5), [-0.5, -0), [0, 0.5), [0.5, 1), [1, 1]\}$ in an ascending order. Given a query term "*car*" and a document $(car, rent, truck, bump, injunction, runway)$, and the corresponding local interactions based on cosine similarity are $(1, 0.2, 0.7, 0.3, -0.1, 0.1)$, we will obtain a matching histogram as $[0, 1, 3, 1, 1]$. We explore three ways of the matching histogram mapping:

Count-based Histogram (CH): This is the simplest way of transformation as described above which directly takes the count of local interactions in each bin as the histogram value.

Normalized Histogram (NH): We normalize the count value in each bin by the total count to focus on the relative rather than the absolute number of different levels of interactions.

LogCount-based Histogram (LCH): We apply logarithm over the count value in each bin, both to reduce the range, and to allow our model to more easily learn multiplicative relationships [1].

Term Gate Network $$g_i = \frac{\exp(\boldsymbol{w}_g \boldsymbol{x}_i^{(q)})}{\sum_{j=1}^{M} \exp(\boldsymbol{w}_g \boldsymbol{x}_j^{(q)})}, \quad i = 1, \ldots, M,$$

where $\boldsymbol{w}_g$ denotes the weight vector of the term gating network and $\boldsymbol{x}_i^{(q)}, i = 1, \ldots, M$ denotes the i-th query term input.

Query term input as (1) query term vectors or (2) IDF of the query terms

Result

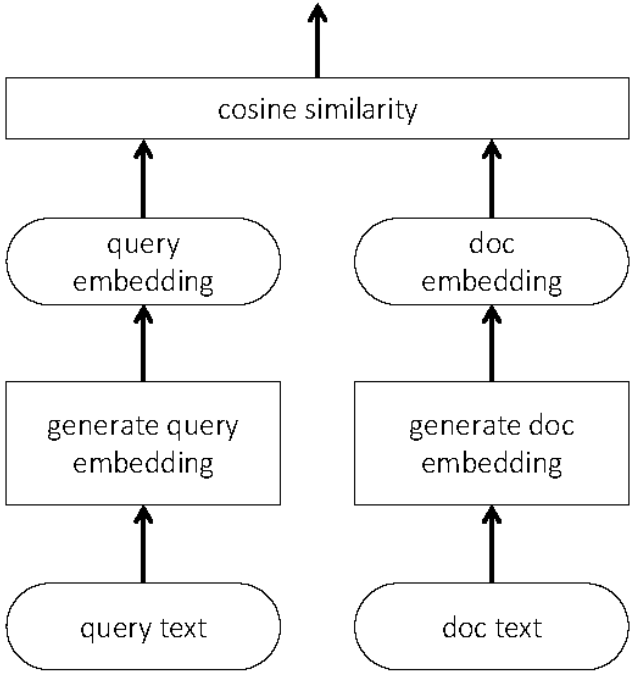
Table 2: Comparison of different retrieval models over the Robust-04 and ClueWeb-09-Cat-B collections. Significant improvement or degradation with respect to QL is indicated (+/-) (p-$value \leq 0.05$).

Robust-04 collection

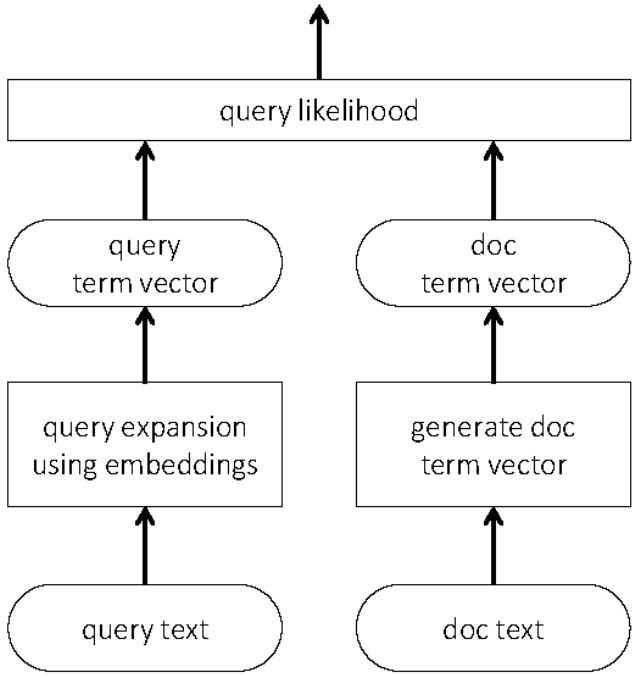
Model Type	Model Name	Topic titles			Topic descriptions		
		MAP	nDCG@20	P@20	MAP	nDCG@20	P@20
Traditional Retrieval Baselines	QL	0.253	0.415	0.369	0.246	0.391	0.334
	BM25	0.255	0.418	0.370	0.241	0.399	0.337
Representation-Focused Matching Baselines	$DSSM_D$	0.095^-	0.201^-	0.171^-	0.078^-	0.169^-	0.145^-
	$CDSSM_D$	0.067^-	0.146^-	0.125^-	0.050^-	0.113^-	0.093^-
	ARC-I	0.041^-	0.066^-	0.065^-	0.030^-	0.047^-	0.045^-
Interaction-Focused Matching Baselines	ARC-II	0.067^-	0.147^-	0.128^-	0.042^-	0.086^-	0.074^-
	MP_{IND}	0.169^-	0.319^-	0.281^-	0.067^-	0.142^-	0.118^-
	MP_{COS}	0.189^-	0.330^-	0.290^-	0.094^-	0.190^-	0.162^-
	MP_{DOT}	0.083^-	0.159^-	0.155^-	0.047^-	0.104^-	0.092^-
Our Approach	$DRMM_{CH\times TV}$	0.253	0.407	0.357	0.247	0.404	0.341
	$DRMM_{NH\times TV}$	0.160^-	0.293^-	0.258^-	0.132^-	0.217^-	0.186^-
	$DRMM_{LCH\times TV}$	0.268^+	0.423	0.381	0.265^+	0.423^+	0.360^+
	$DRMM_{CH\times IDF}$	0.259	0.412	0.362	0.255	0.410^+	0.344
	$DRMM_{NH\times IDF}$	0.187^-	0.312^-	0.282^-	0.145^-	0.243^-	0.199^-
	$DRMM_{LCH\times IDF}$	$\mathbf{0.279}^+$	$\mathbf{0.431}^+$	$\mathbf{0.382}^+$	$\mathbf{0.275}^+$	$\mathbf{0.437}^+$	$\mathbf{0.371}^+$

➔ Log count histogram (LCH) and IDF produced the best result.

