CSE 258 – Lecture 14

Web Mining and Recommender Systems

Ten minutes of tensorflow

Tensorflow

Tensorflow (other than doing deep learning and all that stuff) is a library to **specify learning algorithms at a high-level**

This allows you to specify the **objective** (e.g. regularized mean squared error), without having to worry about the details of the solution (e.g. computing derivatives and gradient descent)

Tensorflow

e.g. minimize the MSE:

(http://jmcauley.ucsd.edu/code/tensorflow.py)

$$\begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_1 & \cdots & b_n \end{bmatrix} = \begin{bmatrix} a_1+b_1 & a_1+b_2 & a_1+b_n \\ & & \\ a_n+b_1 & \cdots & a_n+b_n \end{bmatrix}$$

$$\min_{\theta} \frac{1}{N} \sum_i (y_i - X_i \cdot \theta)^2$$

Tensorflow

regularized MSE

(http://jmcauley.ucsd.edu/code/tensorflow.py)

Tensorflow

l1 – regularized MSE

(http://jmcauley.ucsd.edu/code/tensorflow.py)

Tensorflow

logistic regression with only positive parameters

(http://jmcauley.ucsd.edu/code/tensorflow.py)

$$\sum_{y_i=1} \log \sigma(X_i \cdot \theta) + \sum_{y_i=0} \log(1-\sigma(X_i \cdot \theta))$$

