Web Mining and Recommender Systems

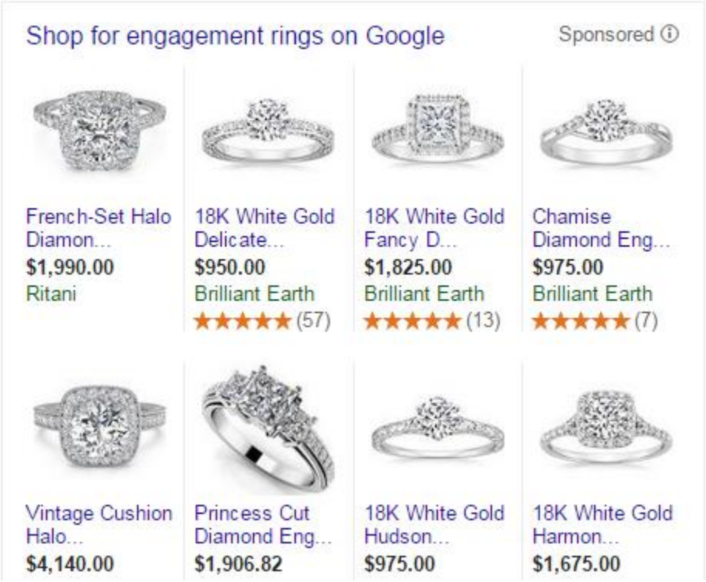
Algorithms for advertising

Learning Goals

- Introduce the topic of algorithmic advertising

Classification

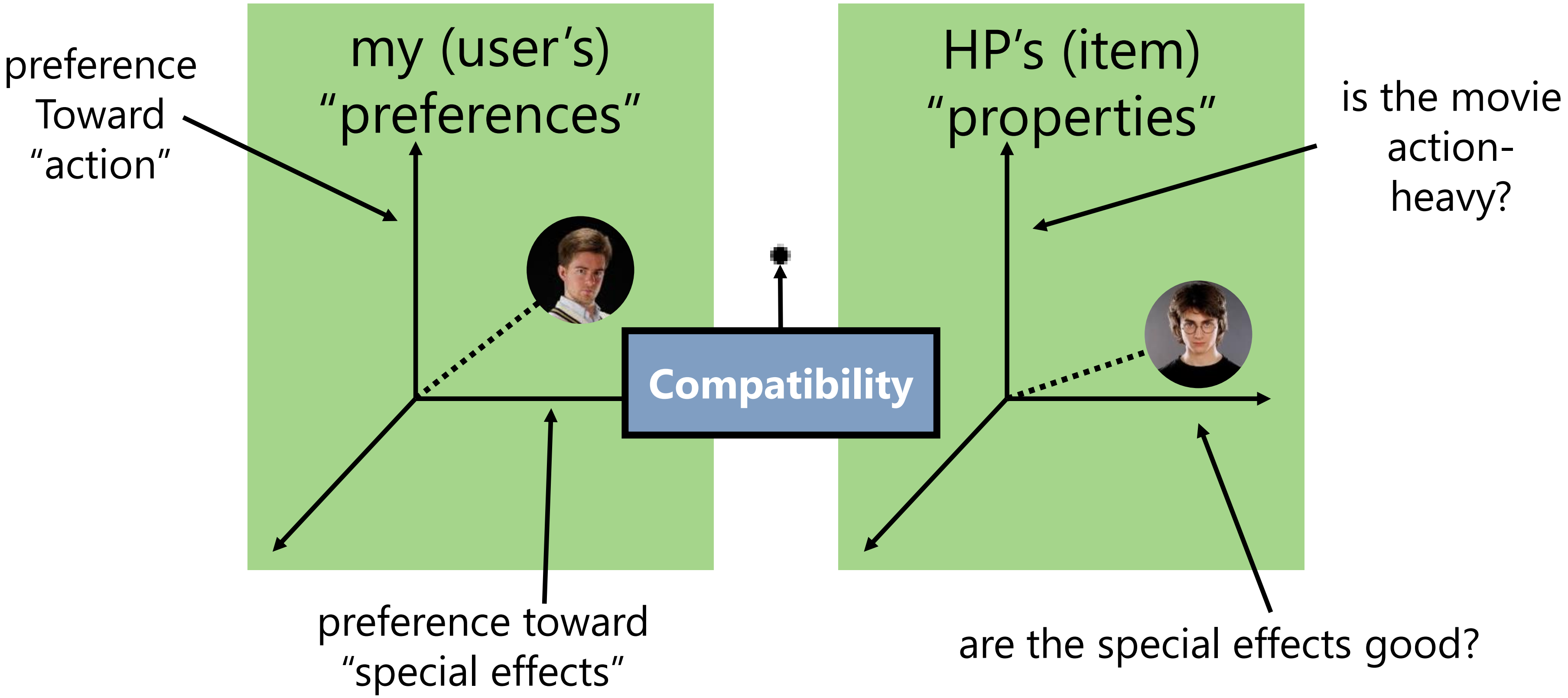
Predicting which ads people click on might be a **classification** problem



Will I **click on** this ad?

Recommendation

Or… predicting which ads people click on might be a **recommendation** problem



Advertising

So, we already have good algorithms for **predicting** whether a person would click on an ad, and generally for **recommending** items that people will enjoy.

So what's different about **ad recommendation?**

Advertising

1. We can’t recommend everybody the same thing (even if they all want it!)

- Advertisers have a limited budget – they wouldn’t be able to afford having their content recommended to everyone
- Advertisers **place bids** – we must take their bid into account (as well as the user’s preferences – or not)

- In other words, we need to consider **both** what the **user and the advertiser** want (this is in contrast to recommender systems, where the content didn’t get a say about whether it was recommended!)

Advertising

2. We need to be **timely**

- We want to make a personalized recommendations immediately (e.g. the moment a user clicks on an ad) – this means that we can’t train complicated algorithms (like what we saw with recommender systems) in order to make recommendations later
- We also want to update users’ models **immediately** in response to their actions

- (Also true for some recommender systems)

Advertising

3. We need to take **context** into account

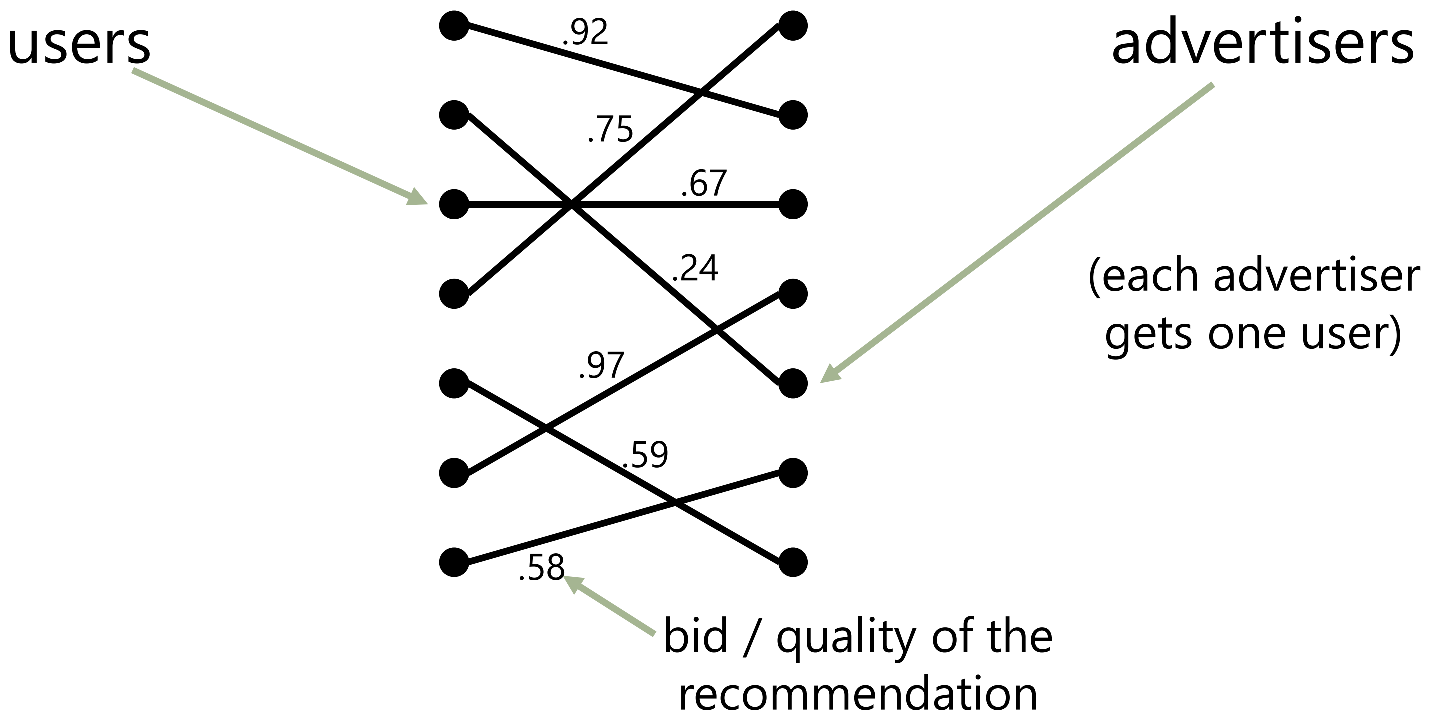
- Is the page a user is currently visiting particularly relevant to a particular type of content?
- Even if we have a good model of the user, recommending them the same type of thing over and over again is unlikely to succeed – nor does it teach us anything **new** about the user

- In other words, there's an **explore-exploit** tradeoff – we want to recommend things a user will enjoy (exploit), but also to discover new interests that the user may have (explore)