Neural IR



(c) Learning query and document representations for matching (*e.g.*, (Huang *et al.*, 2013; Mitra *et al.*, 2016a))



(d) Query expansion using neural embeddings (*e.g.*, (Roy *et al.*, 2016; Diaz *et al.*, 2016))

- In (c) the focus is on learning effective representations of text using neural methods.
- Neural models can also be used to expand or augment the query before applying traditional IR techniques, as shown in (d).

A Dual Embedding Space Model for Document Ranking – example of Neural IR (c)

Table 1: The nearest neighbours for the words "yale", "seahawks" and "eminem" according to the cosine similarity based on the IN-IN, OUT-OUT and IN-OUT vector comparisons for the different words in the vocabulary. These examples show that IN-IN and OUT-OUT cosine similarities are high for words that are similar by function or type (*typical*), and the IN-OUT cosine similarities are high between words that often co-occur in the same query or document (*topical*). The *word2vec* model used here was trained on a query corpus with a vocabulary of 2,748,230 words.

yale			**seahawks**			**eminem**		
IN-IN	OUT-OUT	IN-OUT	IN-IN	OUT-OUT	IN-OUT	IN-IN	OUT-OUT	IN-OUT
yale	yale	yale	seahawks	seahawks	seahawks	eminem	eminem	eminem
harvard	uconn	faculty	49ers	broncos	highlights	rihanna	rihanna	rap
nyu	harvard	alumni	broncos	49ers	jerseys	ludacris	dre	featuring
cornell	tulane	orientation	packers	nfl	tshirts	kanye	kanye	tracklist
tulane	nyu	haven	nfl	packers	seattle	beyonce	beyonce	diss
tufts	tufts	graduate	steelers	steelers	hats	2pac	tupac	performs

https://arxiv.org/abs/1602.01137

Approach

$$DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{\mathbf{q}_i^T \overline{\mathbf{D}}}{\|\mathbf{q}_i\| \|\overline{\mathbf{D}}\|}, \tag{5}$$

where

$$\overline{\mathbf{D}} = \frac{1}{|D|} \sum_{\mathbf{d}_j \in D} \frac{\mathbf{d}_j}{\|\mathbf{d}_j\|} \tag{6}$$

Here $\overline{\mathbf{D}}$ is the centroid of all the normalized vectors for the words in the document serving as a single embedding for the whole document. In this formulation of the DESM, the document embeddings can be pre-computed, and at the time of ranking, we only need to sum the score contributions across the query terms. We expect that the ability to pre-compute a single document embedding is a very useful property when considering runtime efficiency.

Two variants:

$$DESM_{IN-OUT}(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_{IN,i}^T \overline{D_{OUT}}}{\|q_{IN,i}\| \|\overline{D_{OUT}}\|} \tag{7}$$

$$DESM_{IN-IN}(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_{IN,i}^T \overline{D_{IN}}}{\|q_{IN,i}\| \|\overline{D_{IN}}\|} \tag{8}$$

Result

- First get top candidate docs given a query from Bing search. Then rerank the top candidate docs by each of the below approaches:

Table 3: NDCG results comparing the $DESM_{IN-OUT}$ with the BM25 and the LSA baselines. The $DESM_{IN-OUT}$ performs significantly better than both the BM25 and the LSA baselines at all rank positions. It also performs better than the $DESM_{IN-IN}$ on both the evaluation sets. The DESMs using embeddings trained on the query corpus also performs better than if trained on document body text. The highest NDCG values for every column is highlighted in bold and all the statistically significant ($p < 0.05$) differences over the BM25 baseline are marked with the asterisk (*).

	Explicitly Judged Test Set			Implicit Feedback based Test Set		
	NDCG@1	NDCG@3	NDCG@10	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77	13.65	27.41	49.26
LSA	22.41*	28.25*	44.24*	16.35*	31.75*	52.05*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*	18.62*	33.80*	53.32*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*	18.37*	35.18*	54.20*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*	19.67*	35.53*	54.13*
DESM (IN-OUT, trained on queries)	**25.02***	**31.14***	**47.89***	**20.66***	**37.34***	**55.84***

Result

- Given a document collection, apply each of the below approaches.

Table 4: Results of NDCG evaluations under the non-telescoping settings. Both the DESM and the LSA models perform poorly in the presence of random irrelevant documents in the candidate set. The mixture of $DESM_{IN-OUT}$ with BM25 achieves the best NDCG. The best NDCG values are highlighted per column in bold and all the statistically significant ($p < 0.05$) differences with the BM25 baseline are indicated by the asterisk (*)

	Explicitly Judged Test Set			Implicit Feedback based Test Set		
	NDCG@1	NDCG@3	NDCG@10	NDCG@1	NDCG@3	NDCG@10
BM25	21.44	26.09	37.53	11.68	22.14	33.19
LSA	04.61*	04.63*	04.83*	01.97*	03.24*	04.54*
DESM (IN-IN, trained on body text)	06.69*	06.80*	07.39*	03.39*	05.09*	07.13*
DESM (IN-IN, trained on queries)	05.56*	05.59*	06.03*	02.62*	04.06*	05.92*
DESM (IN-OUT, trained on body text)	01.01*	01.16*	01.58*	00.78*	01.12*	02.07*
DESM (IN-OUT, trained on queries)	00.62*	00.58*	00.81*	00.29*	00.39*	01.36*
BM25 + DESM (IN-IN, trained on body text)	21.53	26.16	37.48	11.96	22.58*	33.70*
BM25 + DESM (IN-IN, trained on queries)	**21.58**	26.20	37.62	11.91	22.47*	33.72*
BM25 + DESM (IN-OUT, trained on body text)	21.47	26.18	37.55	11.83	22.42*	33.60*
BM25 + DESM (IN-OUT, trained on queries)	21.54	**26.42***	**37.86***	**12.22***	**22.96***	**34.11***

