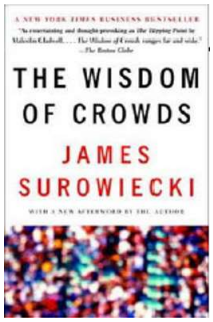
This is lecture 1 of 2 in a series on collaborative filtering and ensemble methods. We will start by examining the concept of crowd intelligence.

The Wisdom of Crowds

- Under some conditions, group IQ > individual IQ
- Examples:
  - Guess the weight of a bull: 542,9 kg (543,4 kg)
  - Guess the number of sweets in a jar: 871 (850)
  - Find the way out of a maze:
    - First individual attempt: 34,3 turns on average
    - Second individual attempt: 12,8 turns on average
    - Majority turn decision: 9 turns on average
    - -> We see that group decision is better than best individual



Under some conditions, we observe that the collective intelligence of a group is higher than that of the best individual. Here are some examples from the book “The wisdom of crowds”:

A group of people were asked to guess the weight of a bull at a country fair. The average guess was only half a kilo off. Similarly, when guessing the number of sweets in a jar, the average guess was much closer than any individual guess. When trying to find their way out of a maze, even after two tries, the best individual still had to take more turns than a collective group. What all of

these examples show us is that in these cases, the group decision is better than the best individual.

Some other examples

- Online user reviews and product recommendations
- Google search - based on collective linking preferences (pageRank)
- eBay edge: collective reputation building
- Netflix, Spotify, Facebook etc.
- Voting
  - How a user's preference is gauged

Some other examples include things like online user reviews and product recommendations. We are more likely to trust a product's review if multiple people vouch for it. It is also less likely to be a scam, than if only one person is shilling for it. Similarly, Google search is more likely to place websites higher up on their results based on how many other webpages are linking to it. In short, pages are ranked based on collective linking preferences. There are many other determining factors when it comes to Google search, but that is a discussion for another day. We also see this on sites like eBay: sellers gain

reputation and trust from their buyers, which again makes them less likely to be shills or scams. When you are recommended a movie or show from Netflix, songs on Spotify and posts on Facebook, the underlying algorithms are based on the activity of a subgroup with similar preferences.

Going forward, we will use the term “Voting”. Votes in this context is a measure of whether a user liked something or not. This can be measured in different ways depending on the context. On Netflix, Facebook and Spotify you got “like” and “dislike” buttons. This is quite intuitive, but there are also less obvious ways to measure user preference. You could for instance look at the time a user spends on a certain activity, and make assumptions. YouTube does this quite effectively, and will recommend videos based on your watch history, even if you didn’t interact with the video through likes or dislikes.

Memory-based algorithms for Collaborative Filtering (Breese et al, UAI98):

- $v_{i,j}$= vote of user *i* on item *j*
- I_i = items for which user *i* has voted
- Mean vote for *i* is

$$\overline{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

- Predicted vote for "active user" ***a*** is weighted sum

$$p_{a,j} = \overline{v}_a + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \overline{v}_i)$$

normalizer — weights of *n* similar users

Now we will take a look at some algorithms for Collaborative Filtering. The basic formula is based on a sum of weighted votes. Remember that a vote can be more than a direct interaction. We compute the mean vote for the user *i* for every item j as such. In short, we are creating a profile for our user. We are then able to use this profile to predict their vote, or interest or whatever, based on the votes of similar users. The other users' votes are weighted based on similarity. We may also introduce a normalization factor to account for any deviations and outliers.

How to compute weights

- K-nearest neighbor

$$w(a,i) = \begin{cases} 1 & \text{if } i \in \text{neighbors}(a) \\ 0 & \text{else} \end{cases}$$

- Pearson correlation coefficient (Resnick '94, Grouplens):

$$w(a,i) = \frac{\sum_j (v_{a,j} - \overline{v}_a)(v_{i,j} - \overline{v}_i)}{\sqrt{\sum_j (v_{a,j} - \overline{v}_a)^2 \sum_j (v_{i,j} - \overline{v}_i)^2}}$$

- Cosine distance

$$w(a,i) = \sum_j \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}}$$

The weighted votes of similar users can be computed in many ways. One way you should be familiar with by now is the K nearest neighbor. This is quite simple in that either you are part of a neighborhood, or not. The weights thus become zero and one. We also have more advanced methods like the Pearson correlation coefficient and Cosine distance.

How to compute weights 2

- Cosine with "inverse user frequency" $f_i = log(n/n_j)$, where n is number of users, n_j is number of users voting for item j

$$w(a,i) = \frac{\sum_j f_j \sum_j f_j v_{a,j} v_{i,j} - (\sum_j f_j v_{a,j})(\sum_j f_j v_{i,j}))}{\sqrt{UV}}$$

where

$$U = \sum_j f_j (\sum_j f_j v_{a,j}^2 - (\sum_j f_j v_{a,j})^2)$$

$$V = \sum_i f_j (\sum_i f_j v_{i,j}^2 - (\sum_i f_j v_{i,j})^2)$$

We also have the concept of inverse user frequency, which basically means that universally liked items as less useful than less common items when finding similar users. If a lot of users like a lot of the same thing, it becomes harder to find the common factor for the subgroups. We can account for this by using the log function; the frequency for item j thus becomes log number of users divided by the number of users voting for item j. This can also be combined with the previous methods, like the Cosine distance as seen here.

Evaluation

- split users into train/test sets
- for each user *a* in the test set:
  - split *a's* votes into observed *(I)* and to-predict *(P)*
  - measure average absolute **deviation** between predicted and actual votes in *P*
  - predict votes in *P*, and form a **ranked list**
  - assume (a) utility of *k*-th item in list is $max(v_{a,j}-d,0)$, where *d* is a "default vote" (b) probability of reaching rank *k* drops exponentially in *k*. Score a list by its expected utility R_a
- average R_a over all test users

Let's close off by looking at evaluation. Start by splitting users into training and test sets. Then we loop over every user a in the test set. We split the active users votes into two groups: observed and to-predict. We measure the average absolute deviation between the predicted votes and actual votes. We predict the votes and form a ranked list. We assume A, that the utility of the k-th item in the list is the maximum of the difference between the user's vote and a default vote, and B, that the probability of reaching rank k drops exponentially in k. We score the list by its expected utility R of a.

Finally, we average R of a over all test users.