Advertising

So, ultimately we need

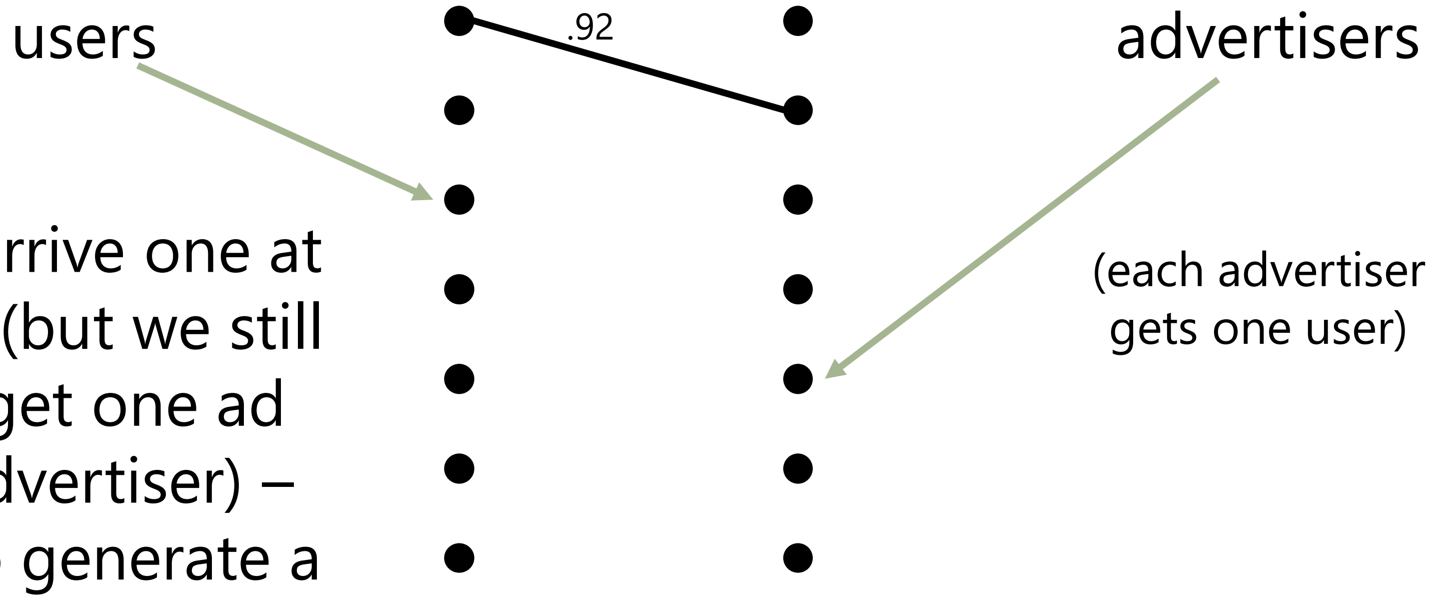
1) Algorithms to match users and ads, given **budget constraints**



Advertising

So, ultimately we need

2) Algorithms that work in real-time and don't depend on monolithic optimization problems



users arrive one at a time (but we still only get one ad per advertiser) – how to generate a good solution?

(each advertiser gets one user)

Advertising

So, ultimately we need

3) Algorithms that adapt to users and capture the notion of an exploit/explore tradeoff

Web Mining and Recommender Systems

Advertising: Matching problems

Learning Goals

- Introduce matching algorithms
- Explain the key differences between ad recommendation and other types of recommendation

Let's start with…

1. We can't recommend everybody the same thing (even if they all want it!)

- Advertisers have a limited budget – they wouldn't be able to afford having their content recommended to everyone
- Advertisers **place bids** – we must take their bid into account (as well as the user's preferences – or not)

- In other words, we need to consider **both** what the **user and the advertiser** want (this is in contrast to recommender systems, where the content didn't get a say about whether it was recommended!)

Bipartite matching

Let's start with a simple version of the problem we ultimately want to solve:

1) Every advertiser wants to show **one ad**
2) Every user gets to see **one ad**
3) We have some pre-existing model that assigns a score to user-item pairs

Bipartite matching

Suppose we're given some scoring function:

$$f(u, a) = \text{score for showing user } u \text{ ad } a$$

Could be:

- How much the owner of **a** is willing to pay to show their ad to **u**
- How much we expect the user **u** to spend if they click the ad **a**
- Probability that user **u** will click the ad **a**

Output of a regressor / logistic regressor!

Bipartite matching

Then, we'd like to show each user one ad, and we'd like each ad to be shown exactly once **so as to maximize this score** (bids, expected profit, probability of clicking etc.)

$$\sum_u f(u, ad(u))$$

s.t.

$$ad(u) = ad(v) \rightarrow u = v$$

each advertiser gets to show one ad

Bipartite matching

Then, we'd like to show each user one ad, and we'd like each ad to be shown exactly once **so as to maximize this score** (bids, expected profit, probability of clicking etc.)

$$\sum_{u,a} A_{u,a} f(u,a)$$

s.t.

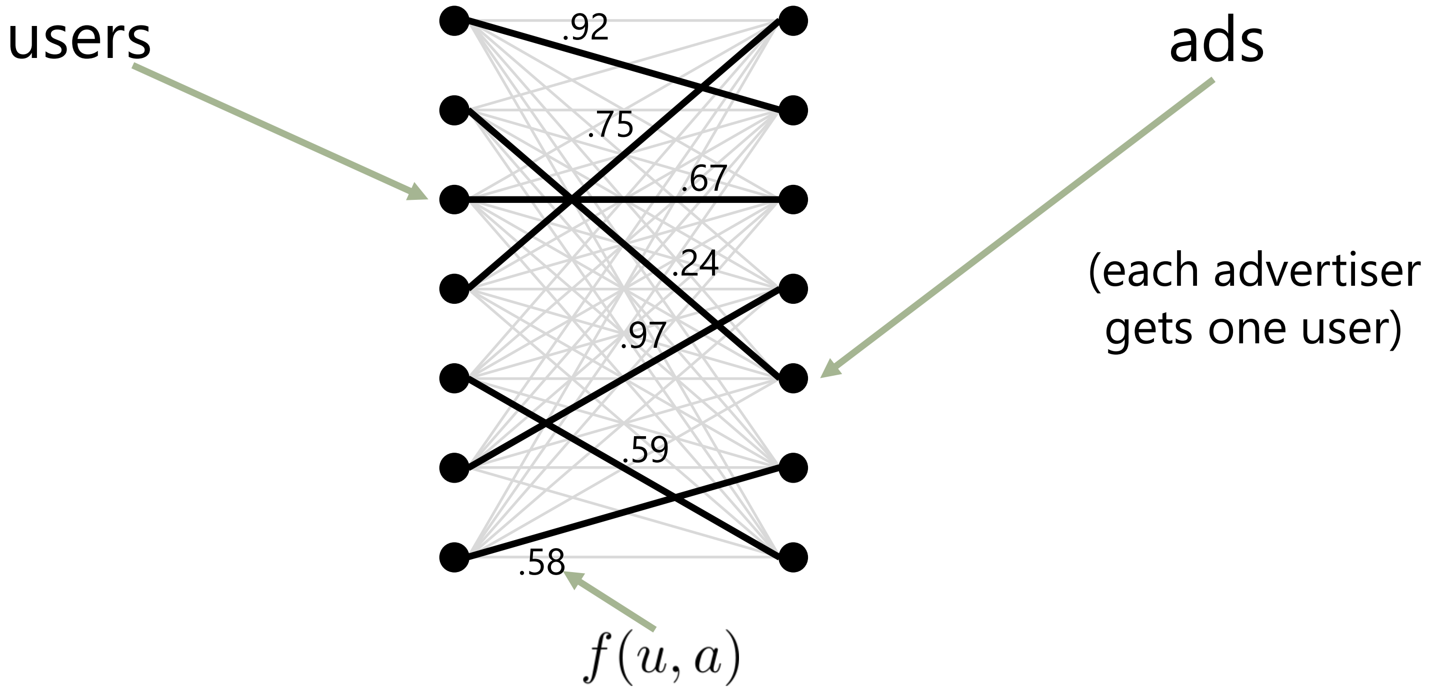
$$\forall a \;\; \sum_u A_{u,a} = 1$$

each advertiser gets to show one ad

Bipartite matching

We can set this up as a **bipartite matching** problem

- Construct a complete bipartite graph between users and ads, where each edge is weighted according to *f(u,a)*
- Choose edges such that each node is connected to exactly one edge

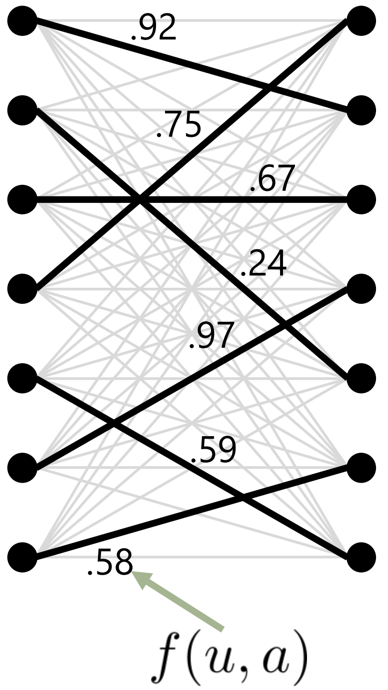


Bipartite matching

This is similar to the problem solved by (e.g.) online dating sites
to match men to women
For this reason it is called a **marriage problem**

men

women



(each user of an online dating platform gets shown exactly one result)

Bipartite matching

This is similar to the problem solved by (e.g.) online dating sites to match men to women
For this reason it is called a **marriage problem**