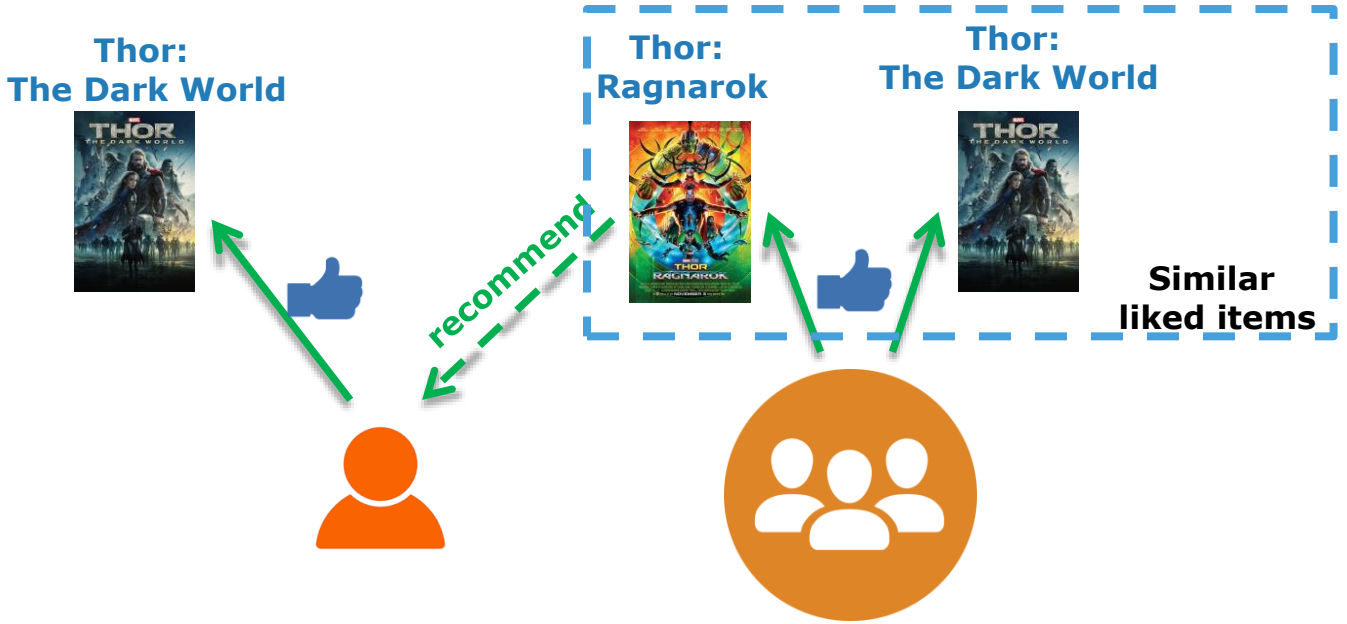
  - Solely using DESM would perform poorly. It produces too many false positives.
  - BM25 + DESM produces better results than BM25.

Regularizing Matrix Factorization with User and Item Embeddings for Recommendation

T. Tran, K. Lee, Y. Liao, and D. Lee. Regularizing Matrix Factorization with User and Item Embeddings for Recommendation. CIKM, 2018.

Three observations in Recommender Systems: (1/3)

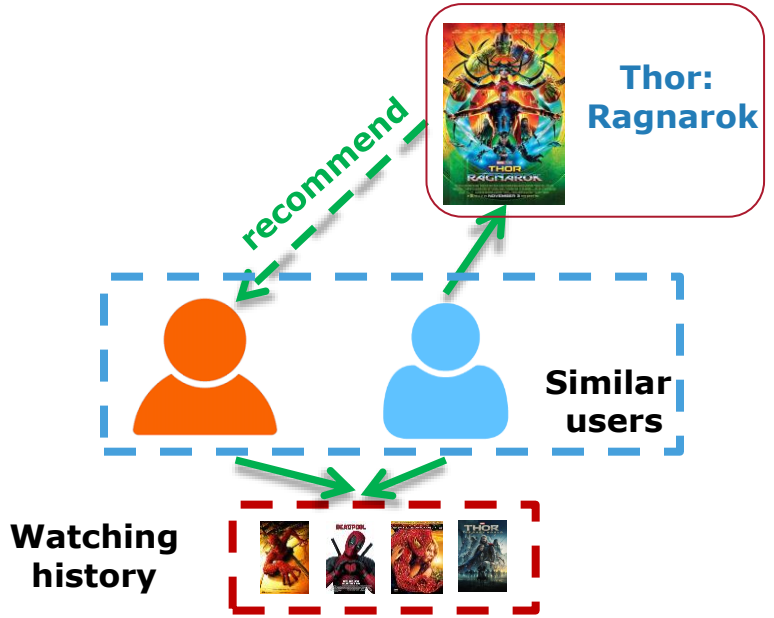
- **First observation**: recommend similar (co-liked) items to users.



Which pair of items is co-liked by how many people?

Three observations in Recommender Systems: (2/3)

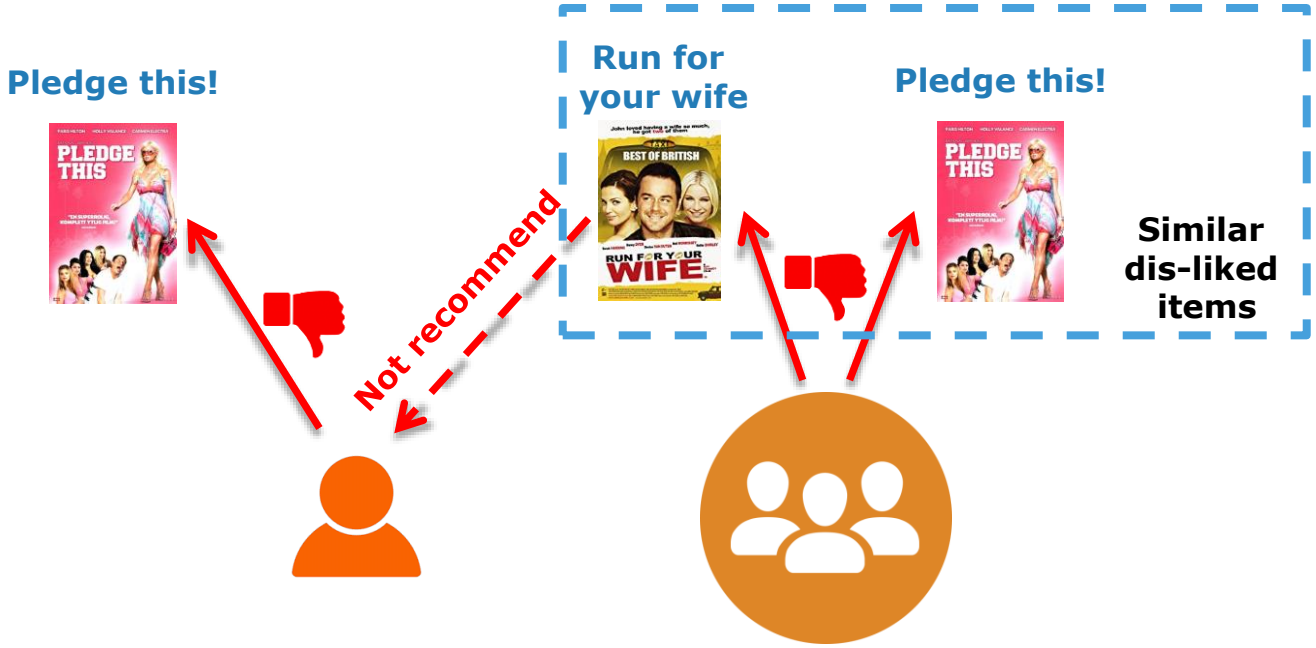
- **Second observation**: recommend same preferred items of a user to another similar user.