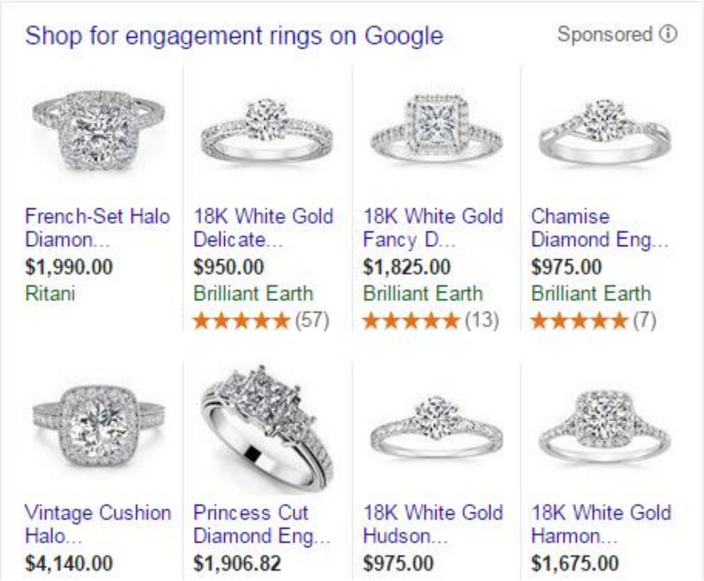
CSE 258 – Lecture 14

Web Mining and Recommender Systems

Algorithms for advertising

Classification

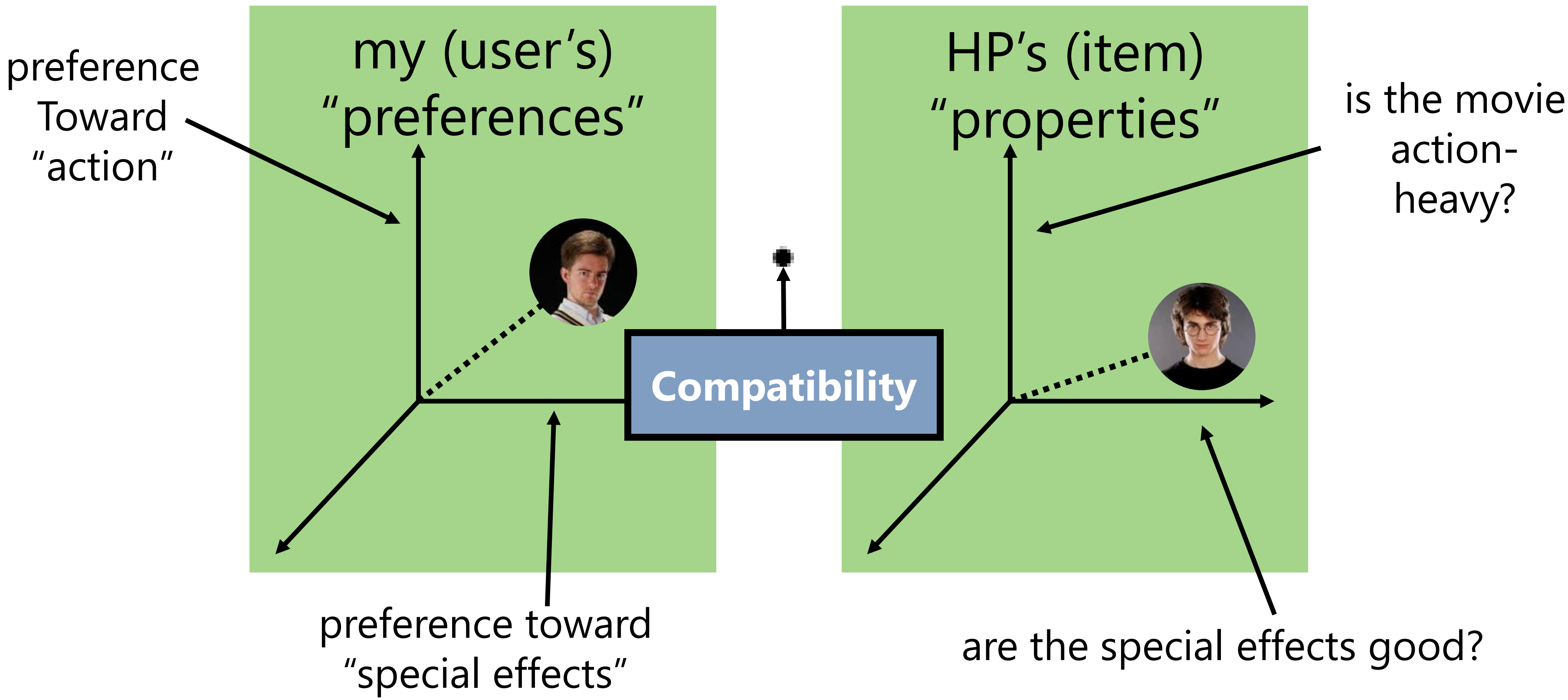
Predicting which ads people click on might be a **classification** problem