- A group of men should marry an (equally sized) group of women such that happiness is maximized, where "happiness" is measured by *f(m,w)*

compatibility between male *m* and female *w*

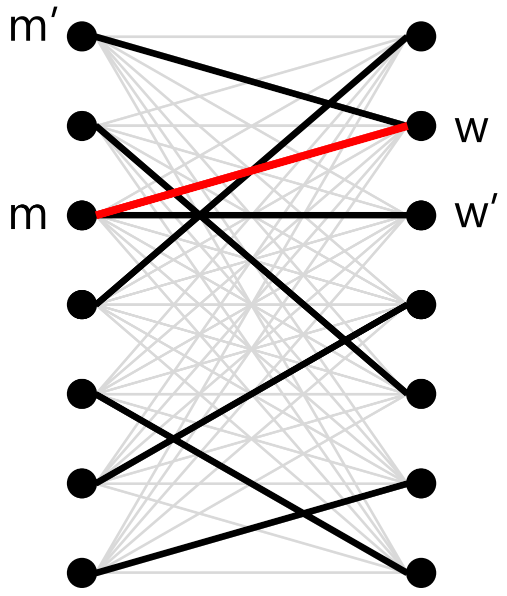
- Marriages are monogamous, heterosexual, and everyone gets married

(see also the original formulation, in which men have a preference function over women, and women have a *different* preference function over men)

Bipartite matching

We'll see one solution to this problem, known as **stable marriage**

- Maximizing happiness turns out to be quite hard
  - **But,** a solution is "**unstable**" if:



- A man *m* is matched to a woman *w′* but would prefer *w* (i.e., f(m,w′) < f(m,w))

and

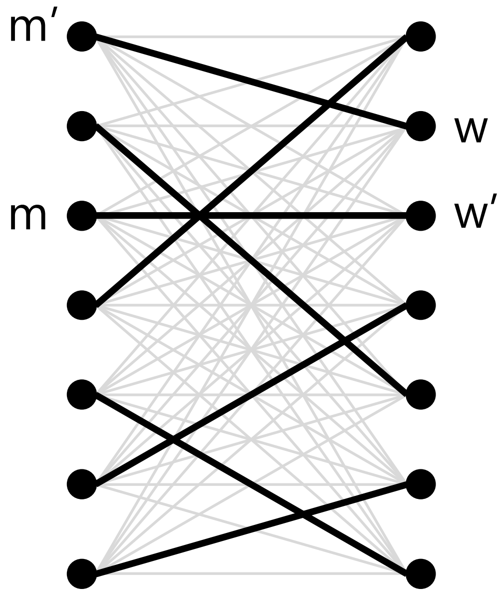
- The feeling is mutual – *w* prefers *m* to her partner (i.e., f(w,m′) < f(m,w))

- In other words, *m* and *w* would both want to "cheat" with each other

Bipartite matching

We'll see one solution to this problem, known as **stable marriage**

- A solution is said to be **stable** if this is **never satisfied** for any pair (m,w)



- Some people may covet another partner,

but

- The feeling is never reciprocated by the other person

- So no pair of people would **mutually** want to cheat

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- Men propose to women (this algorithm is from 1962!)
- While there is a man *m* who is not engaged
  - He selects his most compatible partner, $\max_w f(m,w)$ (to whom he has not already proposed)
  - If she is not engaged, they become engaged
  - If she *is* engaged (to *m'*), but prefers *m*, she breaks things off with *m'* and becomes engaged to *m* instead

Bipartite matching

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

```
All men and all women are initially 'free' (i.e., not engaged)
while there is a free man m, and a woman he has not proposed to
  w = max_w f(m,w)
  if (w is free):
    (m,w) become engaged (and are no longer free)
  else (w is engaged to m'):
    if w prefers m to m' (i.e., f(m,w) > f(m',w)):
      (m,w) become engaged
      m' becomes free
```

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The algorithm terminates

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The algorithm terminates

(either the number of free people decreases at each step, or, if it stays the same, the happiness increases)

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The solution is stable

Bipartite matching

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The solution is stable

(suppose m and w prefer each other to their current partners, w' and m'

But m would have proposed to w before he proposed to w'

- if w rejected his proposal, she must have been with someone she liked better
- if w accepted his proposal (but dumped him later), it must also have been for someone she likes better)

Bipartite matching

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The solution is $O(n^2)$

Bipartite matching

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The solution is O(n^2)

(every proposal is made at most once, and there are O(n^2) proposals
The input is O(n^2) (i.e., the compatibility function) so it certainly couldn’t be **better** than O(n^2))

Bipartite matching – extensions/improvements

Can all of this be improved upon?

1) It's not optimal

Bipartite matching – extensions/improvements

Can all of this be improved upon?

1) It's not optimal

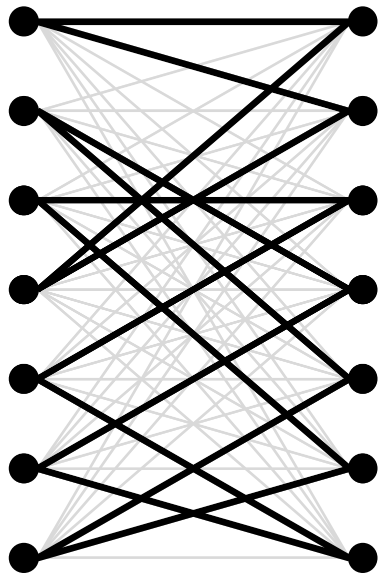
- Although there's no **pair** of individuals who would be happier by cheating, there could be **groups** of men and women who would be ultimately happier if the graph were rewired
- To get a **truly optimal** solution, there's a more complicated algorithm, known as the "Hungarian Algorithm"
  - But it's O(n^3)
- And really complicated and unintuitive (but there's a ref later)

Bipartite matching – extensions/improvements

Can all of this be improved upon?

2) Marriages are **monogamous**, heterosexual, and everyone gets married

(each user gets shown two ads, each ad gets shown to two users)



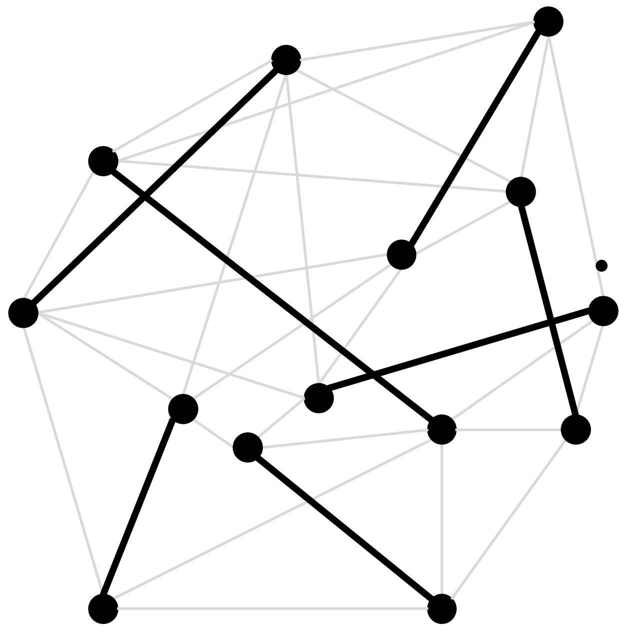
- Each advertiser may have a fixed budget of (1 or more) ads
- We may have room to show more than one ad to each customer

- See "Stable marriage with multiple partners: efficient search for an optimal solution" (refs)

Bipartite matching – extensions/improvements

Can all of this be improved upon?

2) Marriages are monogamous, **heterosexual**, and everyone gets married

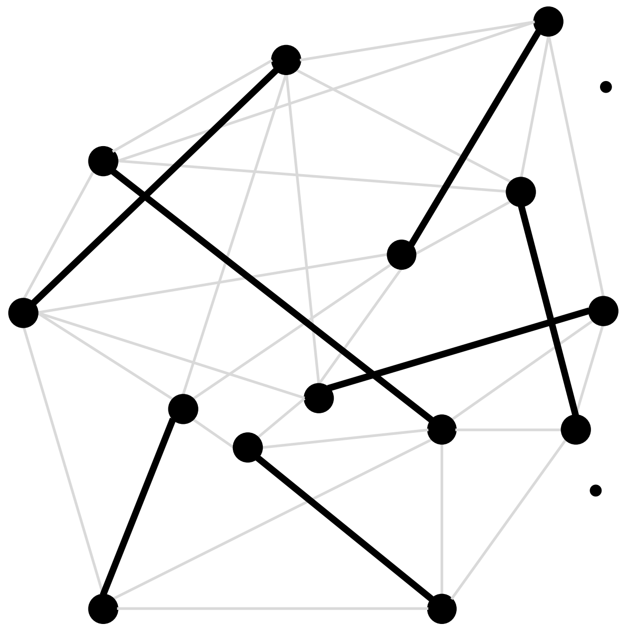


- This version of the problem is know as **graph cover** (select edges such that each node is connected to exactly one edge)
- The algorithm we saw is really just graph cover for a bipartite graph
- Can be solved via the "stable roommates" algorithm (see refs) and extended in the same ways

Bipartite matching – extensions/improvements

Can all of this be improved upon?