How many items does a pair of users liked together?

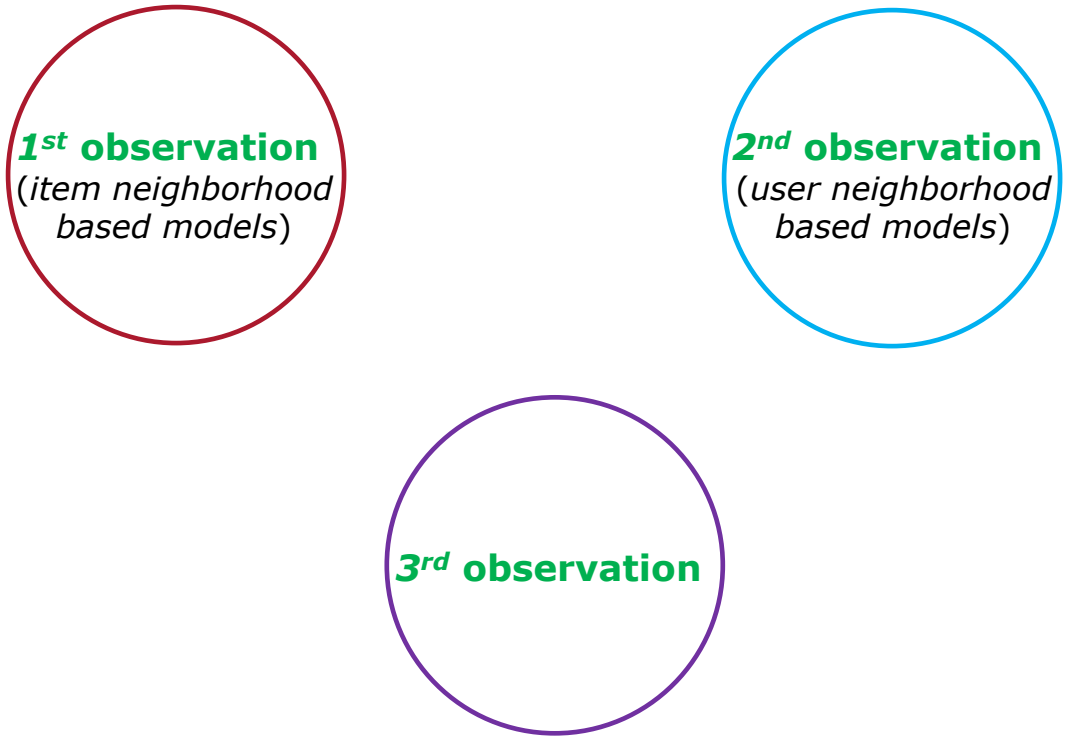
Three observations in Recommender Systems: (3/3)

- **Third observation**: not recommend similar disliked items to users.



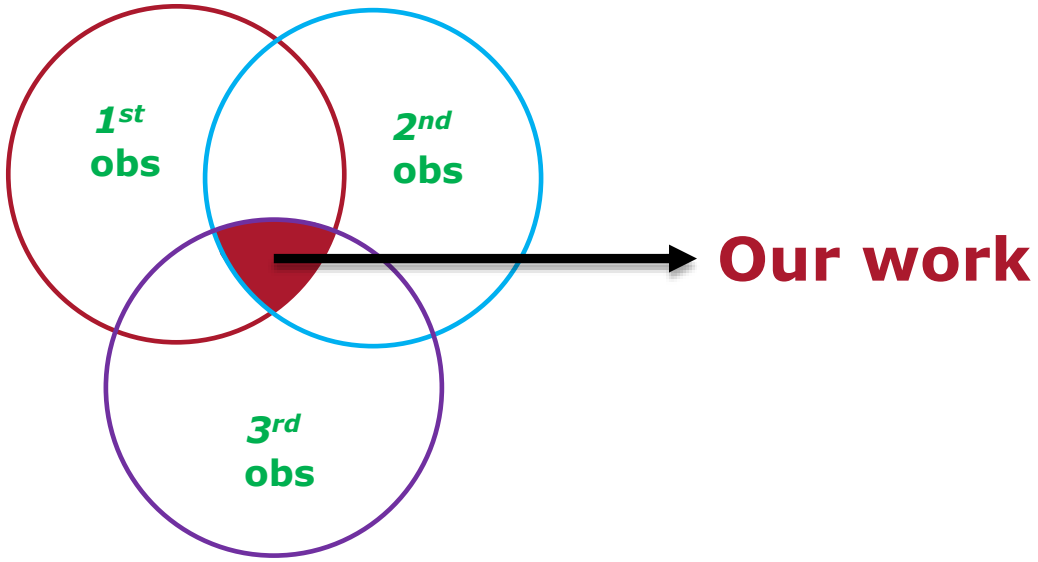
Which pair of items is co-disiked by how many people?

Our goal


1st observation
(item neighborhood based models)
2nd observation
(user neighborhood based models)
3rd observation