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

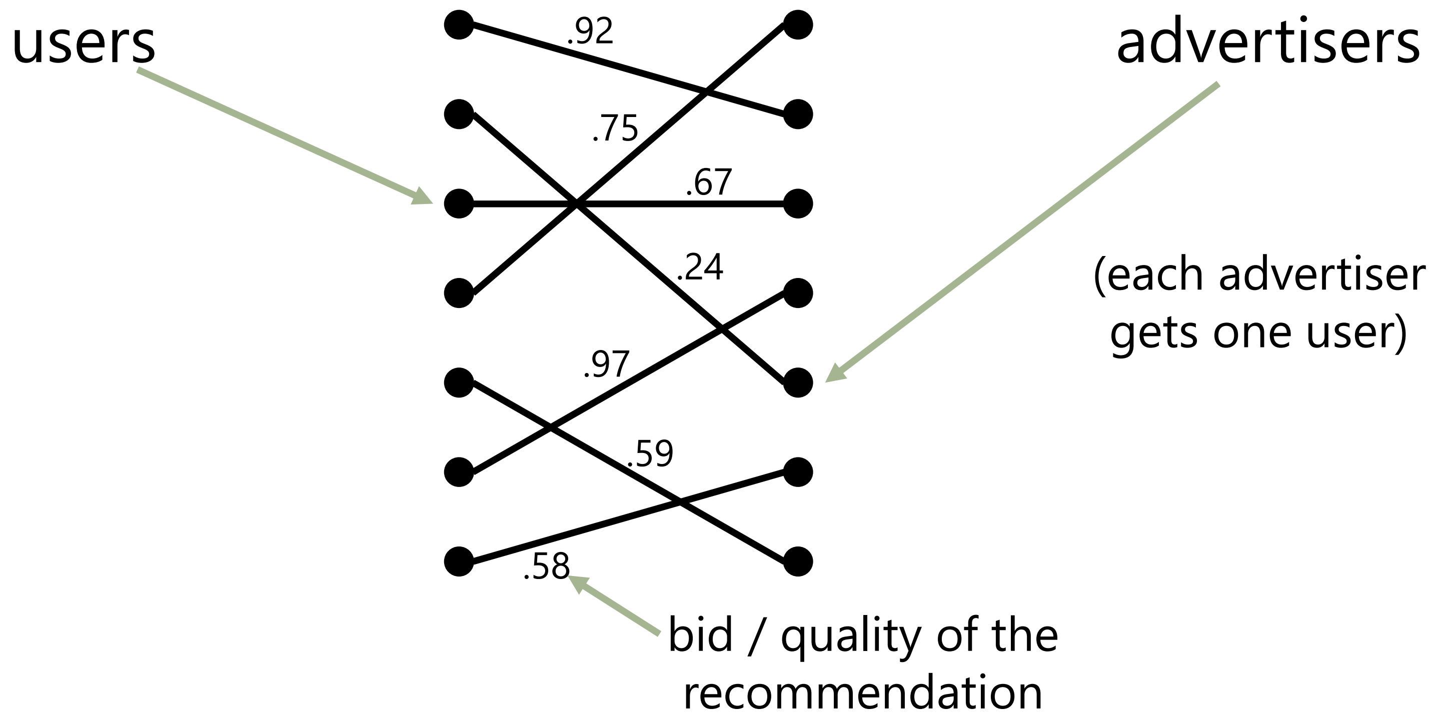
- Is the page a user is currently visiting particularly relevant to a particular type of content?
- Even if we have a good model of the user, recommending them the same type of thing over and over again is unlikely to succeed – nor does it teach us anything **new** about the user

- 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**

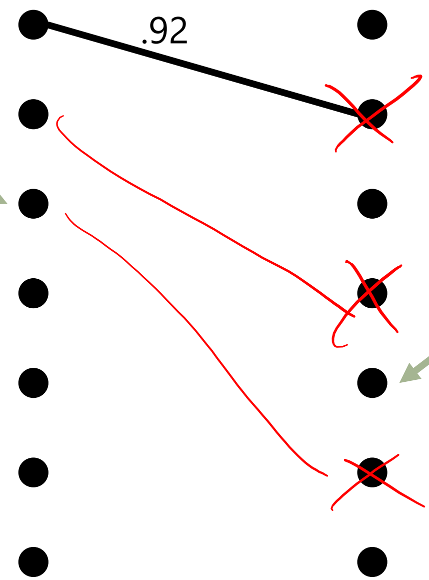


Advertising

So, ultimately we need

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

users



advertisers

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

CSE 258 – Lecture 14

Web Mining and Recommender Systems

Matching problems

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:

= bid placed on u by a

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

$p(u \text{ clicks on } a) = \sigma(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i)$

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 add to be shown exactly once **so as to maximize this score** (bids, expected profit, probability of clicking etc.)

users ad shown to u

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

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 add 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