2) Marriages are monogamous, **heterosexual**, and everyone gets married

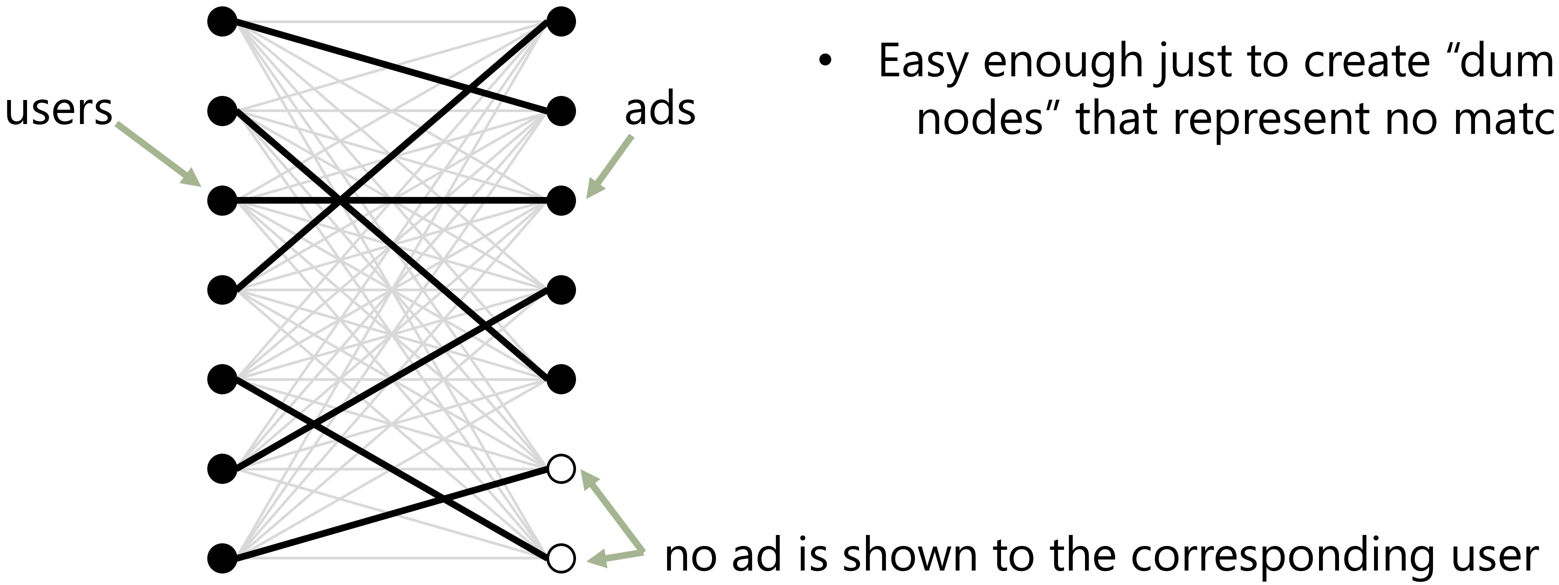


- This version of the problem can address a very different variety of applications compared to the bipartite version
- Roommate matching
- Finding chat partners
- (or any sort of person-to-person matching)

Bipartite matching – extensions/improvements

Can all of this be improved upon?

2) Marriages are monogamous, heterosexual, and **everyone gets married**



- Easy enough just to create "dummy nodes" that represent no match

Bipartite matching – applications

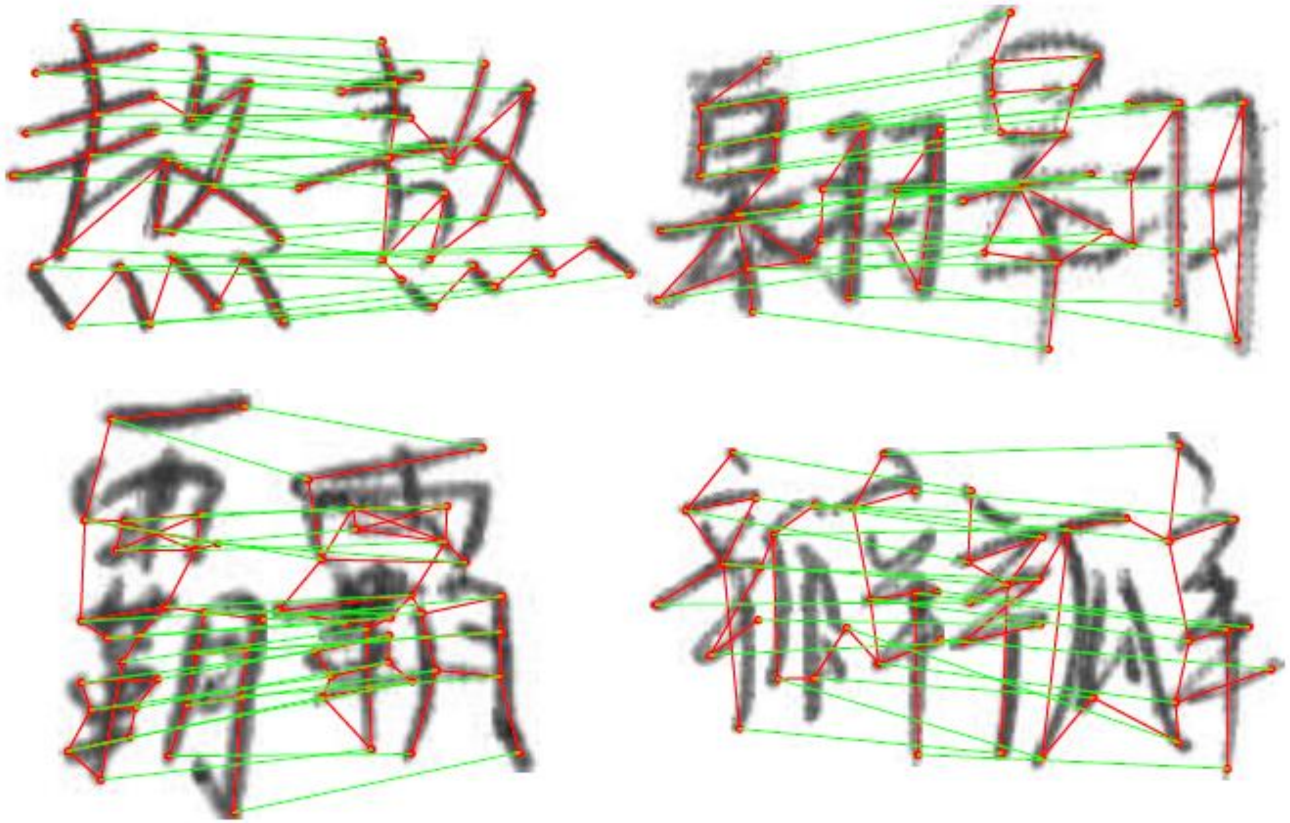
Why are matching problems so important?

- Advertising
- Recommendation
- Roommate assignments
- Assigning students to classes
- General resource allocation problems
- Transportation problems (see “Methods of Finding the Minimal Kilometrage in Cargo-transportation in space”)
- Hospitals/residents

Bipartite matching – applications

Why are matching problems so important?

- Point pattern matching



Bipartite matching – extensions/improvements

What about more complicated rules?

- (e.g. for hospital residencies) Suppose we want to keep couples together
- Then we would need a more complicated function that encodes these pairwise relationships:

$$\sum_{u,v} f(u, v, hospital(u), hospital(v))$$

pair of residents   hospitals to which they're assigned

So far...

Surfacing ads to users is a like a little like building a **recommender system** for ads

- We need to model the compatibility between each user and each ad (probability of clicking, expected return, etc.)
- **But,** we can't recommend the same ad to every user, so we have to handle "budgets" (both how many ads can be shown to each user and how many impressions the advertiser can afford)
- **So,** we can cast the problem as one of "covering" a bipartite graph
- Such **bipartite matching** formulations can be adapted to a wide variety of tasks

Learning Outcomes

- Introduced algorithms for matching
- Explained how ad recommendation problems have *constraints* not present in other forms of recommendation

Questions?

Further reading:

- The original stable marriage paper

“College Admissions and the Stability of Marriage” (Gale, D.; Shapley, L. S., 1962): https://www.jstor.org/stable/2312726

- The Hungarian algorithm

“The Hungarian Method for the assignment problem” (Kuhn, 1955): https://tom.host.cs.st-andrews.ac.uk/CS3052-CC/Practicals/Kuhn.pdf

- Multiple partners

“Stable marriage with multiple partners: efficient search for an optimal solution” (Bansal et al., 2003)

- Graph cover & stable roommates

“An efficient algorithm for the ‘stable roommates’ problem” (Irving, 1985) https://dx.doi.org/10.1016%2F0196-6774%2885%2990033-1

Web Mining and Recommender Systems

AdWords

Learning Goals

- Introduce the AdWords algorithm
- Explain the need to make ad recommendations in "real time"

Advertising

1. We can’t recommend everybody the same thing (even if they all want it!)

- So far, we have an algorithm that takes “budgets” into account, so that users are shown a limited number of ads, and ads are shown to a limited number of users
- **But,** all of this only applies if we see all the users and all the ads **in advance**

- This is what’s called an **offline algorithm**

Advertising

2. We need to be **timely**

- But in many settings, users/queries come in one at a time, and need to be shown some (highly compatible) ads
- But we still want to satisfy the same quality and budget constraints

- So, we need **online algorithms** for ad recommendation

What is adwords?

Adwords allows advertisers to bid on keywords

- This is similar to our matching setting in that advertisers have limited **budgets,** and we have limited space to show ads

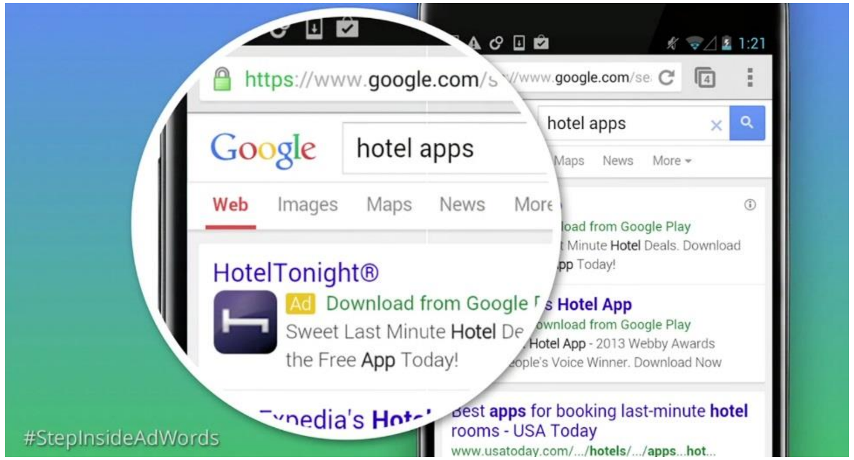


image from blog.adstage.io

What is adwords?

