Ranking and Suggesting

Joint work with Milan Vojnovic, Dinan Gunawardena, Peter Marbach
Outline

1. Example: Tag suggestion
2. Problem: Approval voting with feedback
3. User model
4. Algorithms
   - Most popular
   - Move to set
   - Frequency move to set
5. Conclusion
Example: Tagging
Example: Tagging

Tags:

Suggestions:

news, bbc, world
Example: Tagging

Tags:

news, bbc, current

Suggestions:

news, bbc, world
What does the system see?

- For each bookmark:
  - Users who bookmarked
  - Tags used
- Aim to learn relevant tags for page
  - Improve search
  - Suggest other pages of interest

Example:

- rjrc3: news, bbc, current
- tall-man: bbc, world
- george: current, news, world
Approval Voting with suggestion

- At each voting event system suggests list of items
- User selects set items to vote for.
  - Can be a mix from list suggested and others
- Votes cast are then fed back into algorithm to select items at next event.
What are the aims of suggestions?

1. Learn true popularity ranking
2. Improve speed of learning
3. Suggest items relevant to users
Each item has probability of selection $r_i$ with out suggestion
Assume $r_i < r_j$ for $j < i$
User imitates with probability $p_s$
Top most popular

- Rank items by number of votes
- Select current most popular items to suggest

<table>
<thead>
<tr>
<th>Item</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbc</td>
<td>120</td>
</tr>
<tr>
<td>news</td>
<td>97</td>
</tr>
<tr>
<td>world</td>
<td>56</td>
</tr>
<tr>
<td>current</td>
<td>40</td>
</tr>
<tr>
<td>useful</td>
<td>23</td>
</tr>
<tr>
<td>resource</td>
<td>10</td>
</tr>
</tbody>
</table>
Modelling

1. Under the user model the number of votes per items and suggestion set evolve according to a Markov chain.

2. At each step the probability of item i being voted for is

\[(1-p_s) r_i \quad \text{if not suggested}\]
\[(1-p_s)r_i + p_s r_i/(\sum r_j) \quad \text{if suggested}\]
What effect does the reinforcement have?

For given \( r \) if \( p_s > p_{\text{crit}} \) where

\[
p_{\text{crit}} = \min_{S \neq \{1, \ldots, r\}} \left\{ p : \left( \sum_{i \in S} \gamma_i \right) \left( \frac{\max_{i \in S \setminus S} \gamma_i}{\min_{i \in S} \gamma_i} - 1 \right)^{\frac{1}{r}} < \frac{p}{1 - p} \right\}
\]

the observed limit order is not necessarily the true order. Further more, given a \( p_s > 0 \) there exists an \( r \) such that the true order may not be found.
Proof method

1. It can be shown that after a point the suggestion settles down
   • i.e. small chance of a new item coming into set

2. We then calculate which suggestions sets are “stable sets
   • Where rate of increase of those in the suggestion set is larger than all those not in it in the user model.
Move to set

- Current suggestion set is state
- If an item is picked not currently on suggestion set is picked it is added
- A current item removed uniformly at random

```
bbc, news, current
world
bbc, world, current
```
The suggestion set is related
The suggestion set is related
The suggestion set is related
What does this tell us?

1. The long run proportion of time an item is in the suggestion set is increasing with $r_i$.

2. Every item is suggested infinitely often.

3. Removes the possibility of the suggestion set getting stuck.

4. In the case of a single suggestion this is equivalent to “show most recent selection”
Guaranteed to be correct but can suggest badly

1. The long run ranking given from all the votes is the same as the true popularity ranking

2. But the proportion of time a suggestion set is used is

\[ \pi(S) \propto \prod_{i \in S} r_i \]

4. This means bad suggestions are relatively frequently made.
Frequency move to set

- Two counts kept for items:
  - Total number of votes
  - Number of votes while not suggested
- For suggestion set selection rank by votes while not suggested
- Select current top
- Total votes used for ranking

<table>
<thead>
<tr>
<th>Item</th>
<th>Total Votes</th>
<th>Votes off suggestion</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbc</td>
<td>120</td>
<td>30</td>
</tr>
<tr>
<td>news</td>
<td>97</td>
<td>20</td>
</tr>
<tr>
<td>world</td>
<td>56</td>
<td>26</td>
</tr>
<tr>
<td>current</td>
<td>40</td>
<td>22</td>
</tr>
<tr>
<td>useful</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>resource</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
Again behaviour governed by a Markov chain

1. The state space now is the total number of votes and the number of votes received while not suggested.

2. The suggestion set only depends on the current state giving the Markov property required.

3. Used to study long run behaviour.
In the long run who can be suggested?
In the long run who can be suggested?

Only sufficiently popular items!

\[ s_i = \begin{cases} 
1 - (1 - \frac{s}{|C|}) h_{|C|}(r) & i \in C \\
0 & \text{o.w.} 
\end{cases} \]

Item i in the competing set iff: \( r_i > (1 - \frac{s}{i}) h_i(r) \)
c = 7 competing items

8 non-competing items
The true ranking is found after a time

1. In the long run the ranking from the total votes is the true popularity ranking.

2. The reinforcement effect is mitigated by the suggestion constantly changing... but the competing set phenomenon means the suggestions are relevant.
How quickly is the ranking found?

1. Theoretical understanding results for this is still on going.

2. Some simulations have been carried out showing frequency move to set to be fastest.
Conclusion

1. Reinforcement can lead to bad behaviour and care needs to taken in algorithm design.

2. Many open questions:
   - A better algorithm?
   - How to deal with community structure?
   - Is there a better user model?
   - How quick is the ranking achieved?