max planck institut
informatik

Advanced Topics in Information Retrieval

Learning to Rank

Vinay Setty
vsetty@mpi-inf.mpg.de

Jannik Strötgen
jannik.stroetgen@mpi-inf.mpg.de

ATIR – July 14, 2016

Before we start

oral exams

- July 28, the full day
- if you have any temporal constraints, let us know

Q-A sessions – suggestion

- Thursday, July 21: Vinay and “his topics”
- Monday, July 25: Jannik and “his topics”

max planck institut
informatik

Advanced Topics in Information Retrieval

Learning to Rank

Vinay Setty
vsetty@mpi-inf.mpg.de

Jannik Strötgen
jannik.stroetgen@mpi-inf.mpg.de

ATIR – July 14, 2016

The Beginning of LeToR

learning to rank (LeToR)

- builds on **established methods** from Machine Learning
- allows different targets derived from **different kinds of user input**
- active area of research for **past 10 – 15 years**
- early work already **(end of) 1980s** (e.g., Fuhr 1989)

The Beginning of LeToR

why wasn't LeToR successful earlier?

- IR and ML **communities were not very connected**
- sometimes **ideas take time**
- **limited training data**
  - it was **hard to gather** (real-world) **test collection queries** and **relevance judgments** that are representative of real user needs and **judgments on returned documents**
  - this changed in academia and industry
- **poor** machine learning **techniques**
- **insufficient customization** to IR problem
- **not enough features** for ML to show value

The Beginning of LeToR

standard ranking based on

- term frequency / inverse document frequency
- Okapi BM25
- language models
- ...

traditional ranking functions in IR exploit **very few features**

standard approach to combine different features

- **normalize features** (zero mean, unit standard deviation)
- **feature combination function** (typically: weighted sum)
- **tune weights** (either manually or exhaustively via grid search)

traditional ranking functions **easy to tune**

Why **learning to rank** nowadays?

Why learning to rank?

modern systems use a **huge number of features**
(especially Web search engines)

- **textual relevance** (e.g., using LM, Okapi BM25)
- **proximity** of query keywords in document content
- **link-based importance** (e.g., determined using PageRank)
- **depth of URL** (top-level page vs. leaf page)
- **spamminess** (e.g., determine using SpamRank)
- **host importance** (e.g., determined using host-level PageRank)
- **readability** of content
- **location** and **time** of the user
- **location** and **time** of documents
- ...

Why learning to rank?

high creativity in **feature engineering** task

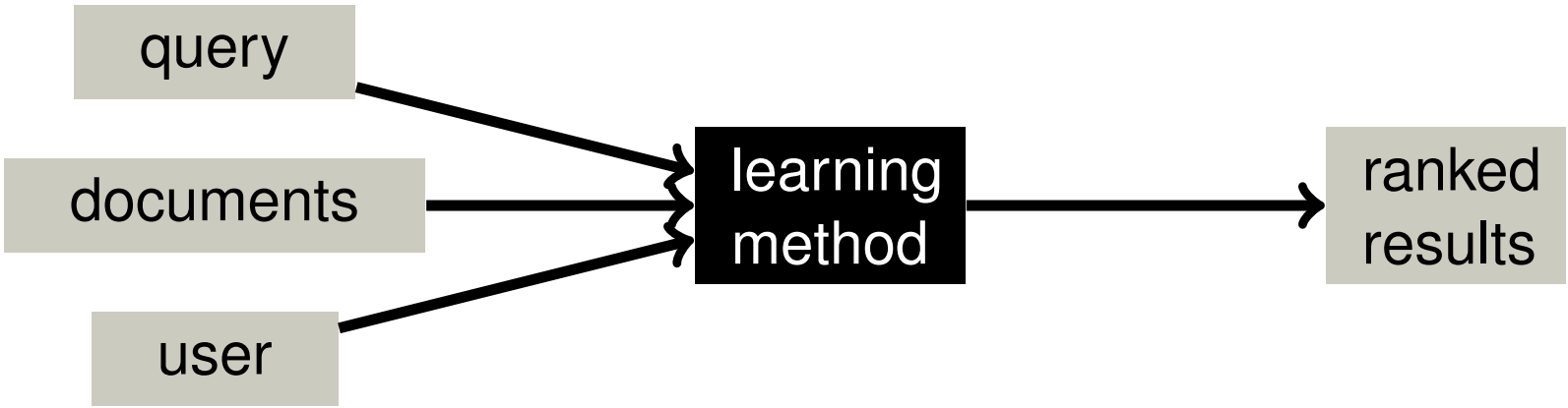
- query word in **color** on page?
- number of **images** on page?
- URL contains ~?
- number of (out) **links** on a page?
- page **edit recency**
- **page length**

learning to rank makes combining features more systematic

Outline I

Outline

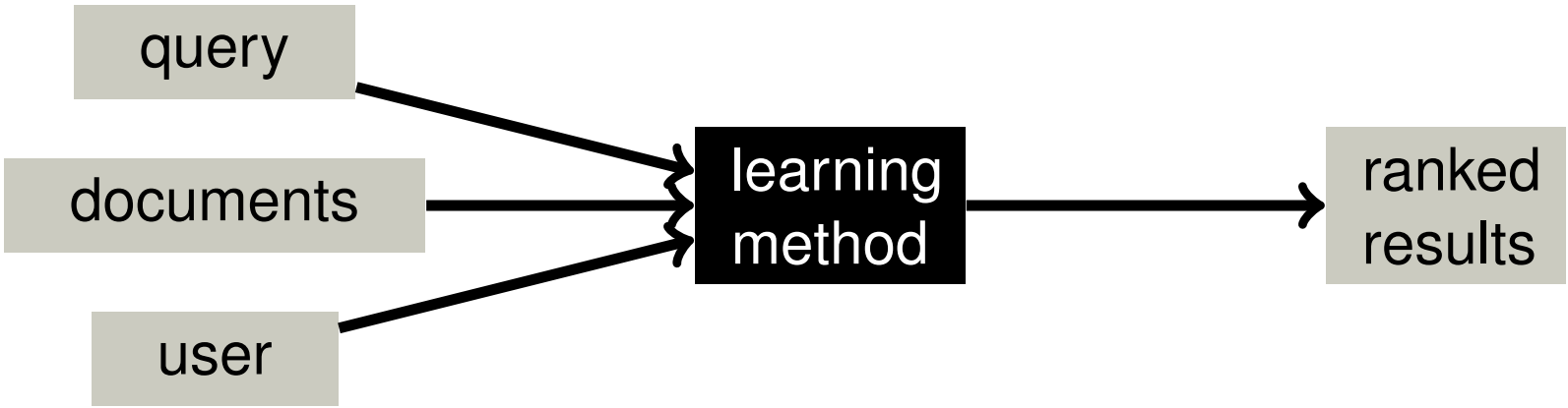
LeToR Framework



open issues

- how do we **model** the problem?
- is it a **regression or classification** problem?
- what about our **prediction target**?

LeToR Framework



scoring as function

- different input signals (features) x_i
- with weights α_i

$$\text{score(d,q)} = f(x_1, ..., x_m, \alpha_1, ..., \alpha_m)$$

where

- weights are learned
- features derived from d, q, and context

Outline

Classification – Regression

classification example

- dataset of $\langle q, d, r \rangle$ triples
- r: relevance (binary or multiclass)
- d: document represented by feature vector
- train ML model to predict class r of a d-q pair
- decide relevant if score is above threshold

classification problems

result in an unordered set of classes

Classification – Regression

classification problems

result in an unordered set of classes

regression problems

map to real values

ordinal regression problems

result in ordered set of classes

LeToR Modeling

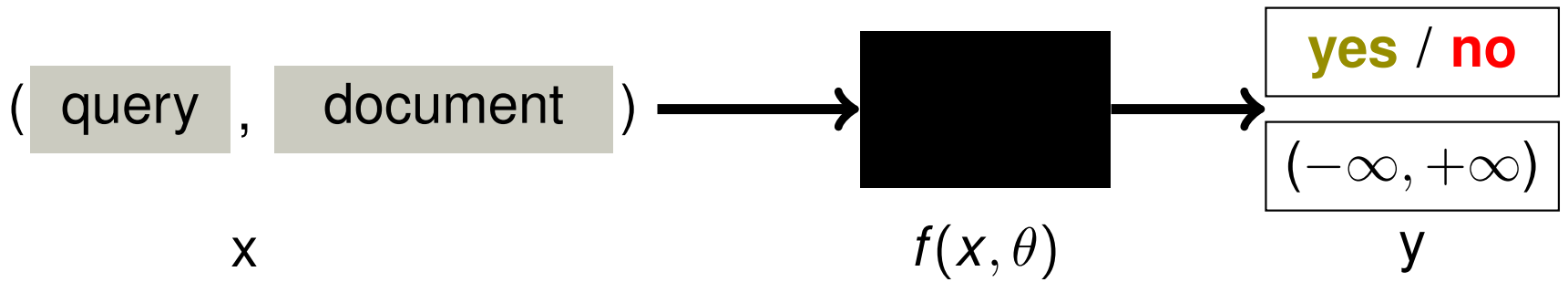
LeToR can be modeled in three ways:

- pointwise: predict goodness of **individual documents**
- pairwise: predict users' relative preference for **pairs of documents**
- listwise: predict goodness of **entire query results**

- each has advantages and disadvantages
- for each concrete approaches exist

in-depth discussion of concrete approaches by Liu 2009

Pointwise Modeling



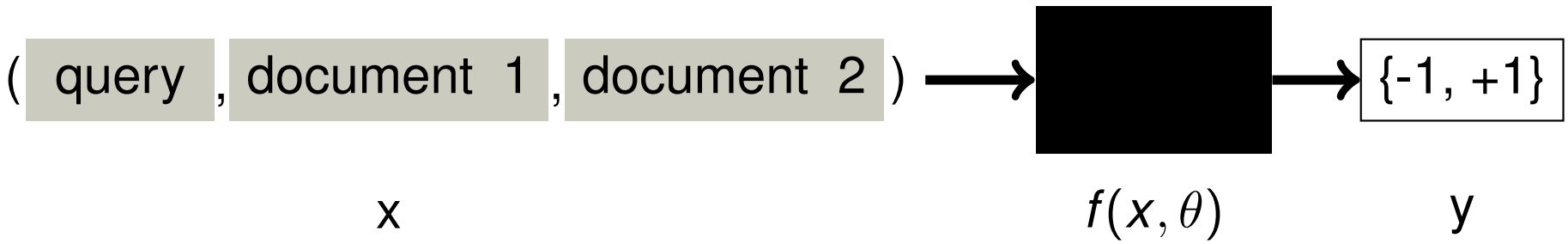
pointwise approaches

- predict for **every document** based on its feature vector **x**
- document **goodness y** (e.g., label or measure of engagement)
- training determines the **parameter** θ based on a **loss function** (e.g., root-mean-square error)

main disadvantage

- as input is single document, relative order between documents cannot be naturally considered in the learning process

Pairwise Modeling



pairwise approaches

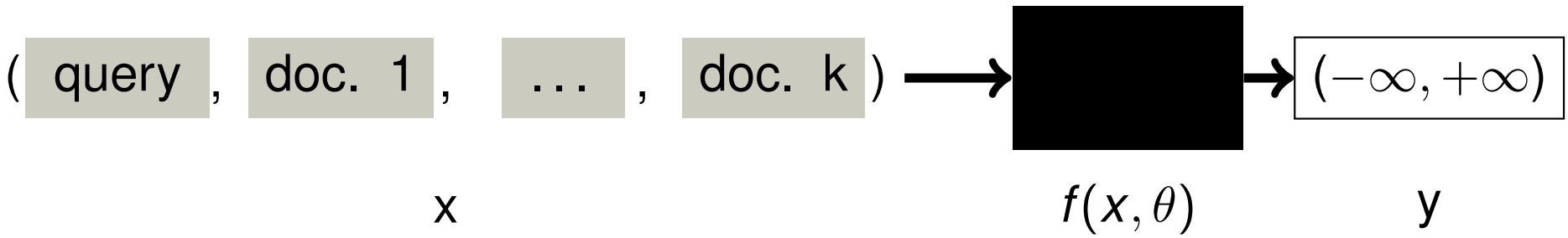
- predict for every **pair of documents** based on feature vector **x**
- users' **relative preference** regarding the documents (**+1** shows preference for document 1; **-1** for document 2)
- training determines the **parameter** θ based on a **loss function** (e.g., the number of inverted pairs)

advantage: models relative order

main disadvantages:

- no distinction between excellent–bad and fair–bad
- sensitive to noisy labels (1 wrong label, many mislabeled pairs)

Listwise Modeling



listwise approaches

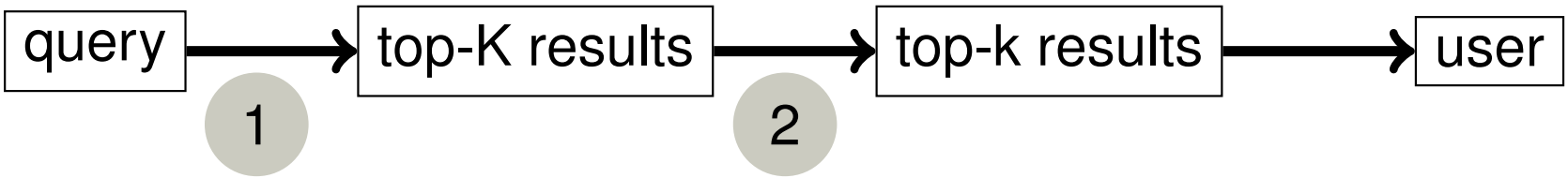
- predict for **ranked list of documents** based on feature vector **x**
- **effectiveness** of ranked list **y** (e.g., MAP or nDCG)
- training determines the **parameter** θ based on a **loss function**

advantage: positional information visible to loss function
disadvantage: high training complexity, ...

Typical Learning-to-Rank Pipeline

learning to rank

- is typically deployed as a **re-ranking step** (infeasible to apply it to entire document collection)



step 1

- Determine a top-K result (K ~1,000) using a proven baseline retrieval method (e.g., Okapi BM25 + PageRank)

step 2

- Re-rank documents from top-K using learning to rank approach, then return top-k (k ~100) to user

Outline

Gathering User Feedback

independent of pointwise, pairwise, or listwise modeling

some **input from the user is required**
to determine prediction target **y**

two types of user input

- **explicit** user input (e.g., relevance assessments)
- **implicit** user input (e.g., by analyzing their behavior)

Relevance Assessments

procedure

- construct a **collection of (difficult) queries**
- pool results from different baselines
- gather **graded relevance assessments** from human assessors

problems

- **hard to represent query workload** within 50, 500, 5K queries
- difficult for queries that require **personalization** or **localization**
- expensive, time-consuming, and subject to Web dynamics

Clicks

track user behavior and measure their **engagement** with results

- **click-through rate** of document when shown for query
- **dwell time**, i.e., how much time user spent on document

problems

- **position bias** (consider only first result shown)
- **spurious clicks** (consider only clicks with dwell time above threshold)
- **feedback loop** (add some randomness to results)

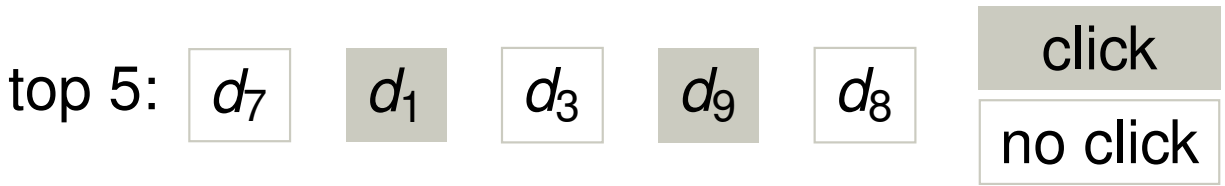
reliability of click data

Joachims et al. (2007) and Radlinski & Joachims (2005)

Skips

user behavior tells us more:

- **skips in addition to clicks** as a source of **implicit feedback**



- **skip previous**: $d_1 > d_7$ and $d_9 > d_3$ (user prefers d_1 over d_7)
- **skip above**: $d_1 > d_7$ and $d_9 > d_3$, $d_9 > d_7$

user study (Joachims et al., 2007): **derived relative preferences**

- are **less biased** than measures merely based on clicks
- show **moderate agreement** with explicit relevance assessments

Outline

Learning to Rank – Evaluation

Several **benchmark datasets** have been released to allow for a comparison of different learning-to-rank methods

LETOR 2.0, 3.0, 4.0 (2007–2009) by Microsoft Research Asia

- based on publicly available document collections
- come with precomputed low-level features and relevance assessments

Yahoo! Learning to Rank Challenge (2010) by Yahoo! Labs

- comes with precomputed low-level features and relevance assessments

Microsoft Learning to Rank Datasets by Microsoft Research U.S.

- comes with precomputed low-level features and relevance assessments

Features

Yahoo! Features

queries, ulrs and features descriptions are not given, only the feature values!