Our goal

- **Combine all:**



Research Questions

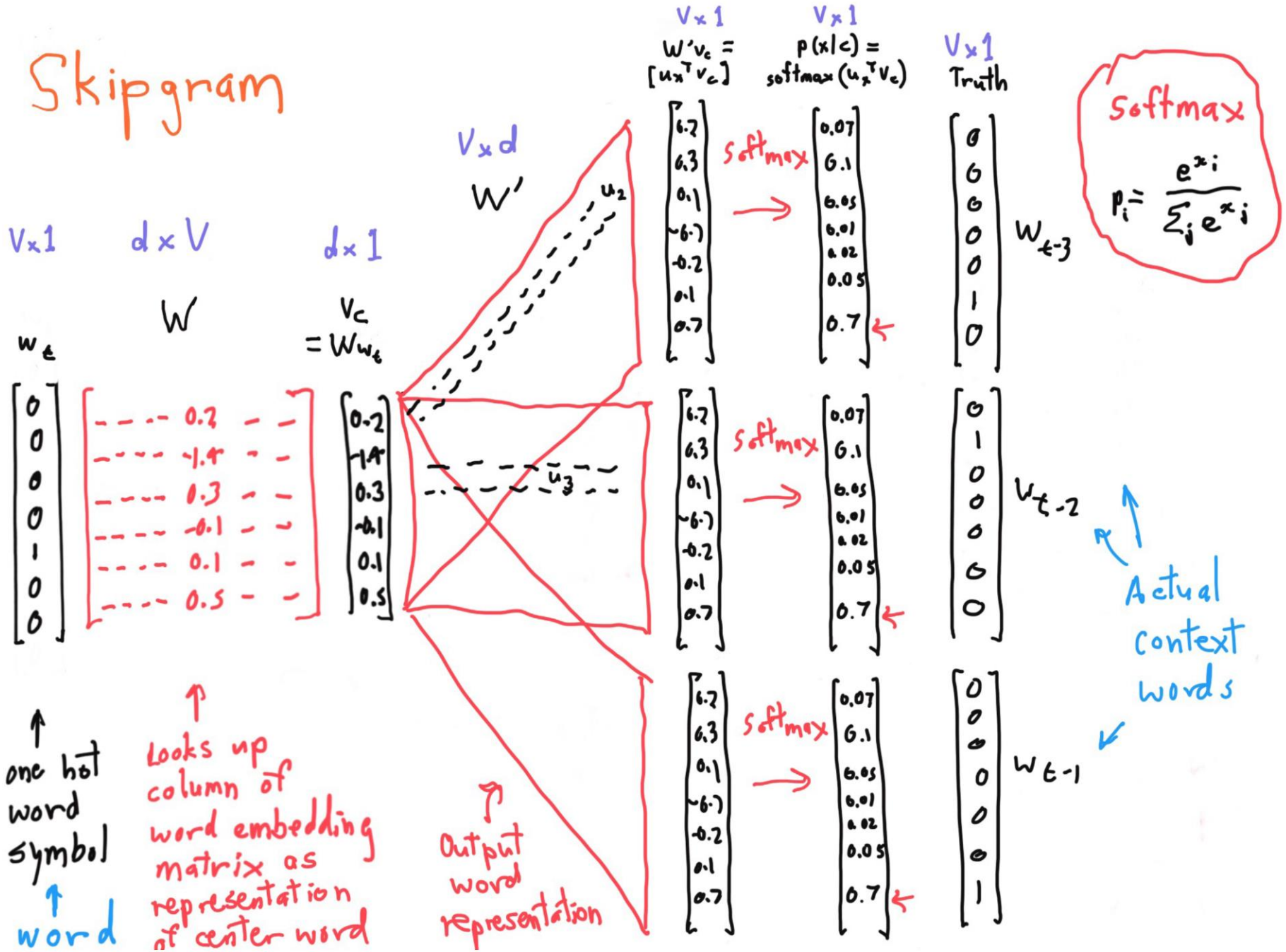
- **RS1:** How to exploit co-occurrence patterns of **users**, **co-liked items** and **co-disliked items**?
  - consider pairs of *co-occurred* **liked/disliked items** or pairs of *co-occurred* **users** as pairs of co-occurred words → apply word embedding technique.
- **RS2**: How to get disliked items in **implicit feedback** datasets?
  - Explicit feedback datasets: rating datasets (e.g. MovieLens), **visible disliked item** info.
  - Implicit feedback datasets: clicking datasets (e.g. TasteProfile), **no visible disliked item** info.

RQ1: Exploit co-occurrence patterns: skip-gram word2vec as implicit matrix factorization

- [Levy et al., 2014]: skip-gram word2vec is equivalent to implicitly factorizing a word-context matrix.

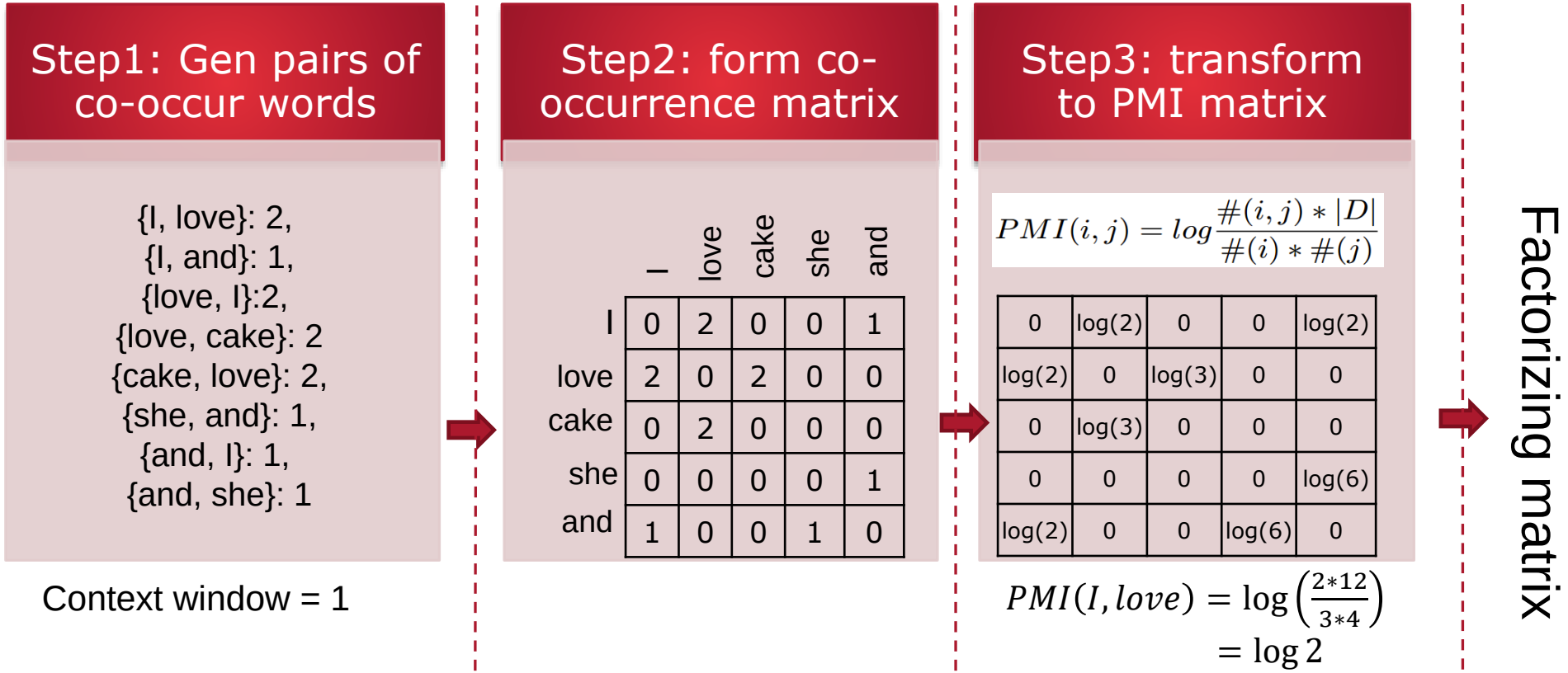
Neutral word embeddings as implicit matrix factorization, Levy & Goldberg, NIPS 2014

