MS&E 235: Lecture 13
Prof. Ashish Goel
May 19, 2009
Notes by Rio Goodman

Lecture 13: More Recommendation Systems

In this lecture we will use PageRank to further discuss recommendation systems. The focus will be on a Netflix-type movie recommendation system, but the concepts can be applied to other areas of internet commerce.

Linear Algebra of Naïve PageRank

Since the set of edges, E, is essentially a matrix, it is natural to express the naïve PageRank equations into matrix form. Recall the original definition of naïve PageRank:

$$\pi_v = \sum_{w:(w,v)\in E} \left(\frac{\pi_w}{d_w}\right)$$

With the additional constraints $\sum_i \pi_i = 1$ and $\pi_i \geq 0$ for all i.

Now let us define an $N \times N$ matrix, A, with entries $A_{(w,v)}$, where N is the number of nodes (pages) in the network. The matrix entries, $A_{(w,v)}$, will represent edges in the following manner.

$$A_{(w,v)} = \begin{cases} 0 \text{ if } (w,v) \notin E \\ \frac{1}{d_w} \text{ otherwise} \end{cases}$$

Substituting this into the naïve PageRank equations, we get

$$\pi_v = \sum_w A_{(w,v)}\pi_w$$

In vector form, we can write this system as

$$\vec{\pi} = \vec{\pi}A$$

Notice that $\vec{\pi}$ is a $1 \times N$ PageRank vector. Thus we can see that the PageRank vector is actually the eigenvector of the edge matrix, A, whose eigenvalue is one. In fact there is a class of reputation systems that are PageRank-like, that are eigenvector-based reputation systems.

An even stronger observation is that $\vec{\pi}$ is the *unique* eigenvector of A. We can show this using the iterative method for computing PageRank that we learned in earlier lectures. Recall from lecture 10, that the error in the iterative method quickly decreases to zero, meaning the iterative method converges to the true solution. Therefore, if p were a solution and q were a solution, the iterative method could not converge to both p and q unless $p = q$.

Using PageRank for Recommendation Systems

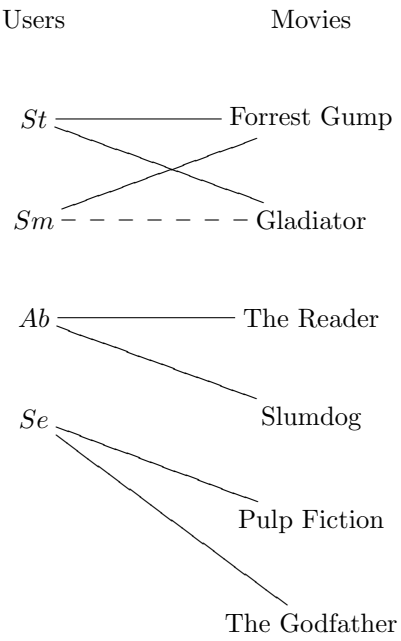
In class, three volunteers (St, Ab and Se) were asked to state their favorite movie. St chose Forrest Gump, Ab chose The Reader, and Se chose Pulp Fiction. A fourth volunteer, Sm, was then asked which movie, out of these three, was his favorite. Sm chose Forrest Gump.

Then the same process was repeated for a second round of movies. St chose Gladiator as his second favorite, Ab chose Slumdog Millionaire, and Se chose Godfather. Again, Sm agreed with St. So the results looked like this:

$$\text{St} \begin{cases} \text{Forrest Gump (Sm)} \\ \text{Gladiator (Sm)} \end{cases} \quad \text{Ab} \begin{cases} \text{The Reader} \\ \text{Slumdog} \end{cases} \quad \text{Se} \begin{cases} \text{Pulp Fiction} \\ \text{The Godfather} \end{cases}$$

The fact that Sm chose St's favorites both times suggests that there is a similarity in tastes. A user is more likely to enjoy movies that similar users have enjoyed. Users are similar if they enjoy the same movies. This recursive definition is reminiscent of PageRank. In fact, we will set up a network where we can use PageRank to assign a similarity index.

In PageRank, web pages were our nodes, but here we can assign nodes for every movie and every user. The graph will look something like this:



To generalize this, we could have a system with N users and M movies. Our example has movie titles, but we could also have a graph with movie categories instead (Comedy, Action, etc.). If we were to modify the graph to be a movie category graph, we would also need to assign two values to each edge, (α, β). Where α represents the number of times that a user enjoyed an action movie, and β represents the number of times the user did not enjoy an action movie. This formulation would share similarities with the multi-armed bandit problem.

