Mengying Zhang, Yuying Li
SI 608 Networks
Final Project
12/10/2018

Network and non-Network based Approach for Music Recommendation

Abstract: Exploiting user preference and having a personalized recommendation can help online retailers attract customers to spend more time on the site and increase the likelihood of purchasing products. In this paper, we applied network and non-network approaches for song recommendation. We use Collaborative Filtering(CF) for non-network approach and Breadth First Search(BFS) and Hitting Time(HT) for network approach. To resolve data sparsity, we applied k-means clustering to retrieve higher representation of music songs and use Collaborative Filtering to find who are similar to who and which item is similar to which. Early pruning is adopted to reduce computational effort. Network approaches are based on first representing the user-item as one mode projection of bipartite graph and use BFS with path length discount and HT with Multiwalk Monte Carlo α-Terminating Random Walk to estimate the real hitting time. The result shows that half of the time CF is unable to find very similar users with similar songs while BFS and HT can reach a surprisingly well recall rate. The highest recall rate is achieved with BFS.

I. Introduction

One main advantage of using online retailers such as Amazon and eBay is to be able to see all product reviews and easily find related items. As more information become available to the customer, people's purchasing decisions have been influenced a lot by what the merchant displays to the customer. One strategy specifically won a huge marketing success which is personalized recommendation. If the system can improve the recommendation, online shoppers can potentially spend more time on the site, thus increasing the likelihood of purchasing action. In this project, we look at different strategies for building recommendation system for songs using network and non-network based approaches. Our non-network approach uses Collaborative Filtering(CF) and our network based approaches involve representing the user item as bipartite network and apply Breadth First Search(BFS) and Hitting Time(HT) to recommend songs to the user.

II. Research Questions

Nowadays, lots of non-network based approaches have been explored for a recommendation system, mostly implemented based on machine learning. However, we would like to find out a way to achieve high accuracy of a recommendation system without machine learning. Therefore, we bear in mind of the following three research questions:

- Whether network based model of non-network based model performs better in a recommendation system?
- Which network based model for recommendation system prevails?
- What accuracy(recall) can a network-based model reach?

III. Related Work

There has been many researches done in recommendation field. Two most popular approaches are Content-based Filtering and Collaborative Filtering(CF). Content based filtering matches product information to the user preference profile and CF makes uses of how similar users would rate the product. One study looks at heterogeneous information network that allows to exploit different attributes of connected edges labeled by meta-paths and use PathSim to obtain similarities and embed into item-based CF(Zhang, Ganchev, Nikolov, Ji, & Odroma 2017). Another paper proposed two new networks based similarity measure HB1 and HB2 to solve cold start problem and few neighbour problem in traditional CF. We have adopted similar approach when obtaining prediction listening counts but our similarity measure stays as cosine similarity while they use Jaccard for similarity measure(Singh, Patra, & Adhikari 2015). New research frontier also touches on using deep learning to recommend items to users such as Wide & Deep Learning proposed by Google AI lab(Cheng,Heng-Tze, et al 2016).

IV. Dataset

In order to implement a recommendation system, we decide to do it based on the similarity between songs, and the user listening history. The data we use are divided into two parts, the song data and user data.

- Song Data: Temporarily we use 10000 song records. Each song record contains metadata from different dimensions of a song, such as duration, tempo, artist_hotness, key, mode, loudness. The listed metadata here are ones we consider in our project.

- User Taste Data: In total, we have 48,373,586 records of 1,019,318 users' listening history. In first stage, we filtered the records to only those 10,000 songs, since some songs in the listening history is not among the 10000 songs. This stage downsized number of users to 418,252 with 772,661 records. Each record contains [user, song, listening_count].

    - Training and Test data: We split records in User Taste Data into 80(training):20(test), with training data containing 618,128 records of 365,167 users and test data containing 154,533 records of 129,417 users.