- feature engineering critical for any commercial search engine
- releasing queries, URLs leads to risk of reverse engineering
- reasonable consideration, but prevent IR researchers from studying what feature are most effective

LETOR / Microsoft Features

each query-url pair is represented by a 136-dimensional vector

LETOR Features

feature id	feature description	stream	comments
1	covered query term number	body	
2		anchor	
3		title	
4		url	
5		whole document	
6	covered query term ratio	body	
7		anchor	
8		title	
9		url	
10		whole document	
11	stream length	body	
12		anchor	
13		title	
14		url	
15		whole document	

LETOR Features

16	IDF(Inverse document frequency)	body
17		anchor
18		title
19		url
20		whole document
21	sum of term frequency	body
22		anchor
23		title
24		url
25		whole document
26	min of term frequency	body
27		anchor
28		title
29		url
30		whole document

LETOR Features

31	max of term frequency	body
32		anchor
33		title
34		url
35		whole document
36	mean of term frequency	body
37		anchor
38		title
39		url
40		whole document
41	variance of term frequency	body
42		anchor
43		title
44		url
45		whole document

LETOR Features

46	sum of stream length normalized term frequency	body
47		anchor
48		title
49		url
50		whole document
51	min of stream length normalized term frequency	body
52		anchor
53		title
54		url
55		whole document
56	max of stream length normalized term frequency	body
57		anchor
58		title
59		url
60		whole document

LETOR Features

61	mean of stream length	body
62	normalized term frequency	anchor
63		title
64		url
65		whole document
66	variance of stream length	body
67	normalized term frequency	anchor
68		title
69		url
70		whole document
71	sum of tf*idf	body
72		anchor
73		title
74		url
75		whole document

LETOR Features

76	min of tf*idf	body
77		anchor
78		title
79		url
80		whole document
81	max of tf*idf	body
82		anchor
83		title
84		url
85		whole document
86	mean of tf*idf	body
87		anchor
88		title
89		url
90		whole document

LETOR Features

91	variance of tf*idf	body
92		anchor
93		title
94		url
95		whole document
96	boolean model	body
97		anchor
98		title
99		url
100		whole document
101	vector space model	body
102		anchor
103		title
104		url
105		whole document

LETOR Features

106	BM25	body	
107		anchor	
108		title	
109		url	
110		whole document	
111	LMIR.ABS	body	Language model approach for information retrieval (IR) with absolute discounting smoothing
112		anchor	
113		title	
114		url	
115		whole document	
116	LMIR.DIR	body	Language model approach for IR with Bayesian smoothing using Dirichlet priors
117		anchor	
118		title	
119		url	
120		whole document	

LETOR Features

121	LMIR.JM	body	Language model approach for IR with Jelinek-Mercer smoothing
122		anchor	
123		title	
124		url	
125		whole document	
126	Number of slash in URL		
127	Length of URL		
128	Inlink number		
129	Outlink number		
130	PageRank		
131	SiteRank		Site level PageRank
132	QualityScore		The quality score of a web page. The score is outputted by a web page quality classifier.

LETOR Features

133	QualityScore2	The quality score of a web page. The score is outputted by a web page quality classifier, which measures the badness of a web page.
134	Query-url click count	The click count of a query-url pair at a search engine in a period
135	url click count	The click count of a url aggregated from user browsing data in a period
136	url dwell time	The average dwell time of a url aggregated from user browsing data in a period

Learning to Rank – Starting Point

all details

http://research.microsoft.com/en-us/um/beijing/projects/letor/

https://www.microsoft.com/en-us/research/project/mslr/

- datasets
- dataset descriptions
- partitioned in subsets (for cross-validation)
- evaluation scripts, significance test scripts
- feature list

everything required to get started is available

Outline

Learning-to-Rank for Temporal IR

Kanhabua & Nørvåg (2012)

learning-to-rank approach for time-sensitive queries

standard temporal IR approaches

- mixture model linearly combining textual similarity and temporal similarity
- probabilistic model generating a query from the textual and temporal part of document independent

learning-to-rank approach

- two classes of features: entity features and time features
- both derived from annotations (NER, temporal tagging)

Learning-to-Rank for Temporal IR

document model

- a document collection over time
- document is composed of a bag of words and time
  - publication date
  - temporal expressions mentioned in document
- annotated document composed of
  - set of named entities
  - set of temporal expressions
  - set of annotated sentences

temporal query model

- $q = \{q_{text}, q_{time}\}$
- q_{time} might be explicit or implicit

Learning-to-Rank for Temporal IR

learning-to-rank

- a wide range of **temporal features**
- a wide range of **entity features**
- models trained using **labeled query/document pairs**
- documents ranked according to weighted sum of its feature scores

experiments

- show improvement of baselines and other time-aware models
- (many queries also contained entities, news corpus)

Outline

Learning-to-Rank – Beyond Search

learning to rank is applicable beyond web search

Example: **matching in eharmony.com**

Slides by Vaclav Petricek:

`http://www.slideshare.net/VaclavPetricek/data-science-of-love`

basic idea:

- standard approach is search-based, filter out non-matches
- eharmony approach is learning to rank: suggest potential matches

Matching in eHarmony.com

starting point in the 1990s

distinguish marriages that work well and those that don't

step 1: **compatibility matching**

- based on 150 questions: personality, values, attitudes, beliefs
  important attributes for the long term
- **predict marital satisfaction**

even if people are compatible,

they might not be interested to talk to each other

Matching in eHarmony.com

step 2: **affinity matching**

- based on other features: distance, height difference, zoom level of photo
- **predict probability of message exchange**

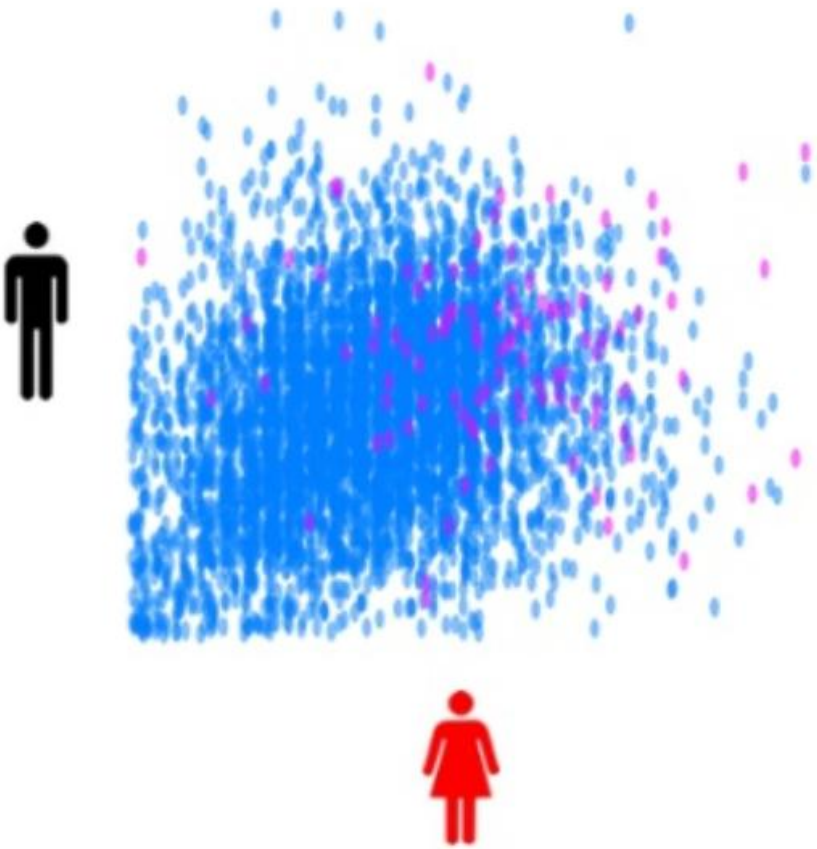
however

who should be introduced to whom and when?

match distribution

- based on graph optimization problem (constrained max flow)

Compatibility Matching › Obstreperousness



ob·strep·er·ous

/əbˈstrepərəs/

Adjective

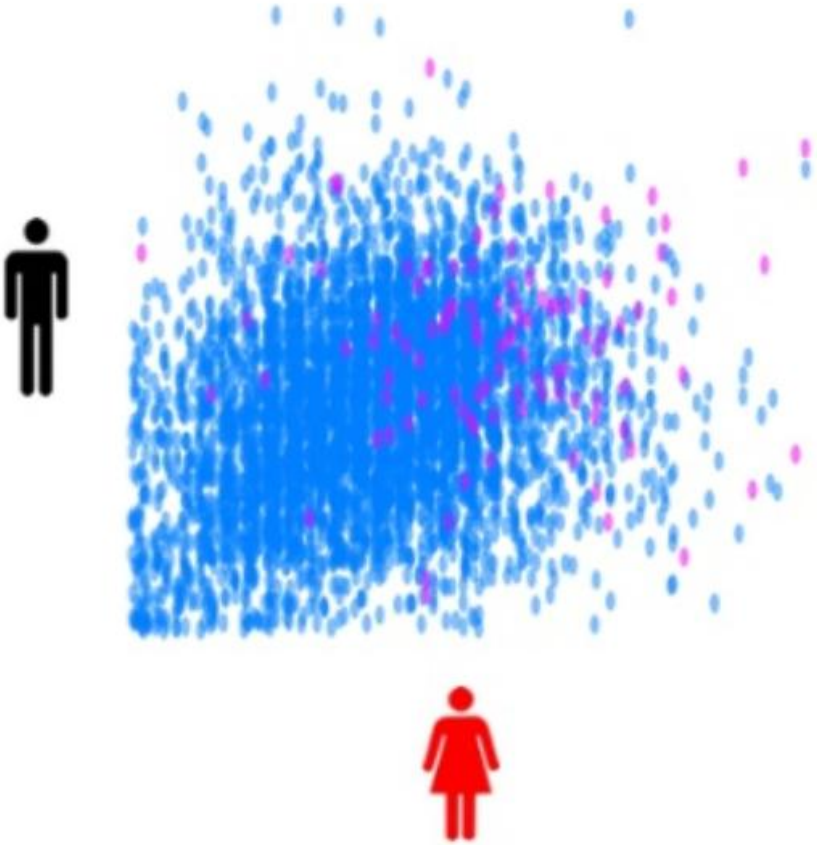
Noisy and difficult to control: "the boy is cocky and obstreperous".