Skipgram


V x 1
W'v_c = [u_x^T v_c]
V x 1
p(x|c) = softmax(u_x^T v_c)
V x 1
Truth
V x d
W'
u_2
u_3
V x 1
d x V
d x 1
W
v_c = W w_t
w_t
Softmax
p_i = e^{x_i} / Σ_j e^{x_j}
softmax
softmax
softmax
W_{t-3}
W_{t-2}
W_{t-1}
Actual context words
one hot word symbol
word
Looks up column of word embedding matrix as representation of center word
Output word representation

RQ1: Exploit co-occurrence patterns: skip-gram word2vec as implicit matrix factorization

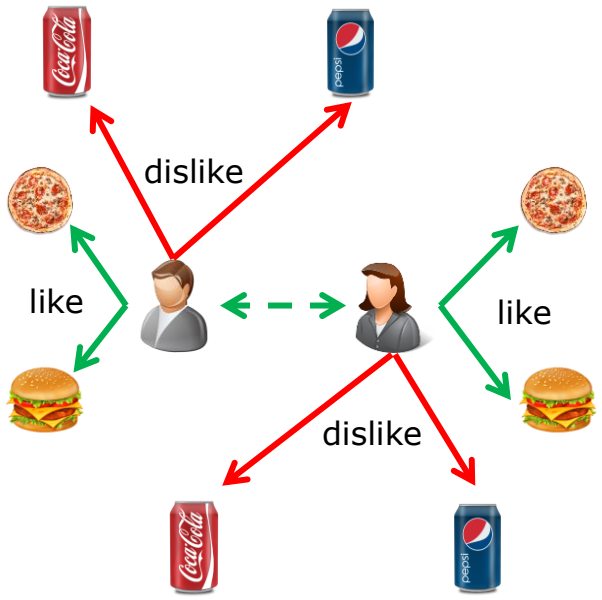
- Example: Given sentences: “I love cake”, “she and I love cake”.



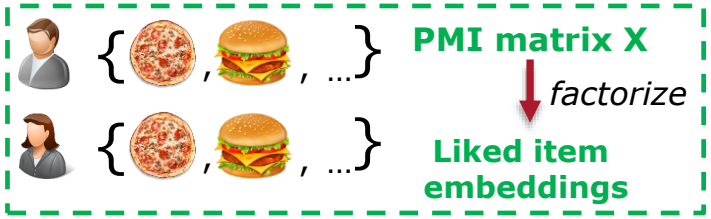
–	love	cake	she	and
I	0	2	0	0	1
love	2	0	2	0	0
cake	0	2	0	0	0
she	0	0	0	0	1
and	1	0	0	1	0

0	log(2)	0	0	log(2)
log(2)	0	log(3)	0	0
0	log(3)	0	0	0
0	0	0	0	log(6)
log(2)	0	0	log(6)	0

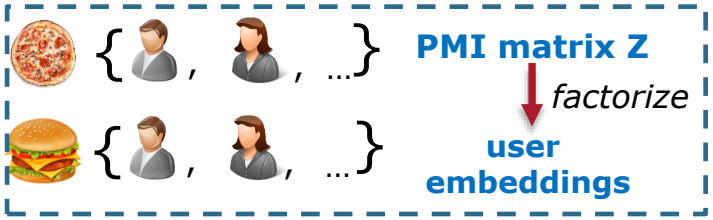
RQ1: Problem mapping



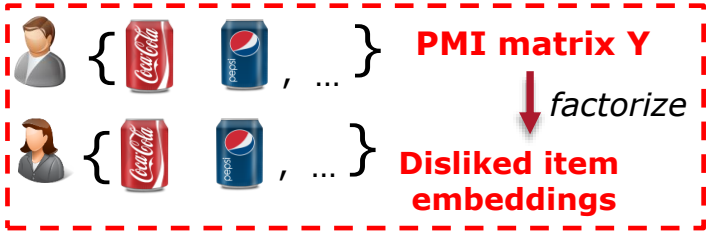
Which pair of items is co-liked by how many users?



How many items does a pair of users like together?



Which pair of items is co-disliked by how many users?