- Overall Network Graph Degree Distribution: We look at the degree distribution of number of songs and their popularity measure by total listening counts received from users, we see a near power-law distribution with heavy tail. Very few songs are a popular hit and majority of songs are not quite listened by users. The degree distribution is shown in **Fig1**.

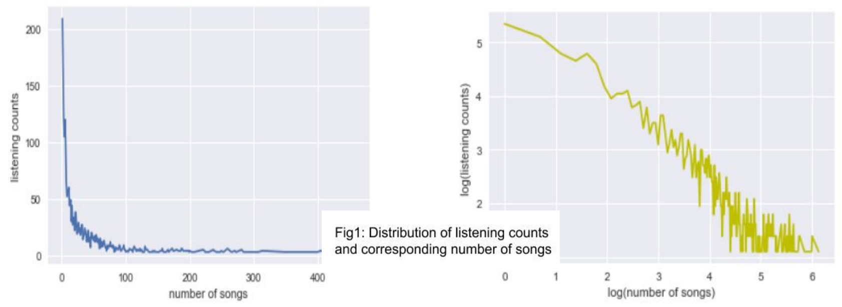


Fig1: Distribution of listening counts and corresponding number of songs

Data Access
Data for this project is called Million Song DataSet, song data can be found from https://labrosa.ee.columbia.edu/millionsong/pages/getting-dataset,
and user taste data can be found from https://labrosa.ee.columbia.edu/millionsong/pages/additional-datasets.
The song data are stored in h5 files and a python source code is offered in https://github.com/tbertinmahieux/MSongsDB to retrieve these metadata out of these h5 files.

V. Methods

5.1 : Non-Network based method: Collaborative Filtering(CF)

The intuition behind CF is to find people who are similar to you and predict your likeness for future products by looking at their likeness. The biggest challenge we face is data sparsity. Not many people have listened to the exact set of music as one user, which makes it difficult to predict using traditional approach. Another challenge is computational complexity. To compare everyone with everyone else is O(m^2) and if we were to search for similar users for each user in test data, it's not going to be something feasible given the large dataset we have. To solve sparsity problem, we performed dimension reduction in unsupervised fashion and use high level representation for music. To reduce computation time, we apply early pruning to unpromising users that are not potentially similar and use music clustering results to narrow down similar song search domain. In implementation, to allow handle big matrices and faster computation, we use python csr sparse matrix as data structure to store all listening counts for all users on all songs. Thus we end up with two matrices. One is a 392,715x74 matrix where each row represents a user and each column represents a music cluster and element represents how many times a given user has listened to some music from a specific cluster. Another is a similar 392,715x3650 matrix but each column represents each song.

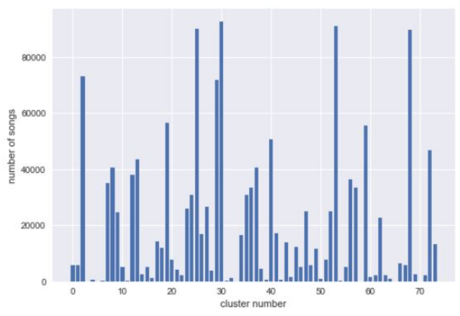
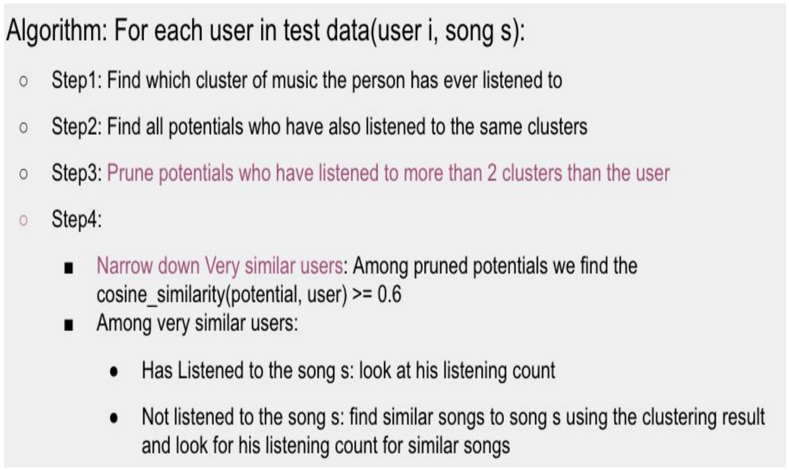