Synonyms

noisy - loud - clamorous - rumbustious - boisterous

Compatibility Matching › Obstreperousness

blue: happy marriages
red: distressed marriages



is that person arguing with anything you say?

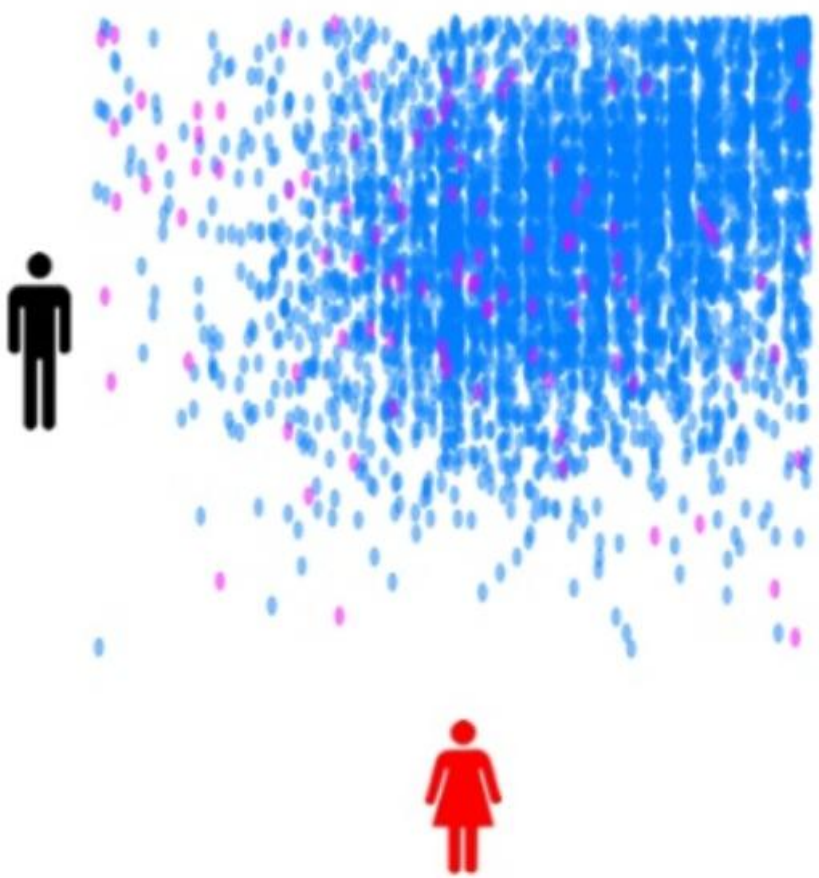
ob·strep·er·ous
/əbˈstrepərəs/

Adjective
Noisy and difficult to control: "the boy is cocky and obstreperous".

Synonyms
noisy - loud - clamorous - rumbustious - boisterous

relation between obstreperousness and marriage happiness

Compatibility Matching › Romantic



ro·man·tic
/rōˈmantik/

Adjective
Inclined toward or suggestive of the feeling of excitement and mystery associated with love.

Noun
A person with romantic beliefs or attitudes.

Synonyms
romanticist

Matching in eHarmony.com

even if people are compatible,

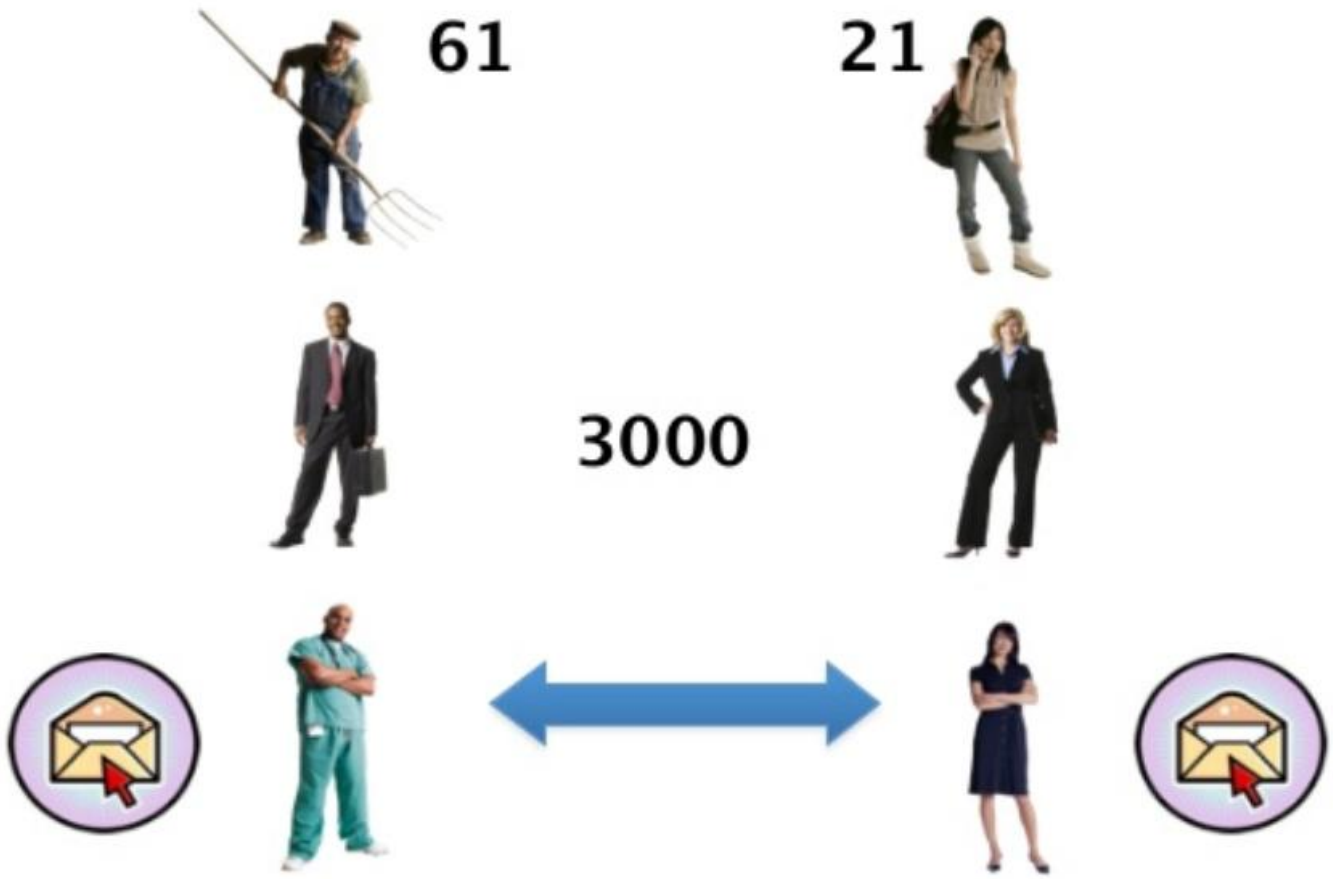
they might not be interested to talk to each other