$$\forall u \ \sum_a A_{u,a} = 1$$

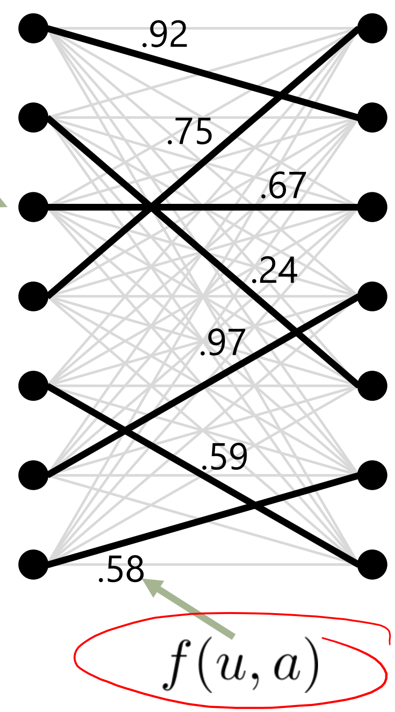
every user sees one ad

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

users

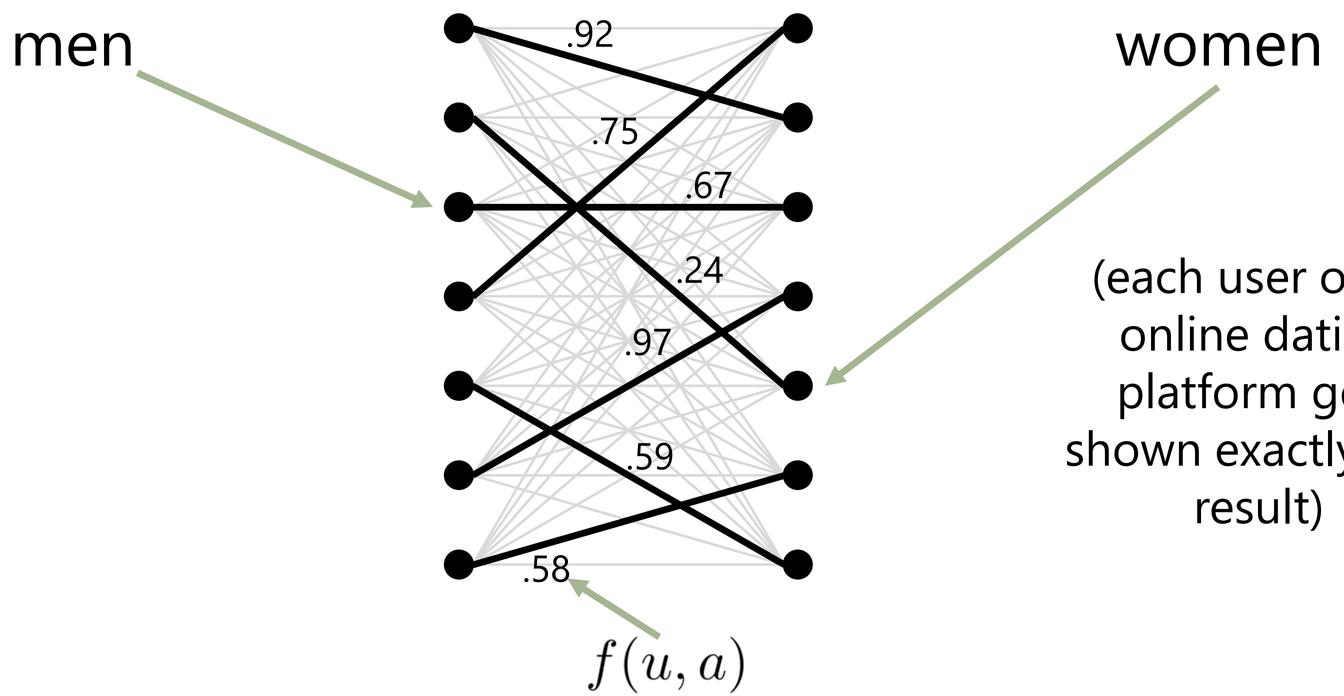


ads

(each advertiser gets one user)

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**



(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

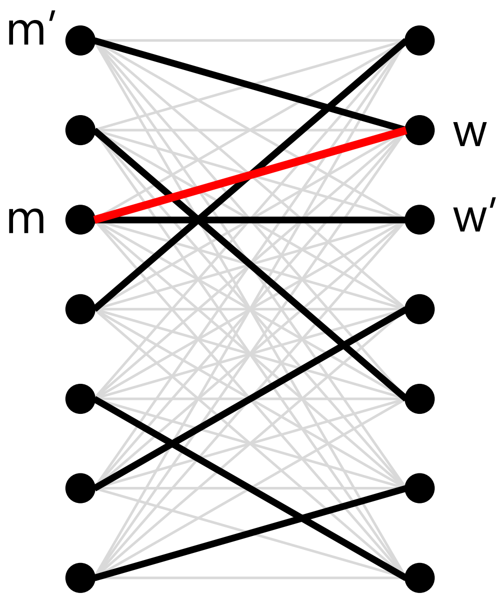
- Marriages are monogamous, heterosexual, and everyone gets married