Fig2: Cluster distribution

To obtain high level representation of music, we performed k-means++ clustering to all songs using song features: duration, tempo, artist hotness, key, mode and loudness. In the end, we obtain 74 clusters and the distribution for each cluster is shown in **Fig2**. The optimal number of clusters is chosen by heuristic threshold which is one half of square root of number of samples and silhouette score.

The detailed pseudo algorithm is shown in **Fig3**. Notice the pruning part is highlighted in purple. If one has small data set, one can potentially skip the pruning process. The final prediction for listening count given user i (U_i) on song s (S_s) is obtained by:



Algorithm: For each user in test data(user i, song s):

- Step1: Find which cluster of music the person has ever listened to
- Step2: Find all potentials who have also listened to the same clusters
- Step3: Prune potentials who have listened to more than 2 clusters than the user
- Step4:
  - Narrow down Very similar users: Among pruned potentials we find the cosine_similarity(potential, user) >= 0.6
  - Among very similar users:
    - Has Listened to the song s: look at his listening count
    - Not listened to the song s: find similar songs to song s using the clustering result and look for his listening count for similar songs

Fig3: CF Algorithm

$$\sum_j \frac{cos\_sim(U_i, U_j)count(U_j, S_s)}{cos\_sim(U_i, U_j)} \text{ if } U_j \text{ has listened to } S_s$$

$$\sum_j \frac{cos\_sim(U_i, U_j)cos\_sim(S_s, S_{s'})count(U_j, S_{s'})}{cos\_sim(U_i, U_j)cos\_sim(S_s, S_{s'})}$$

if U_j has not listened to S_s and $S_{s'}$ is some similar song of S_s. U_j are very similar users found in algorithm and $S_{s'}$ are all similar songs that are found in clustering result.

The algorithm is not going to predict anything if it either encounters a cold start or could not find similar songs for very similar users.

5.2: Network based method

We first generate a bipartite graph for all songs and users for test data which contains 877 records, and an edge is formed if the user listened to the song at once. Then we apply a one-mode projection on songs, with 344 songs in our test data and 564 songs in our training data. Considering the overlapping, overall there are 691 songs in the projection. We get 690 nodes as a connected component and 1 separate node, and we keep the 690 songs in the connected component for study purpose. Overall in the graph, we have 690 nodes, 69912 edges, a relatively high clustering coefficient of 0.703 and the average shortest path length of 1.72. The bipartite and projected graph of songs is shown in **Fig4**. We observe that the graph is densely connected.

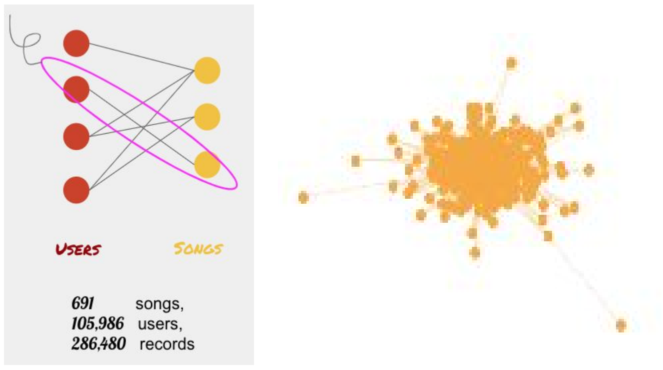


Fig4: Bipartite graph and one mode projection of songs

5.2.1: Breadth First Search(BFS)

Based on the one-mode projection network, we apply BFS(breadth first search) to determine the songs we recommend. The intuition of BFS is to find similar songs to your listened songs in history.