Affinity Matching ›

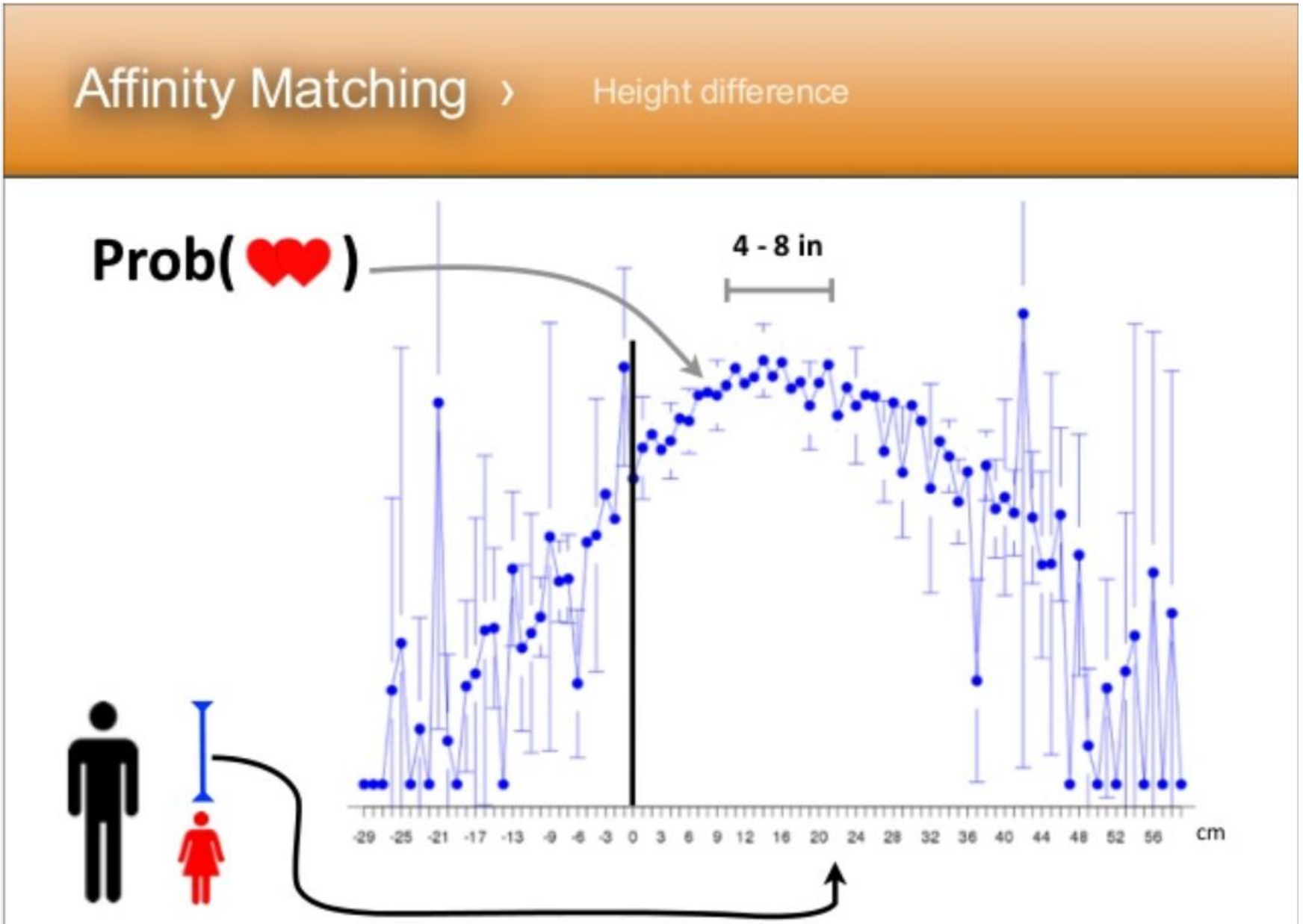




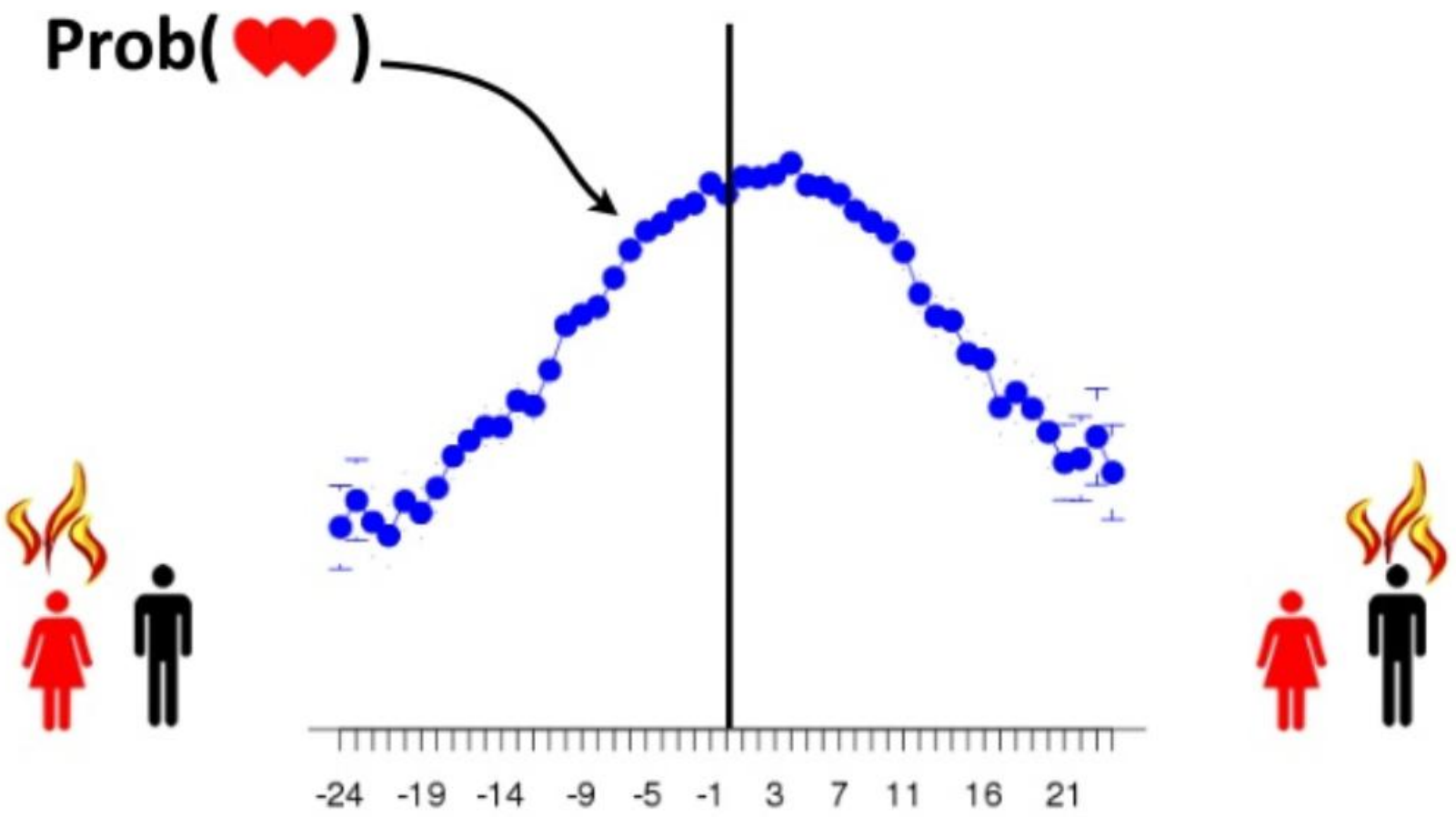

Affinity Matching ›

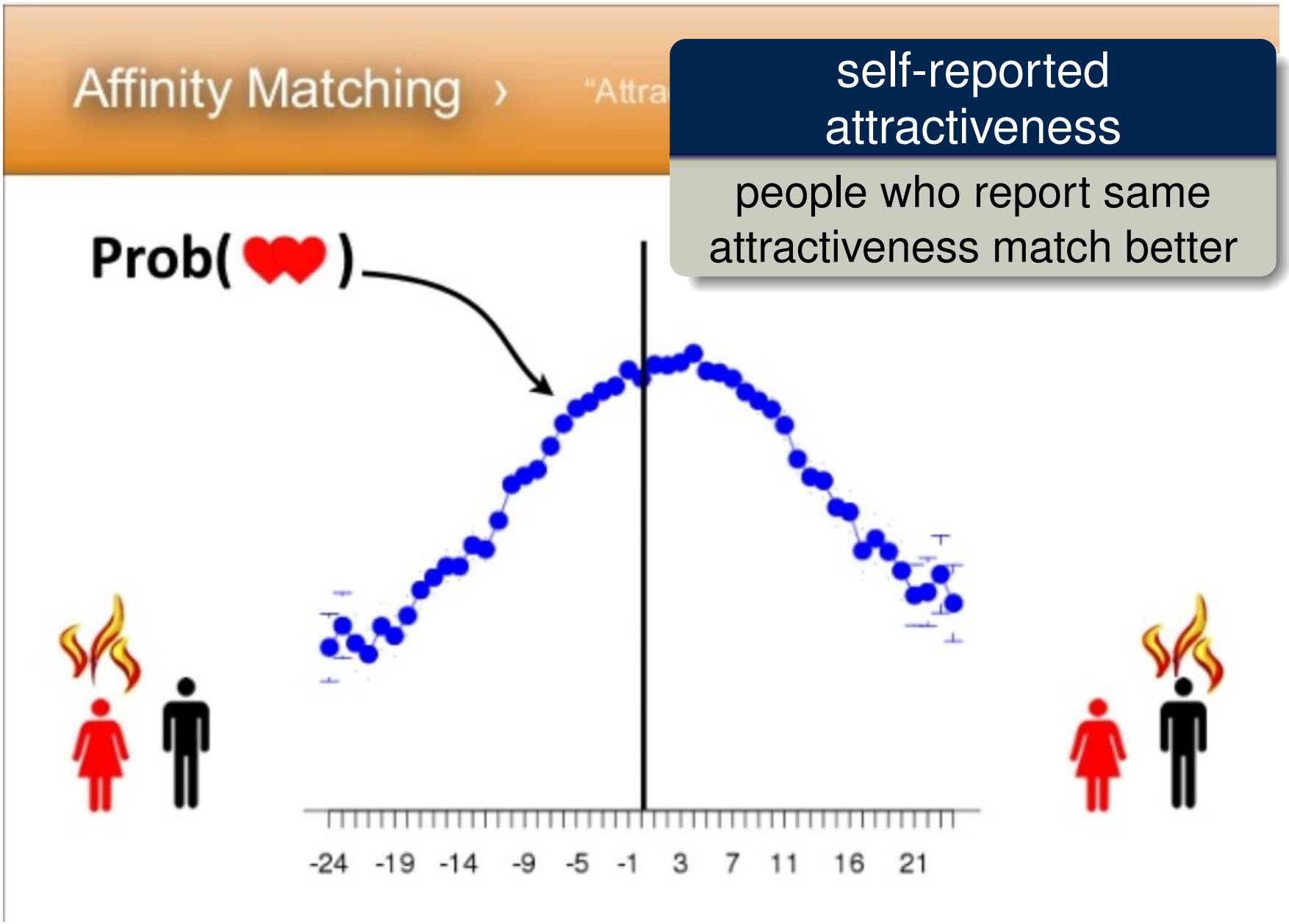


Affinity Matching › Height difference



Affinity Matching › "Attractiveness"



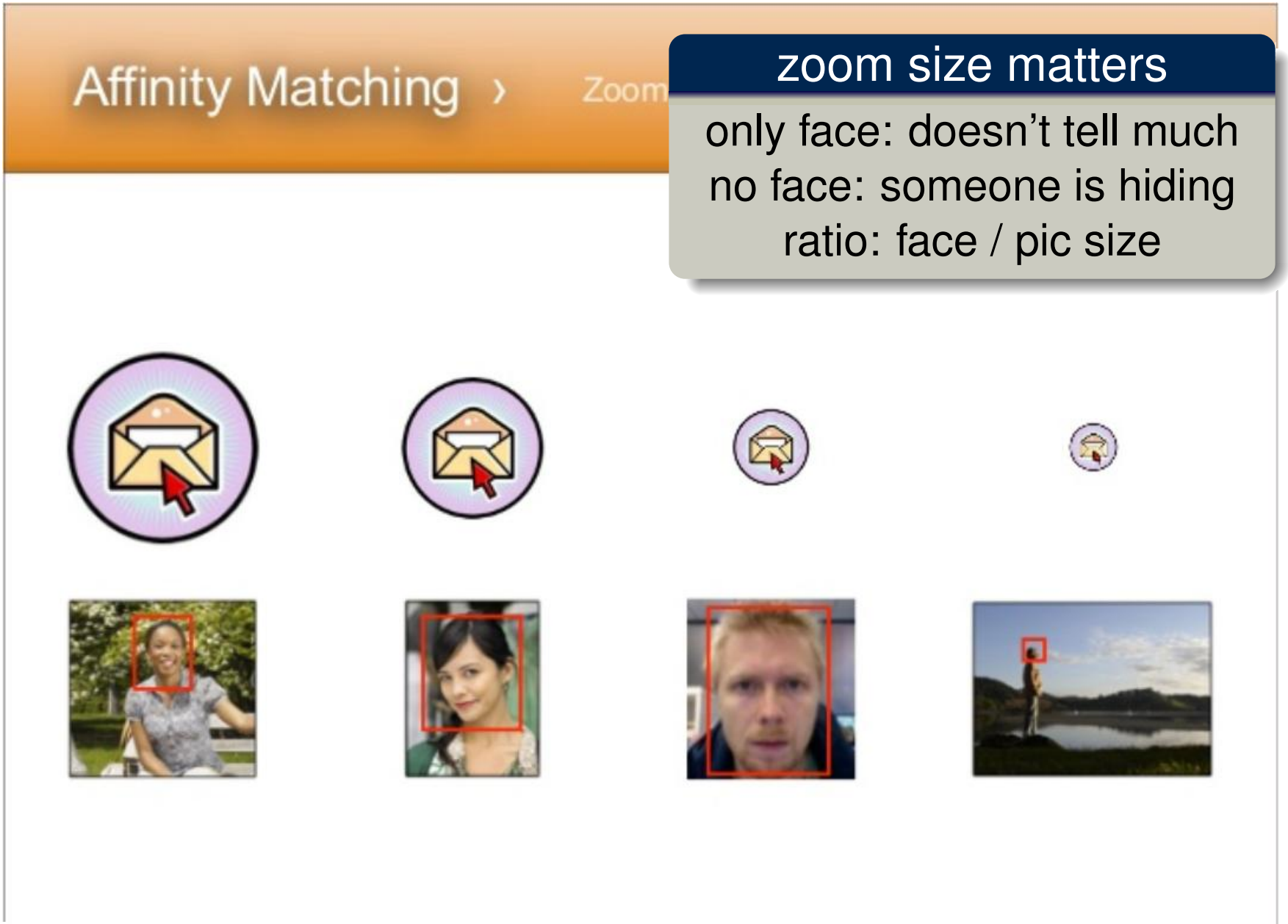

Affinity Matching ›
"Attra
self-reported attractiveness
people who report same attractiveness match better