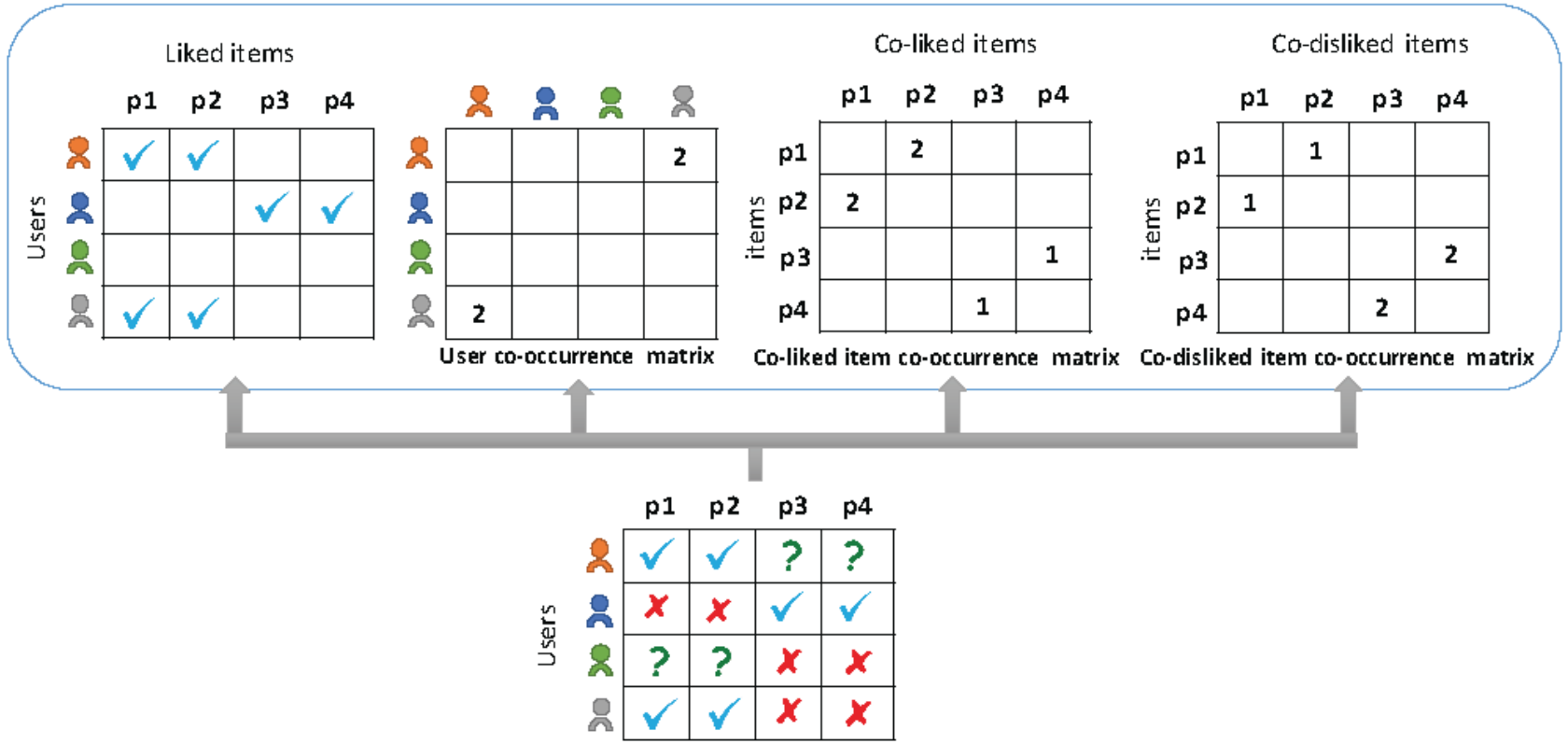


Figure 1: An overview of our RME Model, which jointly decomposes user-item interaction matrix, co-liked item co-occurrence matrix, co-disliked item co-occurrence matrix, and user co-occurrence matrix. (V: liked, X: disliked, and ?: unknown)

Our Regularized Multi-Embedding (RME) model

$$\text{minimize } \mathbf{L} = \frac{1}{2}\sum_{u,p} w_{up}\left(M_{up} - \alpha_u^T \beta_p\right)^2$$

→ Squared loss of weighted matrix factorization

$$+\frac{1}{2}\sum_{X_{pi}\neq 0} w_{pi}\left(X_{pi} - \beta_p^T \gamma_i - b_p - c_i\right)^2$$

→ Liked item embeddings (1st obs)

$$+\frac{1}{2}\sum_{Y_{pj}\neq 0} w_{pj}\left(Y_{pj} - \beta_p^T \delta_j - d_p - e_j\right)^2$$

→ Disliked item embeddings (3rd obs)

$$+\frac{1}{2}\sum_{Z_{uk}\neq 0} w_{uk}\left(Z_{uk} - \alpha_u^T \theta_k - f_u - g_k\right)^2$$

→ User embeddings (2nd obs)

$$+\frac{1}{2}\lambda\left(\sum_u ||\alpha_u||^2 + \sum_p ||\beta_p||^2 + \sum_i ||\gamma_i||^2 + \sum_j ||\delta_j||^2 \sum_k ||\theta_k||^2\right)$$

Regularization

Our Regularized Multi-Embedding (RME) model

$$\textit{minimize}\ \mathrm{L} = \frac{1}{2}\sum_{u,p} w_{up}\left(M_{up} - \alpha_u^T \beta_p\right)^2$$
$$+\frac{1}{2}\sum_{X_{pi}\neq 0} w_{pi}\left(X_{pi} - \beta_p^T \gamma_i - b_p - c_i\right)^2$$
$$+\frac{1}{2}\sum_{Y_{pj}\neq 0} w_{pj}\left(Y_{pj} - \beta_p^T \delta_j - d_p - e_j\right)^2$$
$$+\frac{1}{2}\sum_{Z_{uk}\neq 0} w_{uk}\left(Z_{uk} - \alpha_u^T \theta_k - f_u - g_k\right)^2$$
$$+\frac{1}{2}\lambda\left(\sum_u ||\alpha_u||^2 + \sum_p ||\beta_p||^2 + \sum_i ||\gamma_i||^2 + \sum_j ||\delta_j||^2 \sum_k ||\theta_k||^2\right)$$

Shared item embeddings

Shared users embeddings

RQ2: RME model for implicit feedback dataset

- **How to get disliked items in implicit feedback datasets?**
- Uniform sampling
  - *[He et al. 2017]* --> **not good.**
- Item popularity awareness
  - [He et al. 2016] → **not personalized**.

→ design a **user-oriented EM-like algorithm** to sample disliked items for users.

Neural Collaborative Filtering, *He et al.*, WWW 2017.
Fast Matrix Factorization for online recommendation with implicit feedback, *He et al.*, SIGIR 2016.

RQ2: RME model for implicit feedback dataset

- Initialization: Extract **U, P** by running *WMF*(**M**) in the training set

- Iterate:
  - **E step:**
    - Estimate a probability distribution of items to be disliked by each user *u*.
      - Preference of user u on all items: $\boldsymbol{r^{(u)} = \alpha_u^T P}$
      - Probability of unobserved item ***i*** to be sampled as disliked item:

$$P_i^{(u)} = \frac{exp(-r_i^{(u)})}{\sum_{j=1}^{n} exp(-r_j^{(u)})}$$

    - Then, sample disliked items (= 0.2 * number of positive interactions per user *u)*.

  - **M step:** build *RME(M)* model, and then extract **U, P** from the model

- Terminate the loop until RME does not produce better recommendation results than the previous model in the validation dataset.

Experiment: Data Sets

- MovieLens-10M: movie rating dataset (5-star rating dataset).
- MovieLens-20M: movie rating dataset (5-star rating dataset).
- TasteProfile: users and songs with play count.

	MovieLens-10M	MovieLens-20M	TasteProfile
#users	58,057	111,146	221,011
#items	7,223	9,888	22,713
density	0.978%	0.745%	0.291%

Experiment: Compared methods

Baselines:

- WMF [1]: weighted low rank matrix factorization.
- Item-KNN [2]:an item neighborhood-based CF method.
- Item2Vec [3]: used SGNS to learn item embeddings, then apply Item-KNN to generate item recommendations.
- Cofactor [4]: combined WMF + liked item embeddings. (= **RME** - *user embeddings – disliked item embeddings).*

Two variants:

- U_RME: a variant of our model, **RME without** *disliked item embeddings*.
- I_RME: another variant of our model, **RME without** *user embeddings.*

[1] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering forimplicit feedback datasets. ICDM 2008.
[2] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. TOIS, 2004
[**3]** Oren Barkan and Noam Koenigstein. Item2vec: neural item embedding for collaborative filtering. MLSP, 2016
[4] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factor-ization meets the item embedding: Regularizing matrix factorization with itemco-occurrence. RecSys 2016.

Experiment: Metrics for Evaluation

- Split data: follow 70/10/20 proportion.
- Metrics:
  - Recall@N: consider ranking of first N items equivalent.
  - NDCG@N: normalized discount cumulative gain.
  - MAP@N: mean of user’s average precision (AP).

Experimental Research questions

- **RQ1**: Performance of **RME** versus baselines and its variants?
- **RQ2**: Hyper-parameter sensitivity analysis?
- **RQ3**: **RME**'s performance for different types of users (cold-start, warm-start, highly-active).