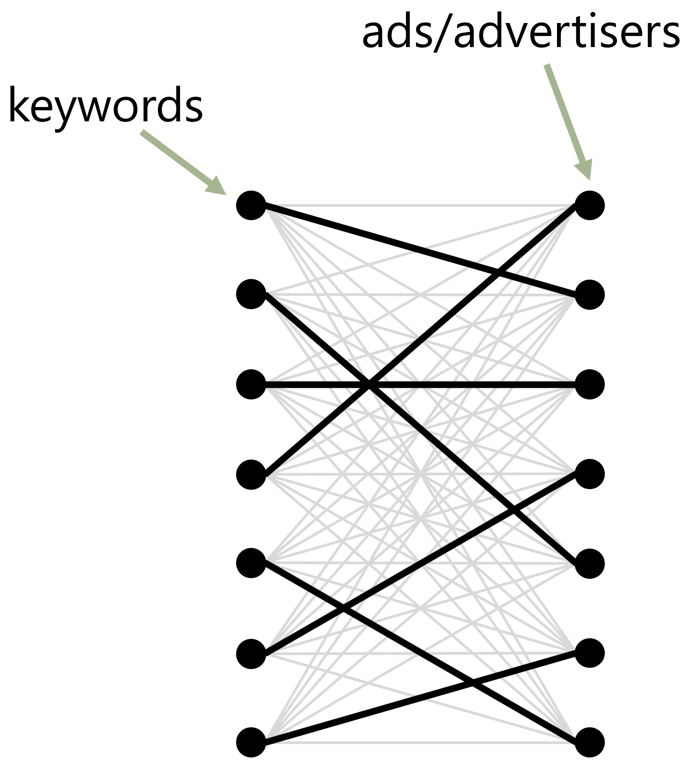
Adwords allows advertisers to bid on keywords

- This is similar to our matching setting in that advertisers have limited **budgets,** and we have limited space to show ads
- **But,** it has a number of key differences:

1. Advertisers don't pay for impressions, but rather they pay when their ads get clicked on
2. We don't get to see all of the queries (keywords) in advance – **they come one-at-a-time**

What is adwords?

Adwords allows advertisers to bid on keywords



- We still want to match advertisers to keywords to satisfy budget constraints
- But can’t treat it as a monolithic optimization problem like we did before
- Rather, we need an **online** algorithm

What is adwords?

Suppose we're given

- Bids that each advertiser is willing to make for each query

$$f(\underbrace{q}_{\text{query}}, \underbrace{a}_{\text{advertiser}})$$

(this is how much they'll pay **if the ad is clicked on**)

- Each is associated with a click-through rate

$$ctr(q, a)$$

- Budget for each advertiser $b(a)$ (say for a 1-week period)
- A limit on how many ads can be returned for each query

What is adwords?

And, every time we see a query

- Return at most the number of ads that can fit on a page
- And which won't overrun the budget of the advertiser (if the ad is clicked on)

Ultimately, what we want is an algorithm that maximizes **revenue** – the number of ads that are clicked on, multiplied by the bids on those ads

Competitiveness ratio

What we'd like is:

the revenue should be as close as possible to what we *would* have obtained if we'd seen the whole problem up front
(i.e., if we didn't have to solve it online)

We'll define the **competitive ratio** as:

$$\frac{\text{revenue of our algorithm}}{\text{revenue of an optimal algorithm}}$$

see http://infolab.stanford.edu/~ullman/mmds/book.pdf for more detailed definition

Greedy solution

Let's start with a simple version of the problem…

1. One ad per query
2. Every advertiser has the same budget
3. Every ad has the same click through rate
4. All bids are either 0 or 1

(either the advertiser wants the query, or they don't)

Greedy solution

Then the greedy solution is…

- Every time a new query comes in, select any advertiser who has bid on that query (who has budget remaining)
- What is the competitive ratio of this algorithm?

Greedy solution

A better algorithm…

- Every time a new query comes in, amongst advertisers who have bid on this query, **select the one with the largest remaining budget**
- How would this do on the same sequence?

The balance algorithm

A better algorithm…

- Every time a new query comes in, amongst advertisers who have bid on this query, **select the one with the largest remaining budget**

- In fact, the competitive ratio of this algorithm (still with equal budgets and fixed bids) is (1 – 1/e) ~ 0.63

see http://infolab.stanford.edu/~ullman/mmds/book.pdf for proof

The balance algorithm

What if bids aren't equal?

Bidder	Bid (on q)	Budget
A	1	110
B	10	100

The balance algorithm

What if bids aren't equal?

Bidder	Bid (on q)	Budget
A		
B		

The balance algorithm v2

We need to make two modifications

- We need to consider the bid amount when selecting the advertiser, and bias our selection toward higher bids
- We also want to use some of each advertiser's budget (so that we don't just ignore advertisers whose budget is small)

The balance algorithm v2

Advertiser: A_i

fraction of budget remaining: f_i

bid on query *q*: $x_i(q)$

Assign queries to whichever advertiser maximizes:

$$\Psi_i(q) = x_i(q) \cdot (1 - e^{-f_i})$$

(could multiply by click-through rate if click-through rates are not equal)

The balance algorithm v2

Properties

- This algorithm has a competitive ratio of $(1 - \frac{1}{e})$.

- In fact, there **is no online algorithm** for the adwords problem with a competitive ratio **better than** $(1 - \frac{1}{e})$.

(proof is too deep for me…)

So far we have seen…

- An **online** algorithm to match advertisers to users (really to queries) that handles both **bids** and **budgets**
  - We wanted our **online** algorithm to be as good as the **offline** algorithm would be – we measured this using the **competitive ratio**
- Using a specific scheme that favored high bids while trying to balance the budgets of all advertisers, we achieved a ratio of $(1 - \frac{1}{e})$.
  - And no better online algorithm exists!

Adwords

We **haven't** seen…

- AdWords actually uses a **second-price** auction (the winning advertiser pays the amount that the **second** highest bidder bid)
- Advertisers don't bid on specific queries, but inexact matches ('broad matching') – i.e., queries that include subsets, supersets, or synonyms of the keywords being bid on

Learning Outcomes

- Introduced the AdWords algorithm
- Showed how to greedily recommend ads in real time
- Discussed theoretical properties of this solution

Questions?

Further reading:

- Mining of Massive Datasets – “The Adwords Problem”
  http://infolab.stanford.edu/~ullman/mmds/book.pdf
- AdWords and Generalized On-line Matching (A. Mehta)
  http://web.stanford.edu/~saberi/adwords.pdf

Web Mining and Recommender Systems

Bandit algorithms

Learning Goals

- Introduce Bandit algorithms
- Discuss the notion of exploration/exploitation tradeoffs for ad recommendation
- Discuss how to incorporate learning into an ad recommendation algorithm

So far…

1. We’ve seen algorithms to handle **budgets** between users (or queries) and advertisers
2. We’ve seen an **online** version of these algorithms, where queries show up one at a time
3. Next, how can we **learn** about which ads the user is likely to click on in the first place?

3. How can we **learn** about which ads the user is likely to click on in the first place?

- If we see the user click on a car ad once, we know that (maybe) they have an interest in cars
  - So… we know they like car ads, should we keep recommending them car ads?

- **No,** they'll become less and less likely to click it, and in the meantime we won't learn anything new about what **else** the user might like

Bandit algorithms

- **Sometimes** we should surface car ads (which we know the user likes),
- **but sometimes,** we should be willing to take a risk, so as to learn what **else** the user might like

one-armed bandit

Setup

. . .

K bandits (i.e., K arms)

round t							
t = 1	1	0	0	1	1	0	1
2	0	0	1	1	0	1	0
3	1	1	1	0	1	1	0
4	1	0	1	0	0	0	0
5	0	1	0	0	1	0	0
6	0	0	0	0	1	1	0
7	0	0	1	0	0	1	0
8	0	1	1	0	0	1	1
9	1	0	1	0	0	0	1

reward $g_{k,t}$

- At each round t, we select an arm to pull
- We'd like to pull the arm to maximize our total reward

Setup

. . .

K bandits (i.e., K arms)

round t							
t = 1	?	?	?	?	?	?	?
2	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?
4	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?
7	?	?	?	?	?	?	?
8	?	?	?	?	?	?	?
9	?	?	?	?	?	?	?

reward $g_{k,t}$

- At each round t, we select an arm to pull
- We'd like to pull the arm to maximize our total reward
- **But –** we don't get to see the reward function!

Setup

. . .