Prob( )
-24 -19 -14 -9 -5 -1 3 7 11 16 21

Affinity Matching › Zoom level







zoom size matters

only face: doesn't tell much
no face: someone is hiding
ratio: face / pic size

Affinity Matching › Food preference

Affinity Matching › Food preference

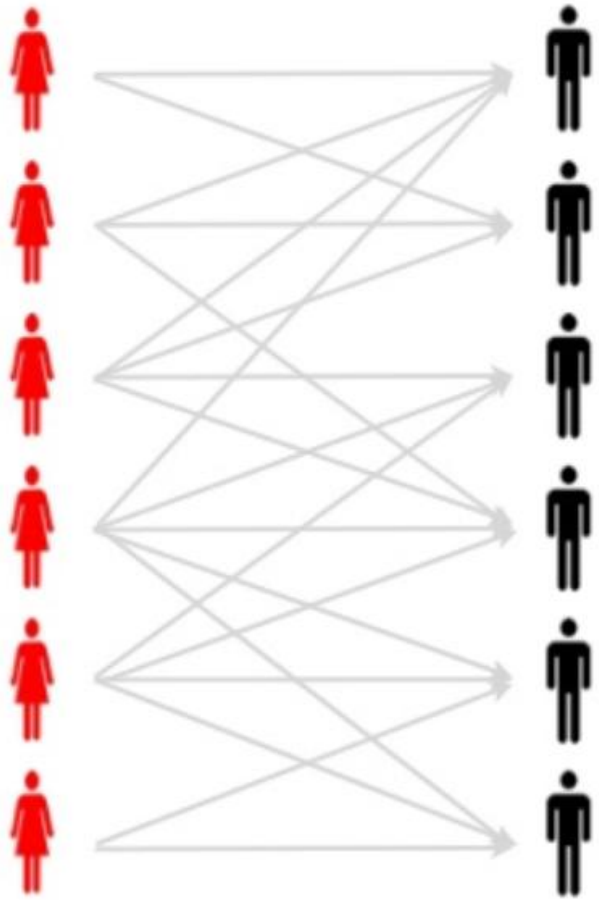
Matching in eHarmony.com

however

who should be introduced to whom and when?