(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))

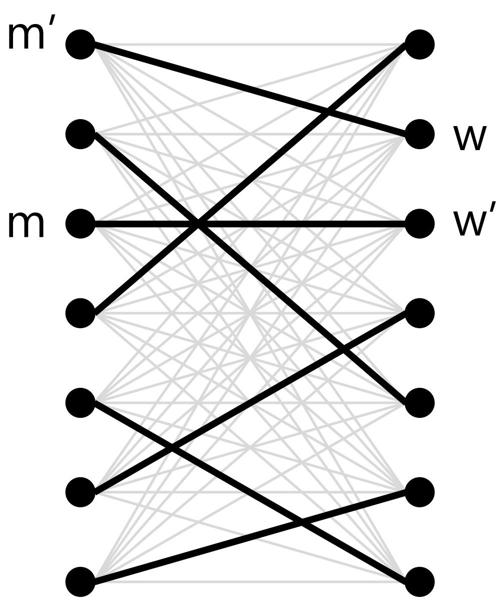
and

- 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

Bipartite matching

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

— every iteration, a new proposal happens

— we'll eventually run out

— happiness increases, or nothing happens

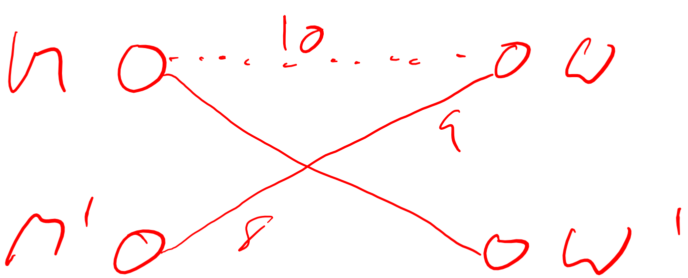
Bipartite matching

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The solution is stable

Assume soln. not stable



m would have proposed to w before w'

① w not available and did not switch when asked
→ w with someone better

② w switched after m proposed
but w wouldn't switch
↓
someone worse

Bipartite matching

The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

- The solution is O(n^2)

– n^2 total proposals possible

– every iteration tries one proposal

couldn't be better

Bipartite matching – extensions/improvements

Can all of this be improved upon?

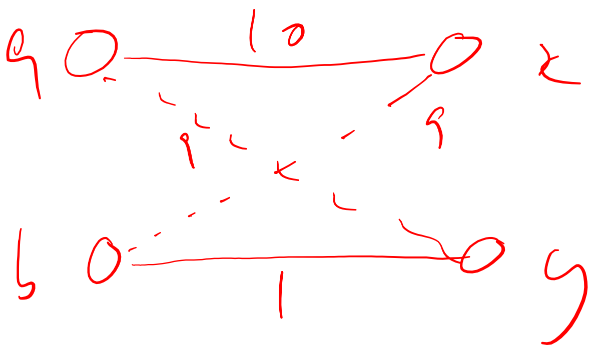
1) It’s not optimal

- 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

Bipartite matching – extensions/improvements

Can all of this be improved upon?