K bandits (i.e., *K* arms)

round t							
t = 1	1	?	?	?	?	?	?
2	?	0	?	?	?	?	?
3	?	?	?	?	1	?	?
4	?	?	?	?	0	?	?
5	0	?	?	?	?	?	?
6	?	?	?	0	?	?	?
7	?	?	?	?	?	1	?
8	?	?	?	?	?	?	1
9	?	?	?	?	?	?	1

reward $g_{k,t}$

- At each round *t*, we select an arm to pull
- We'd like to pull the arm to maximize our total reward
- **But –** we don't get to see the reward function!
- All we get to see is the reward we got **for the arm we picked** at each round

Setup

K : number of arms (ads)

n : number of rounds

$g_t = (g_{1,t}, \ldots, g_{K,t}) \in [0,1]^K$: rewards

$l_t \in \{1, \ldots, K\}$: which arm we pick at each round

$g_{l_t,t} \in [0,1]$: how much (0 or 1) this choice wins us

want to minimize **regret:**

$$R_n = \left(\max_{i=1\ldots K} \mathbb{E} \sum_{t=1}^{n} g_{i,t}\right) - \mathbb{E} \sum_{t=1}^{n} g_{l_t,t}$$

reward we **could** have got, if we had played optimally

reward our strategy would get (in expectation)

Goal

- We need to come up with a **strategy** for selecting arms to pull (ads to show) that would maximize our expected reward
- For the moment, we're assuming that rewards are static, i.e., that they don't change over time

Strategy 1 – "epsilon first"

- Pull arms at random for a while to learn the distribution, then just pick the best arm
- (show random ads for a while until we learn the user's preferences, then just show what we know they like)

$\epsilon \cdot n$: Number of steps to sample randomly

$(1 - \epsilon) \cdot n$: Number of steps to choose optimally

Strategy 1 – "epsilon first"

- Pull arms at random for a while to learn the distribution, then just pick the best arm
- (show random ads for a while until we learn the user's preferences, then just show what we know they like)

Strategy 2 – “epsilon greedy”

- Select the best lever most of the time, pull a random lever some of the time
- (show random ads sometimes, and the best ad most of the time)

ϵ : Fraction of times to sample randomly

$(1 - \epsilon)$: Fraction of times to choose optimally

- Empirically, worse than epsilon-first
- Still doesn’t handle context/time

Strategy 3 – "epsilon decreasing"

- Same as epsilon-greedy (Strategy 2), but epsilon decreases over time

Strategy 4 – "**Adaptive** epsilon greedy"

- Similar to as epsilon-decreasing (Strategy 3), but epsilon can increase **and** decrease over time

Extensions

- The reward function may not be **static,** i.e., it may change each round according to some process
- It could be chosen by an **adversary**
- The reward may not be [0,1] (e.g. clicked/not clicked), but instead a could be a real number (e.g. revenue), and we'd want to estimate the distribution over rewards

Extensions – Contextual Bandits

- There could be **context** associated with each time step
    - The query the user typed
    - What the user saw during the **previous** time step
    - What other actions the user has recently performed
    - Etc.

Applications (besides advertising)

- Clinical trials

(assign drugs to patients, given uncertainty about the outcome of each drug)

- Resource allocation

(assign person-power to projects, given uncertainty about the reward that different projects will result in)

- Portfolio design

(invest in ventures, given uncertainty about which will succeed)

- Adaptive network routing

(route packets, without knowing the delay unless you send the packet)

Learning Outcomes

- Introduced Bandit algorithms
- Discussed the notion of exploration/exploitation tradeoffs for ad recommendation
- Saw some settings beyond advertising where this notion could be useful

References

Further reading:

Tutorial on Bandits:

https://sites.google.com/site/banditstutorial/

Web Mining and Recommender Systems

Case study – Turning down the noise

Turning down the noise

"Turning down the noise in the Blogosphere"

(By Khalid El-Arini, Gaurav Veda, Dafna Shahaf, Carlos Guestrin)

Goals:

1. Help to **filter** huge amounts of content, so that users see content that is **relevant** – rather than seeing popular content over and over again
2. Maximize **coverage** so that a variety of different content is recommended
3. Make recommendations that are **personalized** to each user

some slides http://www.select.cs.cmu.edu/publications/paperdir/kdd2009-elarini-veda-shahaf-guestrin.pptx

Turning down the noise

“Turning down the noise in the Blogosphere”

(By Khalid El- … los Guestrin)

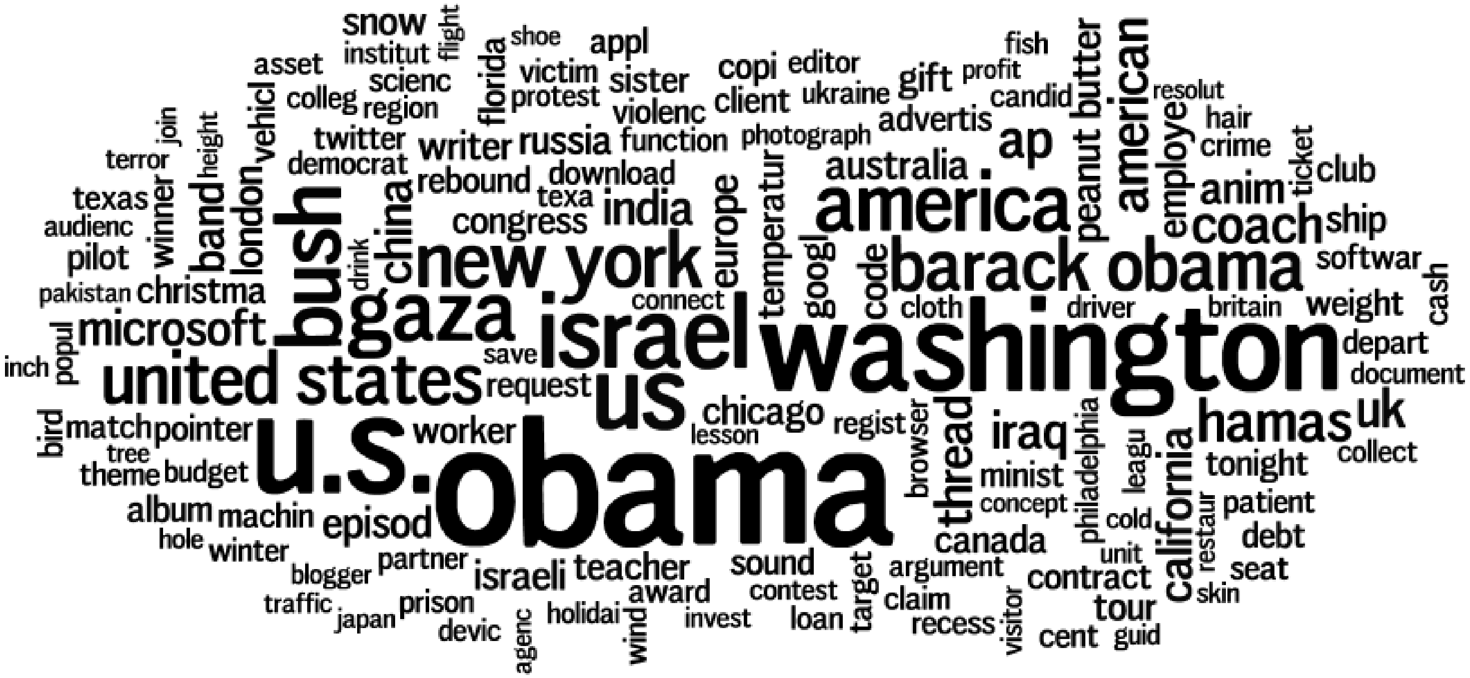
1. Help to … at users see conte… g popular
2. Maximize … nt content is … recommended
3. Make recommendations that are **personalized** to each user

Similar to our goals with **bandit algorithms**

- **Exploit** by recommending content that we user is likely to enjoy (personalization)
- **Explore** by recommending a variety of content (coverage)