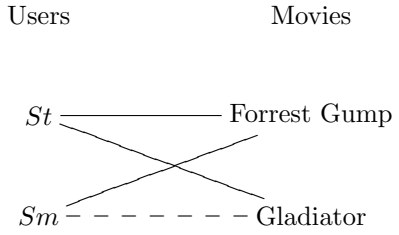
Let us formulate a model for determining a movie recommendation system using PageRank. If we consider the graph above where all the edges are bidirectional, we can find the PageRank of each node, both user and movie. But the PageRank will not be as useful as it was with web pages. If a movie has PageRank $\frac{3}{40}$, for instance, what does that mean? In the example, we are trying to determine which movies Sm is apt to like, based on the other users' favorites, and the fact that Sm likes Forrest Gump. We are currently unable to do this because the PageRank values have nothing depending on Sm specifically.

To modify PageRank to give us an index specifically relevant to Sm, we could use the personalized PageRank algorithm that was presented in earlier lectures. In personalized PageRank, instead of randomly jumping to a node in the graph, we instead randomly jump to a specific node in the graph. In this case, we will reset to Sm with probability ϵ.

If we run personalized PageRank as described, we will find that The Reader, Slumdog, Pulp Fiction and The Godfather have no links to or from Sm. Thus their personalized PageRank will be zero. We have already

added a link to Pulp Fiction, so we could have predicted that Sm would most likely enjoy Gladiator.

Our graph will reduce to:



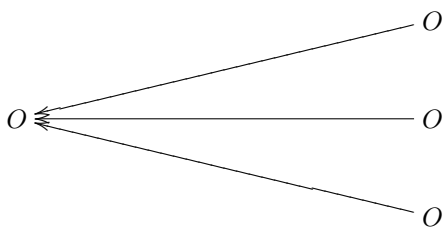
An analogous routine could be used to find similar products (for the Amazon recommendation system, for instance).

Since the PageRank of movies and the PageRank of users have different meanings, let us denote user v's PageRank as a_v, and movie w's PageRank as h_w. We then have the following naïve PageRank equations. Where we again use matrix notation for the edges (from the beginning of this lecture).

$$h_w = \sum_{v \in \text{Users}} \frac{A_{(v,w)} a_v}{d_v}$$

$$a_v = \sum_{w \in \text{Movies}} \frac{A_{(w,v)} h_w}{d_w}$$

PageRank was not the first eigenvector-based system for online ranking. The first method was called HITS and was very similar to the above formulation. HITS was based on traversing the information in online directories such as Yahoo!, which at the time was the best way to find information on the internet. The directories then had links to specific pages where the information resided. The graph for such a system would have information (or authorities) and directories (or hubs) rather than users and movies.



The downfall of this approach to web search was that personalized PageRank must be recomputed for every keyword. Doing this in real time is computationally expensive. Since Google and other web searches have become so popular, the need for online directories has vanished, and this system is no longer used.

In PageRank, a page with a lot of links on it contributes less of its PageRank to the pages it points to (the contribution is divided by d_v). In the hubs and authorities setting, this formulation does not make as much sense, and in the original formulation of HITS, the d_v was eliminated from the equations altogether. Another way to formulate the equations is to divided by $\sqrt{d_v}$. The only difference in the iterative method for computing these pseudo-PageRanks is that if the contributions are not being divided by anything, the PageRank will increase at every iteration. Thus the PageRank must be normalized at each iteration by dividing through by the sum of all the PageRanks.

The iterative equations for such a system will look like this.

$$h_w^{(t)} = \sum_{v \in \text{Users}} A_{(v,w)} a_v^{(t)} \tag{1}$$

$$a_v^{(t)} = \sum_{w \in \text{Movies}} A_{(w,v)} h_w^{(t-1)} \tag{2}$$

The algorithm for iteratively computing the PageRanks will be:

Initial:

$h_w^{(0)} = \frac{1}{M}$, $t = 0$

Repeat:

$t \leftarrow t + 1$
Apply equation (2)
Apply equation (1)
Normalize $h_w^{(t)}$ to sum to 1

The same properties will apply to this method as to our original PageRank formulation.

In the next lecture we will learn more about network structure.