Experimental Results

- **RQ1**: **Performance of RME versus compared models?**
- **RQ2**: Hyper-parameter sensitivity analysis?
- **RQ3**: RME's performance for different types of users (cold-start, warm-start, highly-active).

RQ1: Performance of RME versus baselines (1/2)

- **MovieLens-20M dataset:**

our model's variants | Our model

	Item-KNN	Item2Vec	WMF	Cofactor	**U_RME**	**I_RME**	**RME**
Recall@5	0.0131	0.1066	0.1348	0.1480	0.1524	0.1530	**0.1570**
NDCG@20	0.0345	0.1019	0.1290	0.1387	0.1425	0.1412	**0.1461**
MAP@10	0.0402	0.0539	0.0720	0.0804	0.0847	0.0838	**0.0869**

RME improved: +6.5% on average over the best baseline, +2.6% on average over its two variants.

RQ1: Performance of RME versus baselines? (2/2)

- **TasteProfile dataset:**

	Item-KNN	Item2Vec	WMF	Cofactor	our model's variants: U_RME	our model's variants: I_RME	Our model: RME
Recall@5	0.0793	0.1455	0.1745	0.1771	0.1825	0.1826	**0.1876**
NDCG@20	0.0685	0.1593	0.1853	0.1873	0.1899	0.1915	**0.1954**
MAP@10	0.0904	0.0727	0.0931	0.095	0.0997	0.0996	**0.1025**

RME improved:+6.0% on average over the best baseline, +2.5% on average over its two variants.

Addition: RME with Implicit feedback Inference in Movielens dataset

- **MovieLens-20M dataset:**

Our model

	Item-KNN	Item2Vec	WMF	Cofactor	RME	RME_implicit
Recall@5	0.0131	0.1066	0.1348	0.1480	**0.1570**	**0.1547**
NDCG@20	0.0345	0.1019	0.1290	0.1387	**0.1461**	**0.1457**
MAP@10	0.0402	0.0539	0.0720	0.0804	**0.0869**	**0.0861**

RME_implicit: +5.6% on average over the best baseline, -0.9% on average over RME (with explicit dislike info)

Experimental Results:RQ2

- **RQ1**: Performance of MCF versus baselines?
- **RQ2: Hyper-parameter sensitivity analysis?**
  - **RQ 2.1: Varying top-N recommendation list? (refer to our paper)**
  - **RQ 2.2: Varying latent size K? (refer to our paper)**
  - **RQ 2.3: Varying negative sample ratio? (refer to our paper)**
  - **RQ 2.4: Dynamic settings of regularization hyper-parameters? (refer to our paper)**
- **RQ3**: Performance of MCF for different types of users (cold-start, warm-start, highly-active).

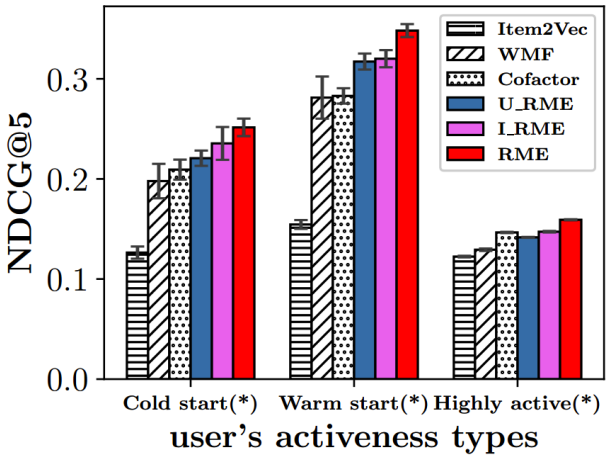
Experimental Results

- **RQ1**: Performance of RME versus baselines?
- **RQ2**: Hyper-parameter sensitivity analysis?
- **RQ3: RME's performance for different types of users (cold-start, warm-start, highly-active).**

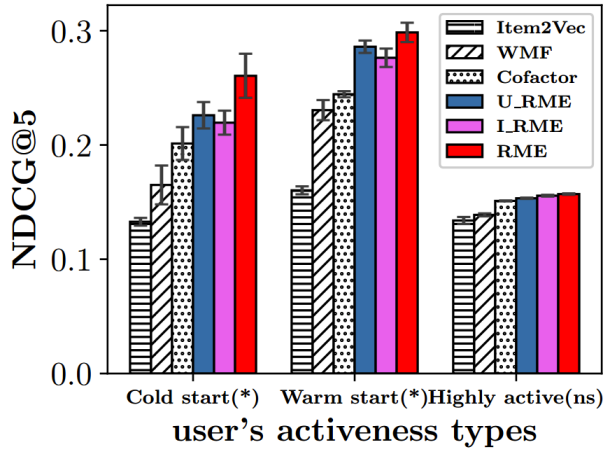
RQ3: Performance for different types of users

- Sort users by their activeness (e.g. #interactions) by ascending order.
- Cold-start: first 20%.
- Warm-start: 20%~80%.
- Highly active: last 20%.

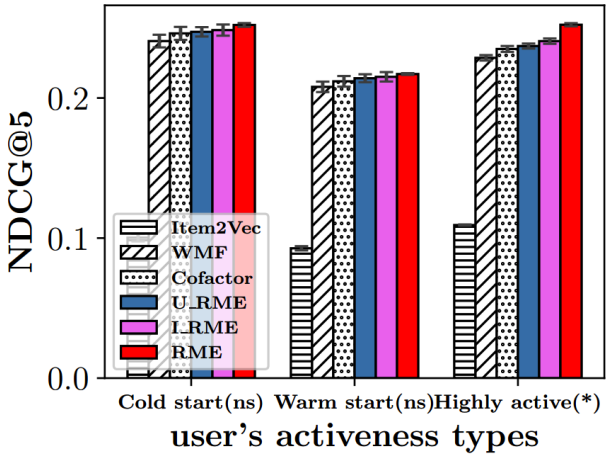
RQ3: Performance for different types of users



Movielens-10M



Movielens-20M



TasteProfile

On average, for cold-start users in two Movielens datasets: baseline: +24.8%; variants: +5.6%.

Summary

- Proposed a **joint model** combining WMF, co-liked item embedding, **co-disliked item** embedding and **user** embedding.
- Designed a **user-oriented EM-like** algorithm to sample **disliked items** for users.
- RME worked well on both explicit and implicit feedback datasets, **without using any additional info**: RME significantly improved **NDCG@20** by **6.6%** in all 3 datasets; RME improved **NDCG@5** by **24.8%** in two Movielens datasets for ***cold-start*** *users* group.

- Code: https://github.com/thanhdtran/RME.git