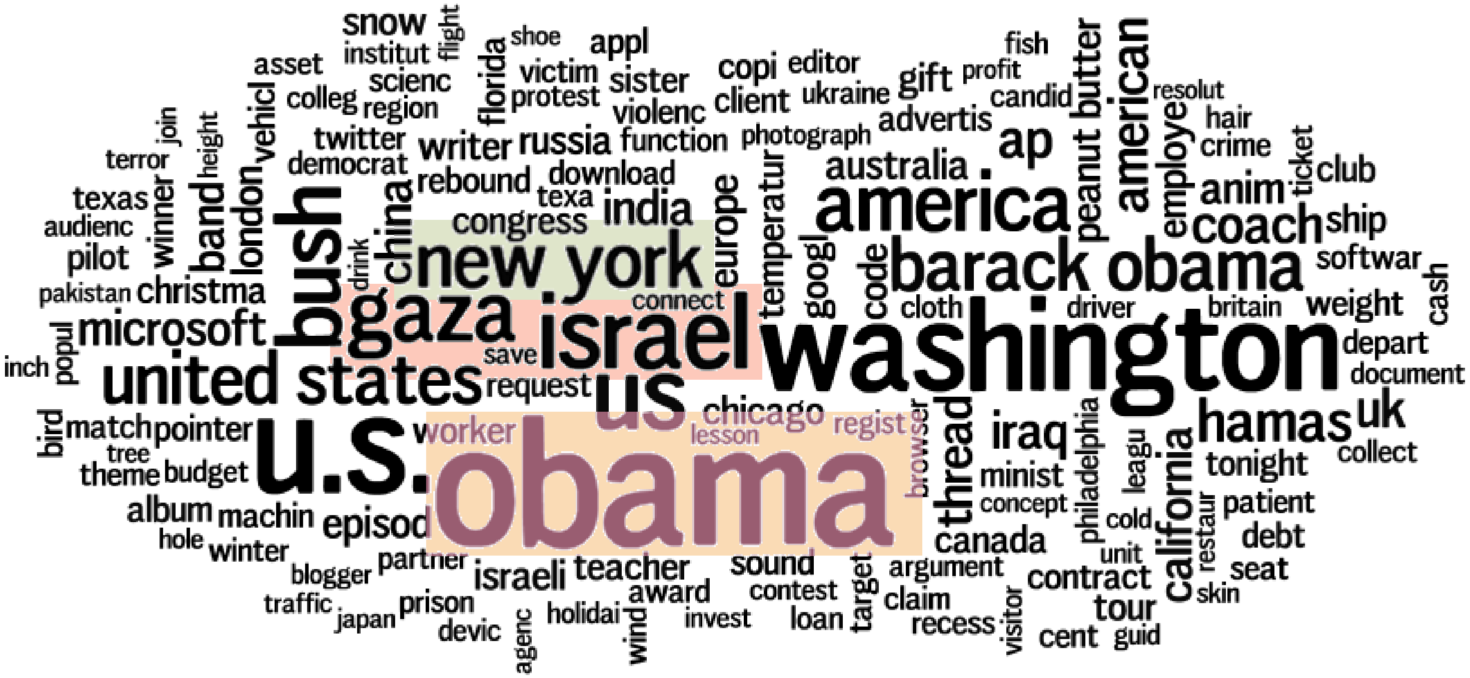
Turning down the noise

1. Help to **filter** huge amounts of content, so that users see content that is **relevant**



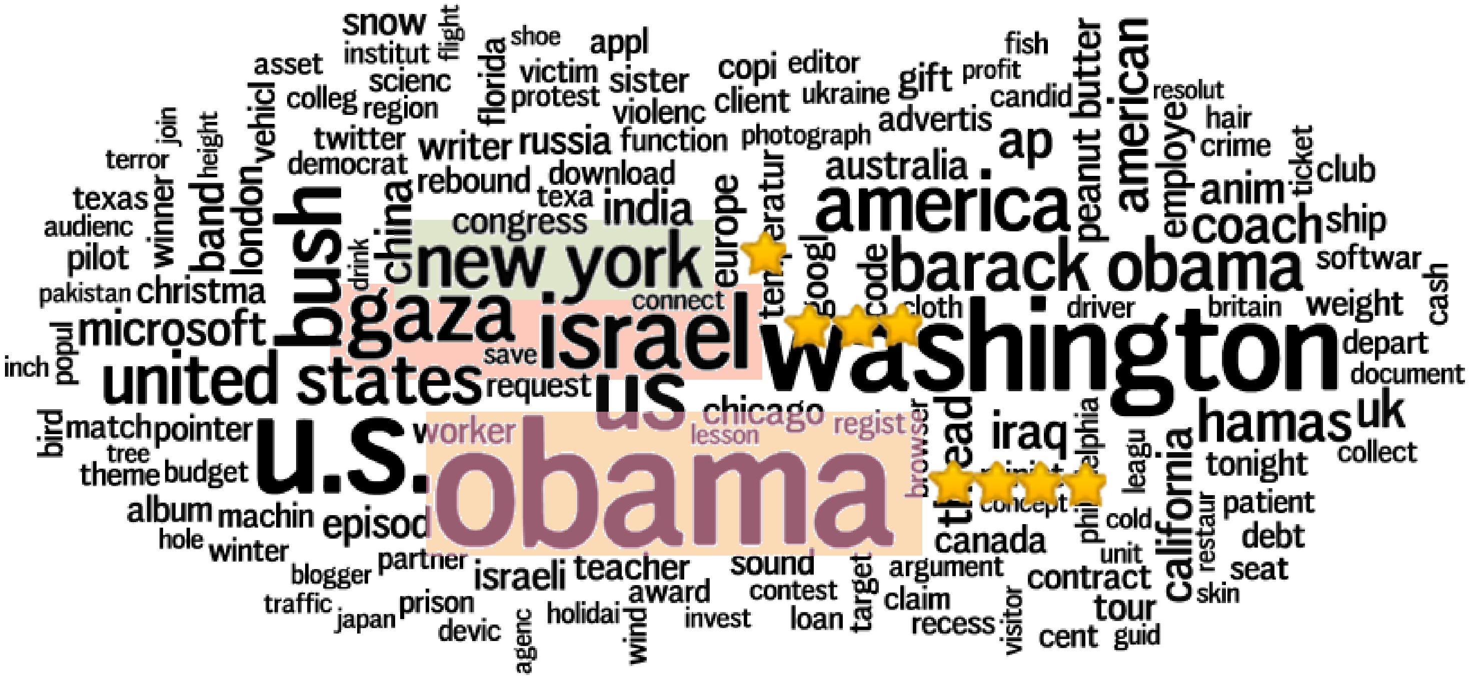
Turning down the noise

2. Maximize **coverage** so that a variety of different content is recommended



Turning down the noise

3. Make recommendations that are **personalized** to each user



1. Data and problem setting

- **Data:** Blogs ("the blogosphere")

THE HUFFINGTON POST

Blogger™

INSTAPUNDIT.COM

WORDPRESS.ORG

engadget

boingboing A Directory Of Wonderful Things

lifehacker

Wonkette

FANHOUSE

TPM

GOAL The New York Times Soccer Blog

- **Comparison:** other systems that aggregate blog data

Google™ Blog Search BETA

Y! buzz™ beta

digg™

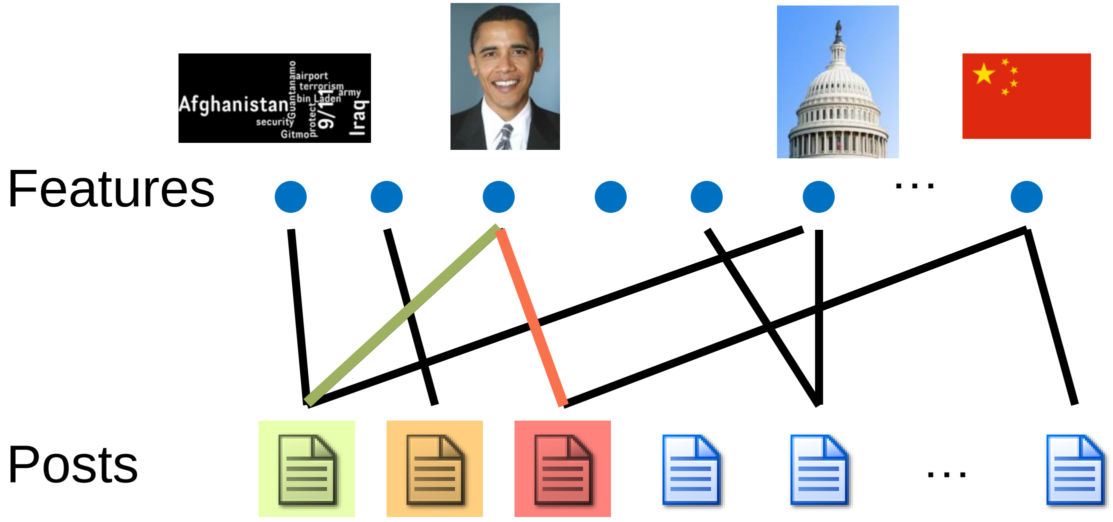
blogpulse™ a service of Nielsen BuzzMetrics

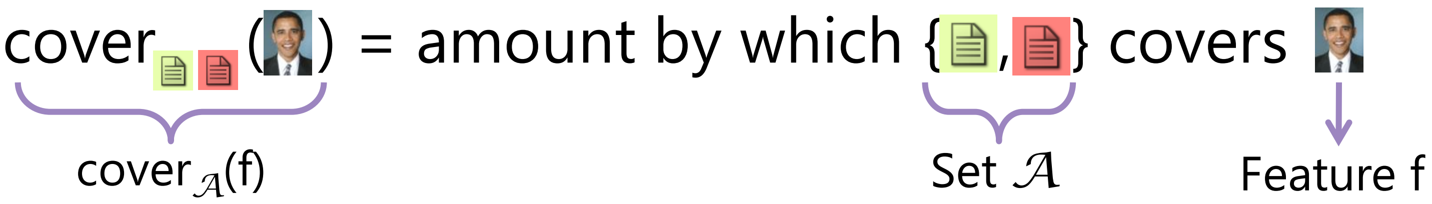
1. Data and problem setting

- **Low-level features**:
  Bags-of-words, noun phrases, named entities

- **High-level features:**
  Low-dimensional document representations, topic models

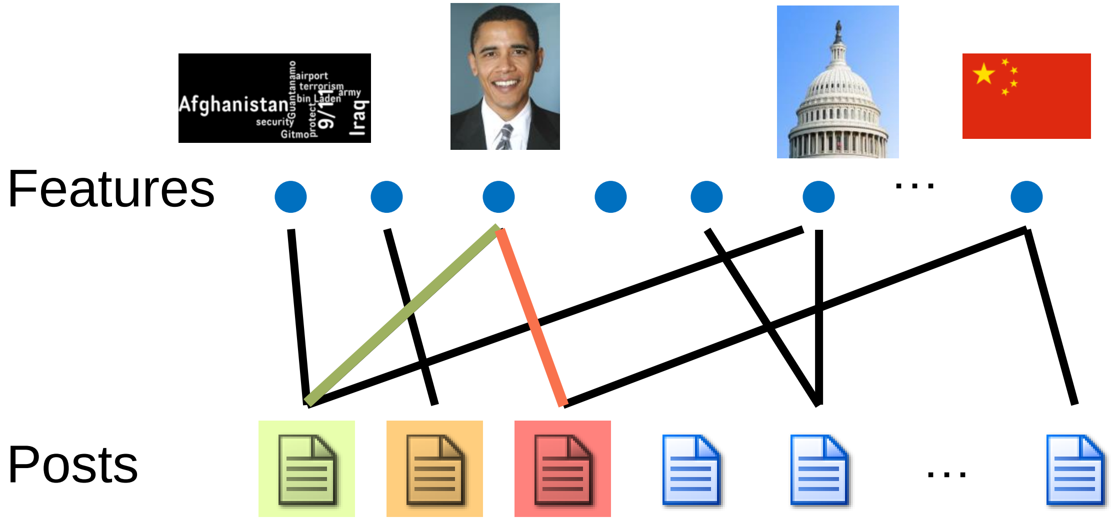
2. Maximize coverage





- We'd like to choose a (small) set of documents that maximally **cover** the set of features the user is interested in (later)