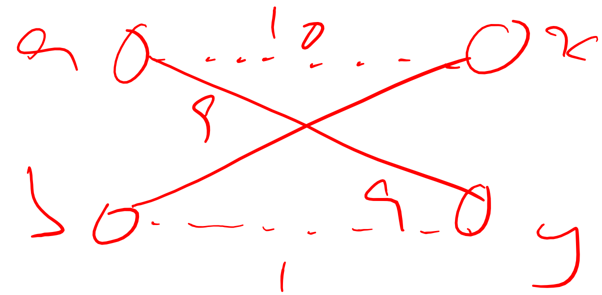
1) It's not optimal



score = 11

optimal score = 18

stable, but not optimal



optimal but not stable

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

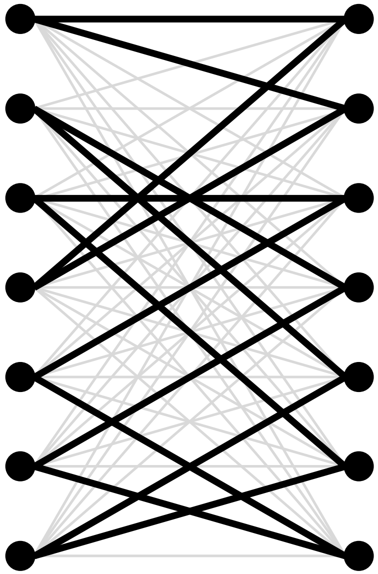
- To get a **truly optimal** solution, there's a more complicated algorithm, known as the "Hungarian Algorithm"
  - But it's O(n^3)
- And really complicated and unintuitive (but there's a ref later)

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)



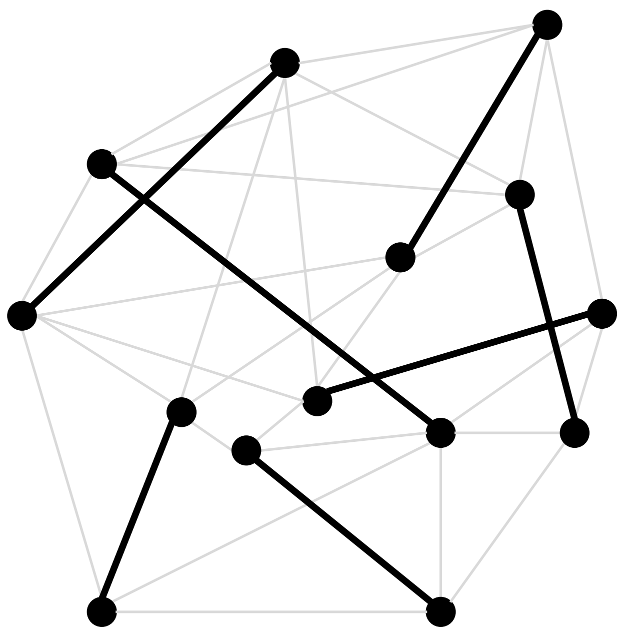
- Each advertiser may have a fixed budget of (1 or more) ads
- We may have room to show more than one ad to each customer

- 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

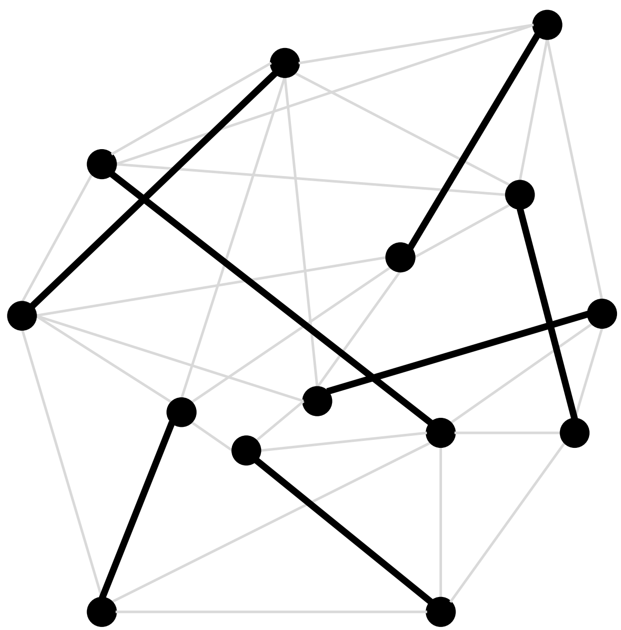


- This version of the problem is know as **graph cover** (select edges such that each node is connected to exactly one edge)
- The algorithm we saw is really just graph cover for a bipartite graph
- Can be solved via the "stable roommates" algorithm (see refs) and extended in the same ways

Bipartite matching – extensions/improvements

Can all of this be improved upon?