Match Distribution › Graph optimization

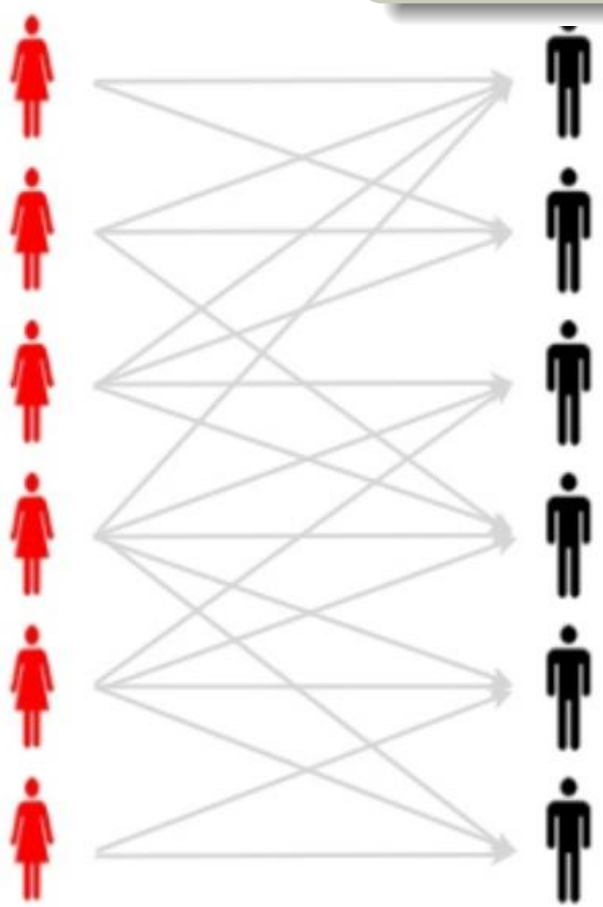
Match Distribution › Graph optimization



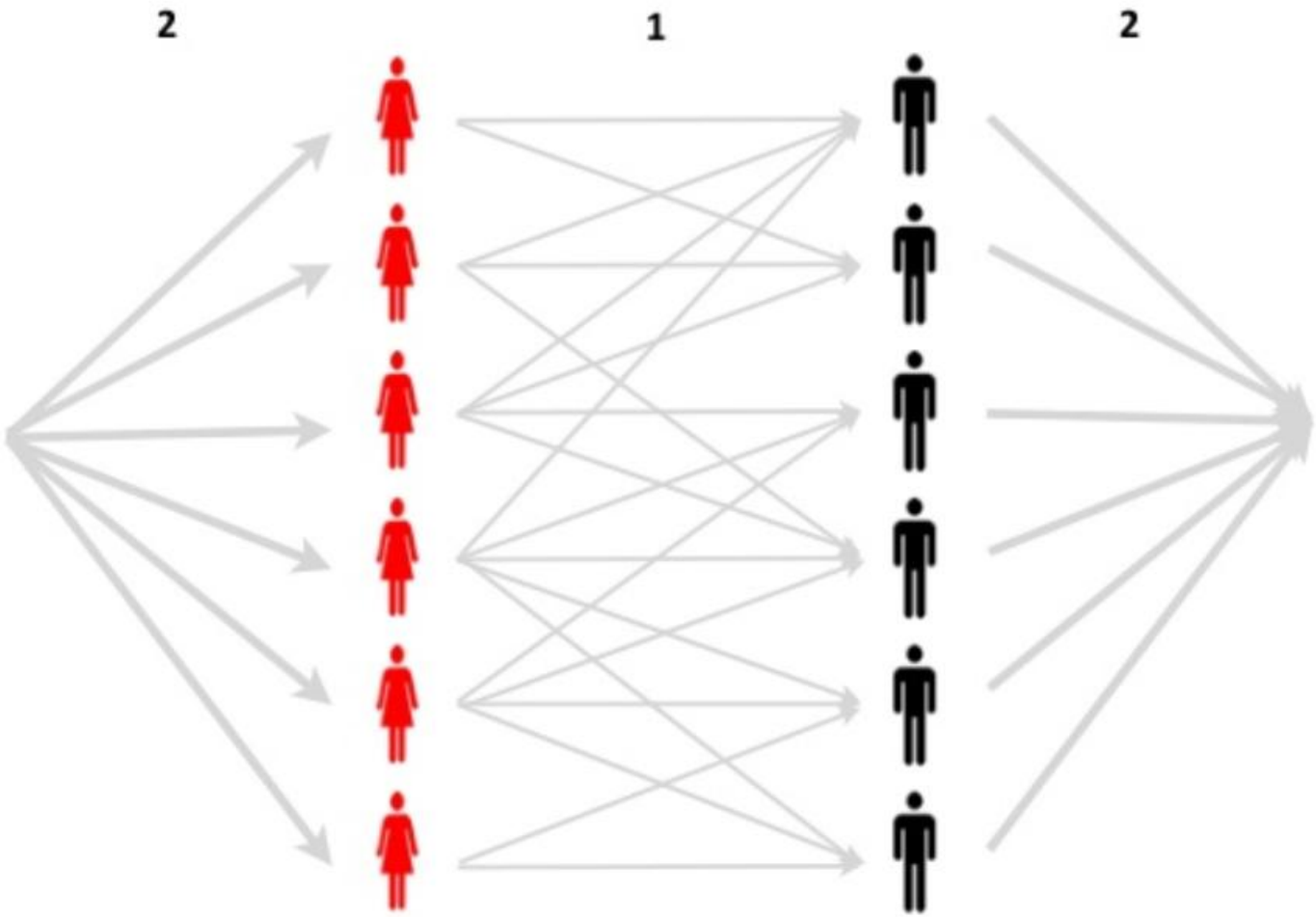
Match Distribution › Gra

although many are compatible

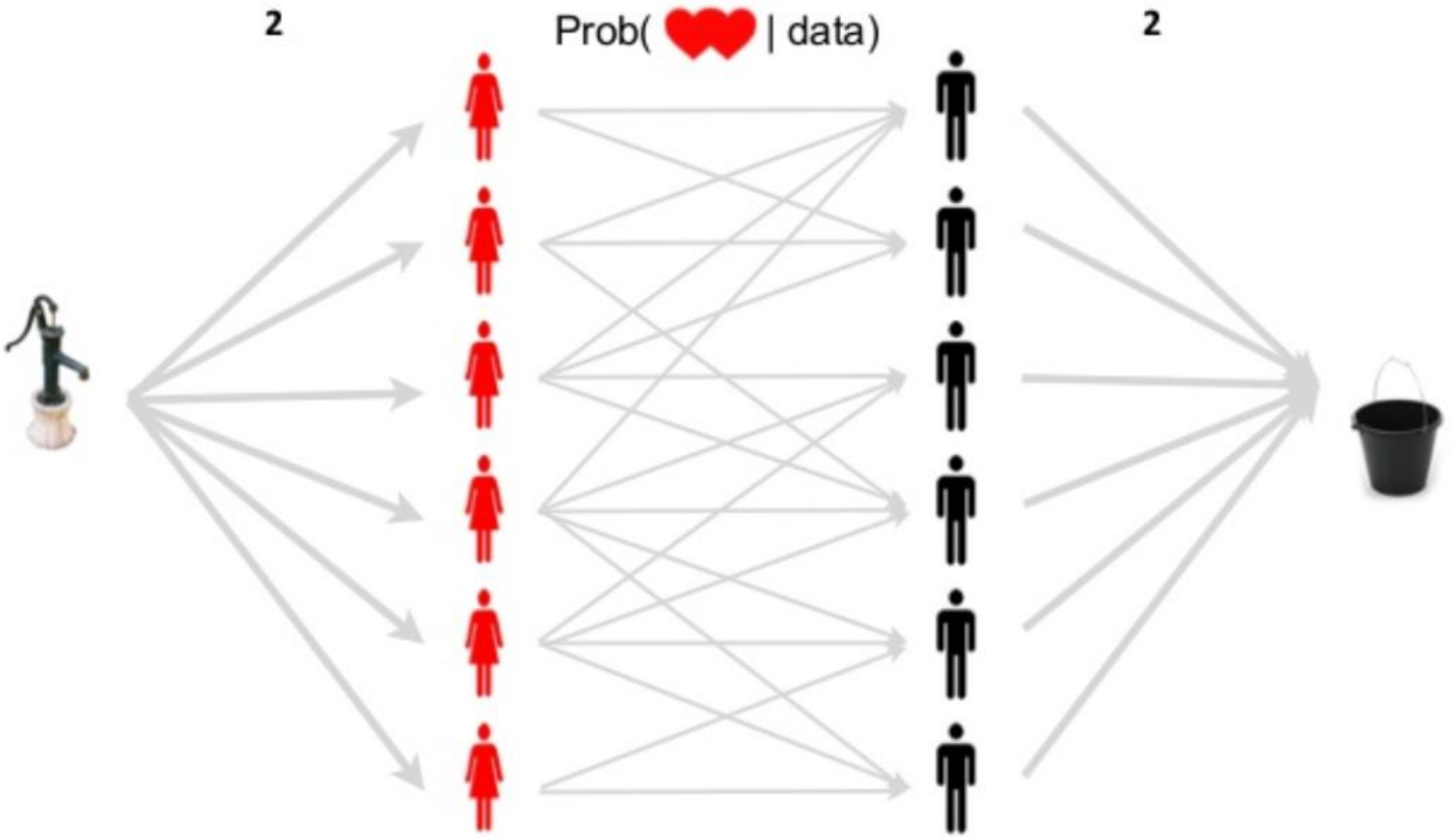
not all should be suggested



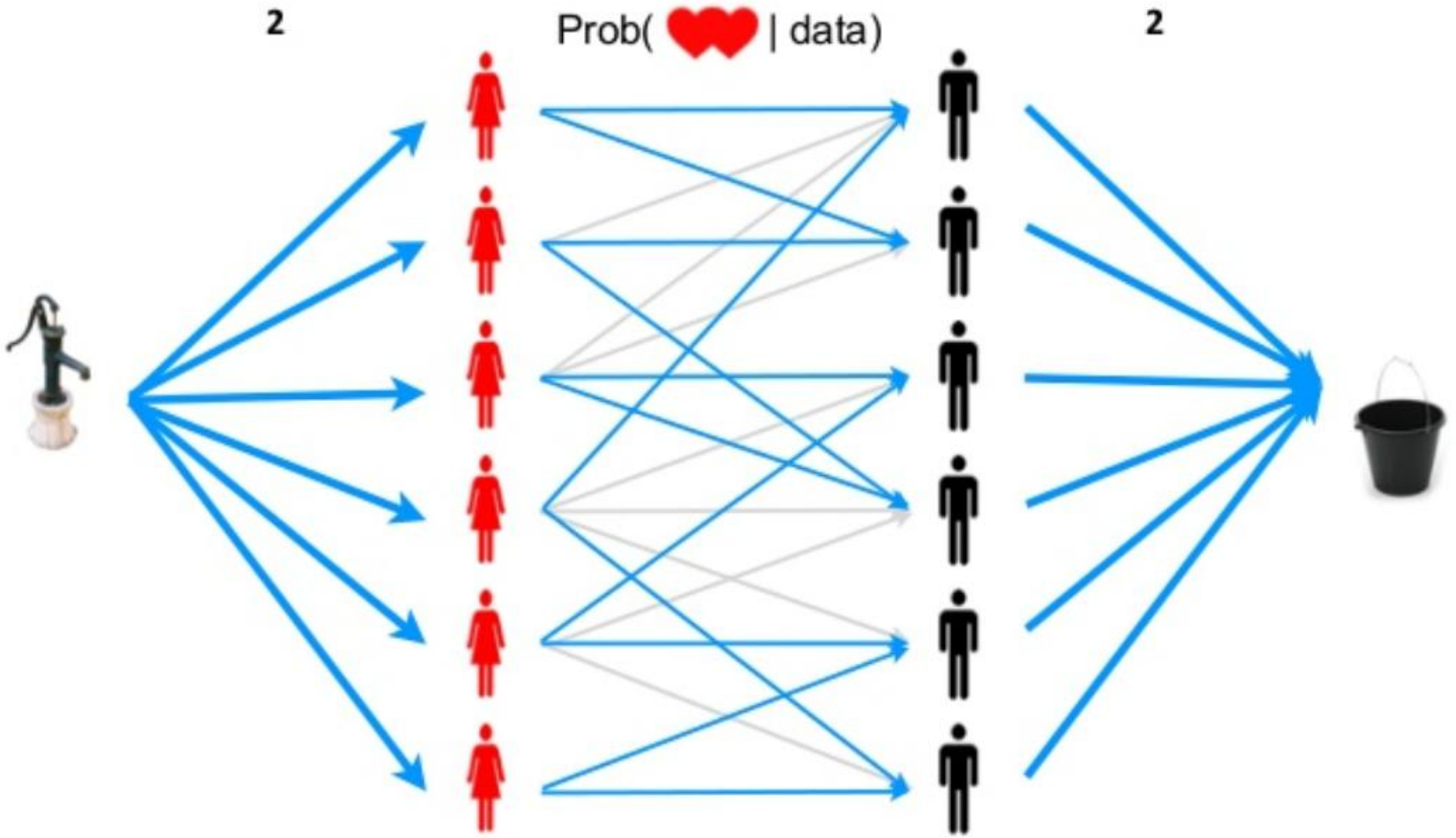
Match Distribution › Graph optimization

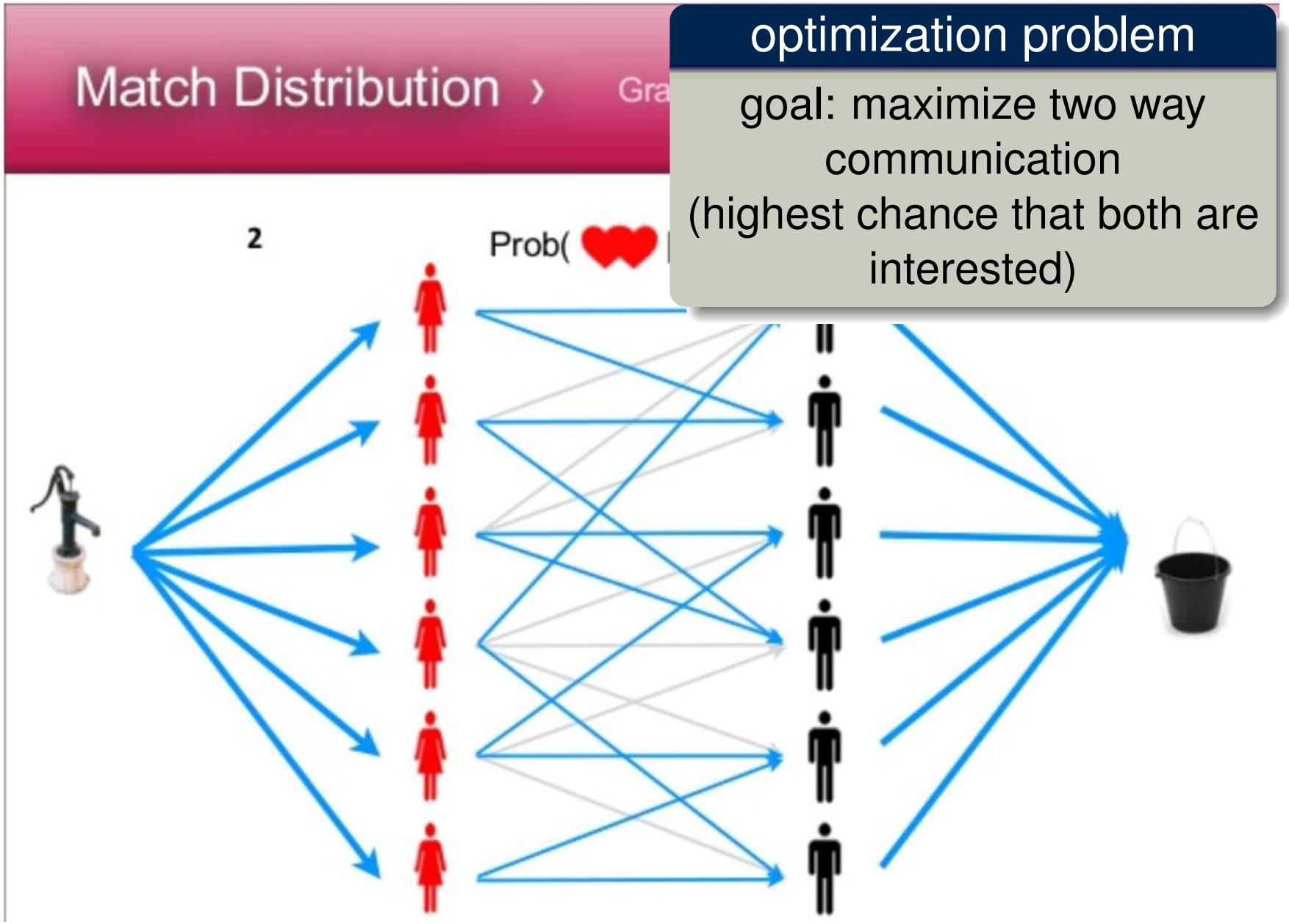


Match Distribution › Graph optimization



Match Distribution › Graph optimization





optimization problem

goal: maximize two way communication
(highest chance that both are interested)

Summary

Learning to Rank

- provides systematic ways to combine features

modeling

- pointwise: predict goodness of individual document
- pairwise: predict relative preference for document pairs
- listwise: predict effectiveness of ranked list of documents

explicit and implicit user inputs

- include relevance assessments, clicks, and skips

Thank you for your attention!

References

- Fuhr (1989): *Optimum Polynomial Retrieval Functions based on the the Probability Ranking Principle*, ACM TOIS 7(3).
- Liu (2009): *Learning to Rank for Information Retrieval*, Foundations and Trends in Information Retrieval 3(3):225â331.
- Joachims et al. (2007): Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search, ACM TOIS 25(2).
- Radlinski & Joachims (2005): Query Chains: Learning to Rank from Implicit Feedback, KDD.
- Kanhabua & Nørvåg (2012): Learning to Rank Search Results for Time-Sensitive Queries, CIKM.

Thanks

some slides / examples are taken from / similar to those of:

- Klaus Berberich, Saarland University, previous ATIR lecture
- Manning, Raghavan, Schütze: Introduction to Information Retrieval (including slides to the book)

- and the eharmony.com slides by Vaclav Petricek: http://www.slideshare.net/VaclavPetricek/data-science-of-love