2. Maximize coverage



$$F(\mathcal{A}) = \sum_{f \in \mathcal{U}} w_f \cdot cover_{\mathcal{A}}(f)$$

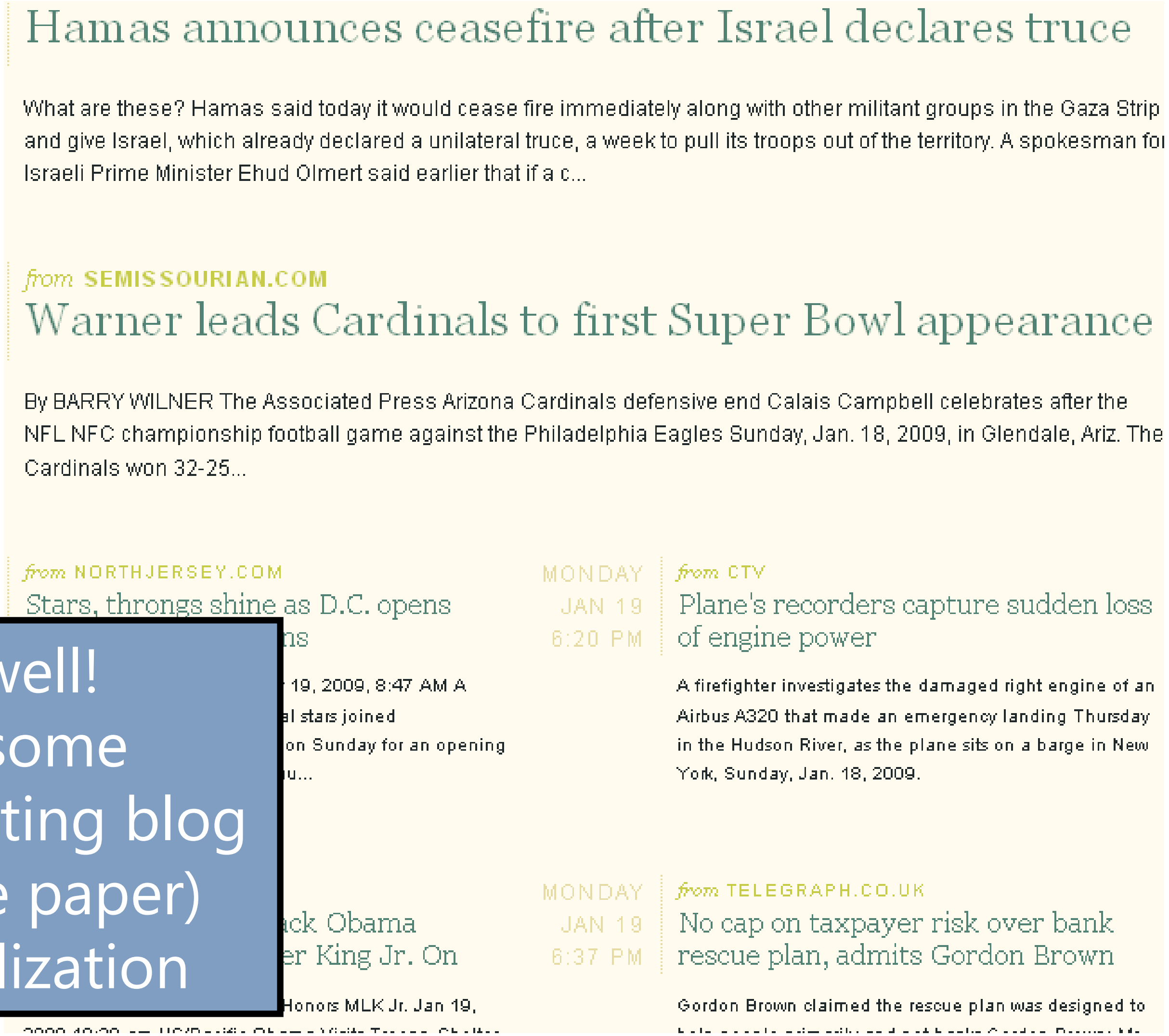
feature set

feature importance

coverage of feature by *A*

- Can be done (approximately) by selecting documents greedily (with an approximation ratio of (1 – 1/e)

2. Maximize **coverage**



Works pretty well!
(and there are some comparisons to existing blog aggregators in the paper)
But – no personalization

3. Personalize

$$F(\mathcal{A}) = \sum_{f \in \mathcal{U}} \pi_{u,f} \cdot w_f \cdot cover_{\mathcal{A}}(f)$$

feature set

personalized feature importance

coverage of feature by *A*

- Need to learn weights for each user based on their **feedback** (e.g. click/not-click) on each post

$\pi_{u,1}$ $\pi_{u,2}$ $\pi_{u,3}$ $\pi_{u,4}$ $\pi_{u,5}$

$\pi_{v,1}$ $\pi_{v,2}$ $\pi_{v,3}$ $\pi_{v,4}$ $\pi_{v,5}$

3. Personalize

$$F(\mathcal{A}) = \sum_{f \in \mathcal{U}} \pi_{u,f} \cdot w_f \cdot cover_{\mathcal{A}}(f)$$

feature set

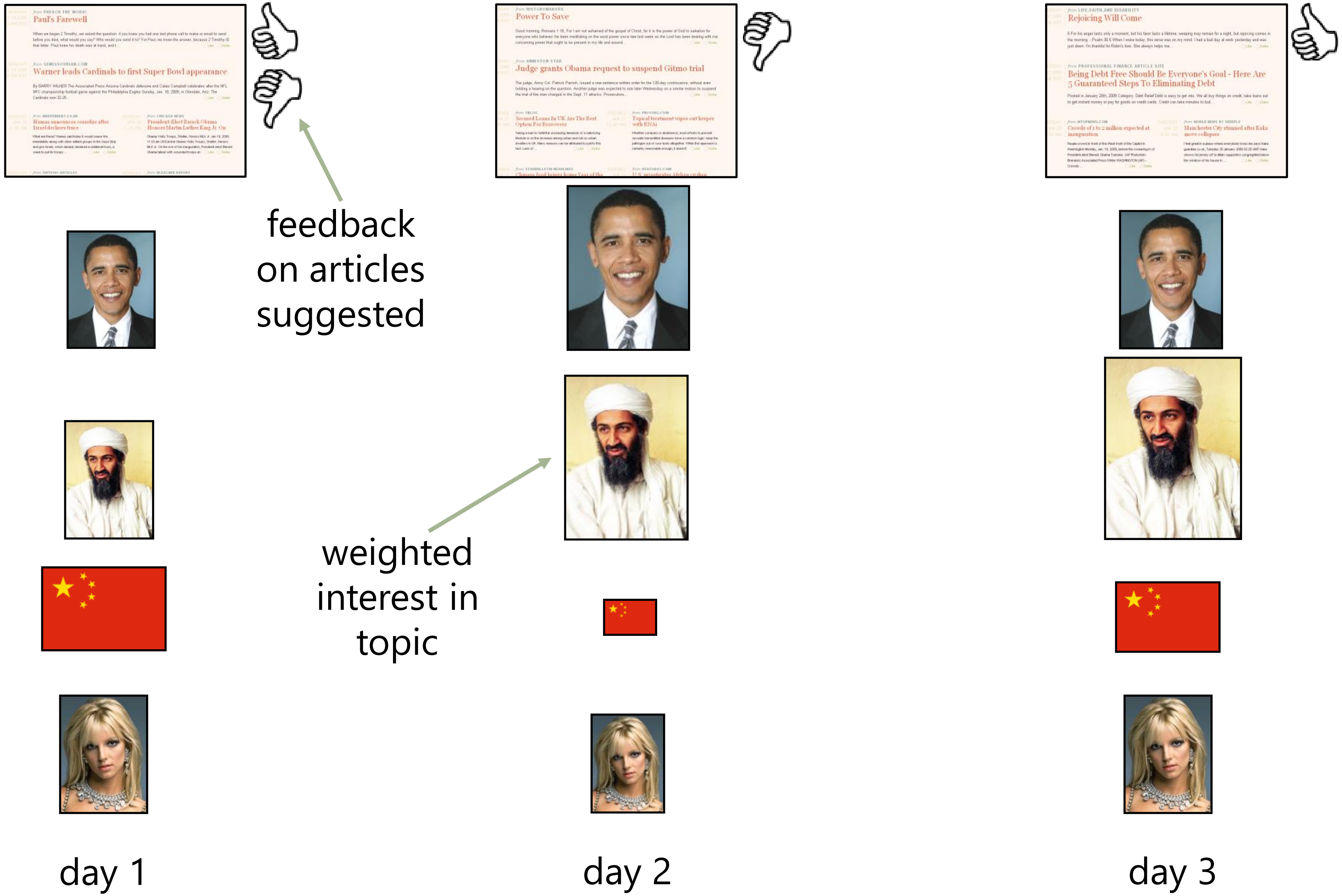
personalized feature importance

coverage of feature by *A*

- Need to learn weights for each user based on their **feedback** (e.g. click/not-click) on each post

- A click (or thumbs-up) on a post **increases** $\pi_{u,f}$ for the features *f* associated with the post
- Not clicking (or thumbs-down) **decreases** $\pi_{u,f}$ for the features *f* associated with the post

3. Personalize



Summary

- Want an algorithm that **covers** the set of topics that each user wants to see
- Articles can be chosen **greedily**, while still covering the topics nearly optimally
- The topics to cover can also be **personalized** to each user, by updating their preferences in response to user feedback
- **Evaluated** on real blog data (see paper!)

Recently...

We’ve looked at three features to handle the properties unique to online advertising

1. We need to handle **budgets** at the level of users and content (Matching problems)
2. We need algorithms that can operate **online** (i.e., as users arrive one-at-a-time) (AdSense)
3. We need to algorithms that exhibit an explore-exploit tradeoff (Bandit algorithms)

Questions?

Further reading:

- Turning down the noise in the blogosphere
(by Khalid El-Arini, Gaurav Veda, Dafna Shahaf, Carlos Guestrin)

http://www.select.cs.cmu.edu/publications/paperdir/kdd2009-elarini-veda-shahaf-guestrin.pptx
http://www.cs.cmu.edu/~dshahaf/kdd2009-elarini-veda-shahaf-guestrin.pdf