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



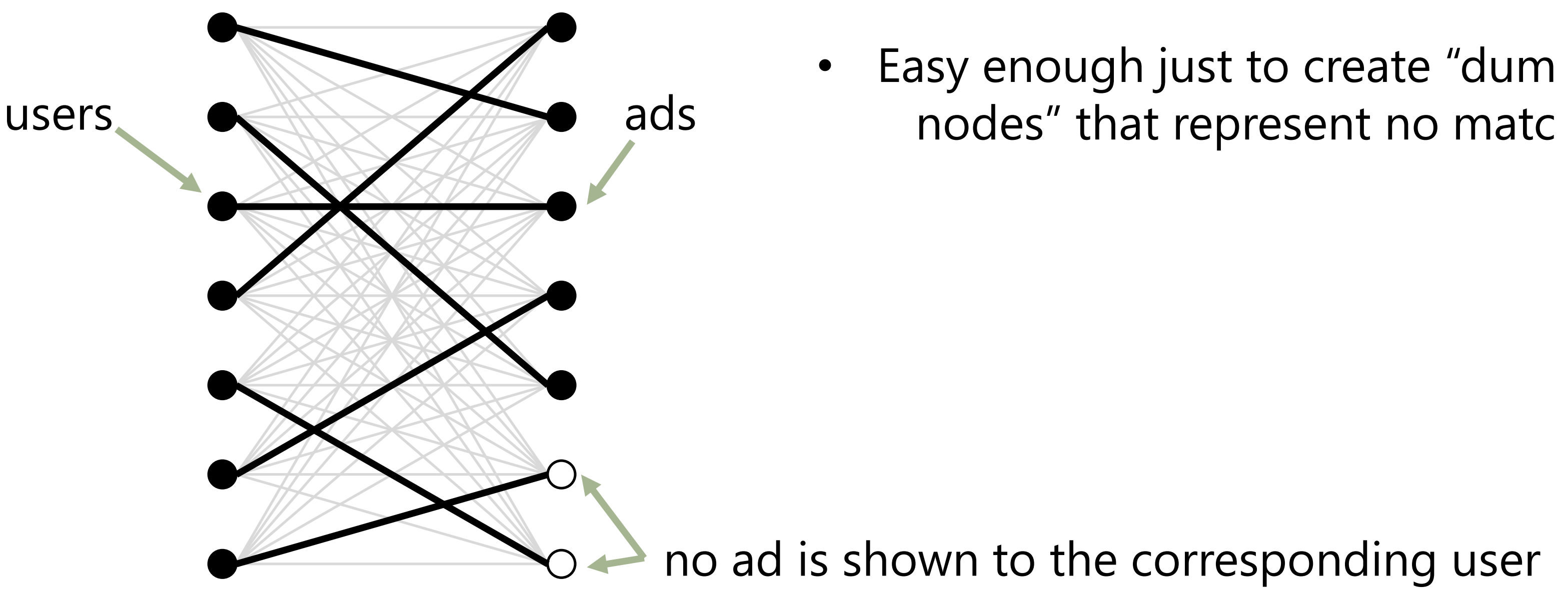
- This version of the problem can address a very different variety of applications compared to the bipartite version

- 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

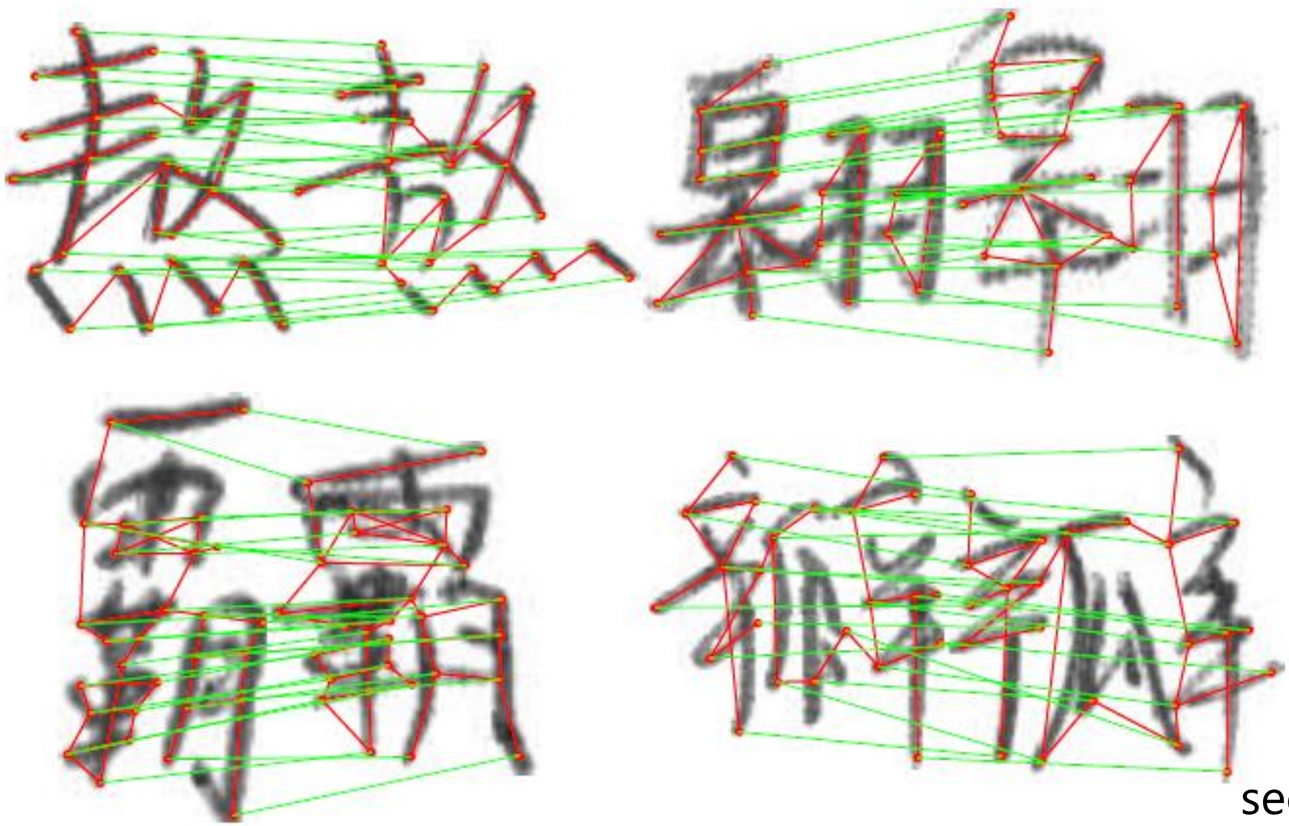
Why are matching problems so important?

- 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



see e.g. my thesis

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:

$$\sum_{u,v} f(\underbrace{u, v}_{\text{pair of residents}}, \underbrace{hospital(u), hospital(v)}_{\text{hospitals to which they're assigned}})$$

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

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

Assignment 1: What worked and what didn't?

Helpfulness:

- rating, $\overline{\text{rating} - \text{rating}}$
- # reviews
- past helpfulness rate user/item
- outOf
- length, av. word length, #!? capitals

Assignment 1: What worked and what didn't?

- Readability index
- age of item
- MAE
- dropping low n Helpful / outof

  ↳ outof

Support Vector Regression

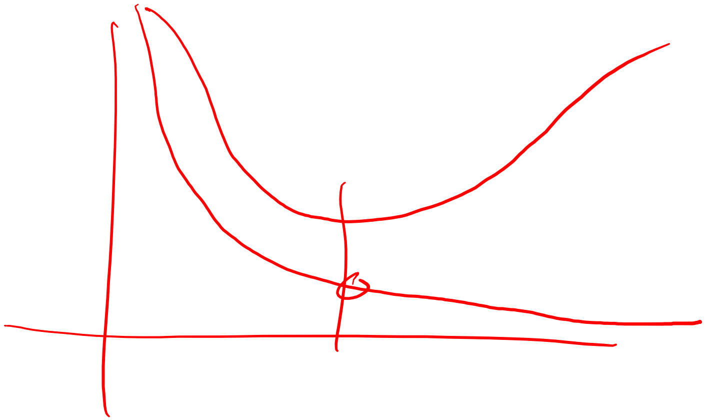
Assignment 1: What worked and what didn't?

Rating Prediction

$$\alpha + \beta_i + \beta_u + \gamma_i \cdot \gamma_u$$

$$\lambda = 7.01$$

early stopping



Assignment 1: What worked and what didn't?