For each user, we set source nodes(search key) to be all listened songs in the training set for that user. According to the data analysis, average shortest path length is 1.72, which indicates the network to be a small world and if we reach out a maximum path length of 2 out of each source node, almost all nodes in the graph have been reached.

In the traditional BFS, all neighbors reached play an equivalent role in the list. However, considering the topology of our graph in terms of their path length to the source node, we give different weighted score for all neighbours we reach. Song nodes directly connected with each other have common listeners, but for a song node with path length two to the source node, they just have common listeners who listened the song in between. Therefore, song nodes at the place of path length two should be penalized for their longer distance.

In order to solve this problem, we come up with an idea to set a path length weight discount parameter $\boldsymbol{\alpha}$ **($0<\alpha<1$).** Assume the weight of the edge between source node and path-length-one node is $\boldsymbol{w1}$, and the weight between path-length-one node and path-length-two node is ***w2***, then path-length-one nodes earn a score of $\boldsymbol{\alpha} \cdot$ ***w1***, and path-length-two nodes earn a score of $\boldsymbol{\alpha} \cdot \boldsymbol{\alpha} \cdot$ ***(w1 + w2)***.

We make up these two scoring functions for reached neighbours because of following inferences and trials. The parameter $\boldsymbol{\alpha}$ times with ***w1*** so that the final score we used to rank all neighbors we reached won't be too high because of their large value of edge weight, path-length-two is farther from source node so we use the power of $\boldsymbol{\alpha}$ to increase the score discount. Besides, if we simply use $\boldsymbol{\alpha} \cdot$ ***w1*** $+ \boldsymbol{\alpha} \cdot \boldsymbol{\alpha} \cdot$ ***w2*** as a representation of the path walking from the source node to where the path-length-two node is, it will cause nodes in path length two to always have a higher score than nodes in path length one, because there is an extra $\boldsymbol{\alpha} \cdot \boldsymbol{\alpha} \cdot$ ***w2*** added in the formula when compared with path-length-one nodes, which does not make sense for these farther nodes. However, if we just cut off the first part in the summation and leave $\boldsymbol{\alpha} \cdot \boldsymbol{\alpha} \cdot$ ***w2***, the score also does not make sense in that this formula does not take into account the whole path to reach this path-length-two node and lose the information about ***w1***. At last, we take a tradeoff state of $\boldsymbol{\alpha} \cdot \boldsymbol{\alpha} \cdot$ ***(w1 + w2)*** for the score of path-length-two nodes. With ***w1 + w2***, we would like to simulate the straight-line distance to the source node, and with the power of $\boldsymbol{\alpha}$**,** we do more discount on these farther nodes.

Additionally, we take an compromise for neighbors if they are reached by different source nodes, and make up the formula of $log(\sum_{i=1}^{n}(s_i))$ ***,*** where ***s*** is the score of this reached node and ***i*** is the number of source nodes reaching them. We pick this formula after following inferences and trials. If we simply store all the score this node earned from source nodes and take the maximum value, we lose information where other source nodes contributes. More importantly, the fact that the node is reached several times indicates it has higher priority to be recommended, and if we just leave the maximum value, we lose a lot of priority and the accuracy will be lowered. Besides, if we simply add up all the score it earned, the summation will be too high so we take a logarithm function over the summation for tradeoff.

After we use scoring functions mentioned above, we get a list of all reached songs and their corresponding scores for each test user, we rank the the list based on the score in a decreasing order. The top ranked songs are recommended to the user.

5.2.1: Hitting Time(HT)

One definition of Hitting Time is expected number of steps that a random walk starts from A and hits i. It can be recursively defined as:

$$h_i^A = \begin{cases} \sum_{j \in V} \boxed{p(i \to j)} \boxed{h_j^A} + 1, & \text{for } i \notin A \\ 0, & \text{for } i \in A \end{cases}$$

where h represents hitting time and A represents starting vertex and j is some intermediate node. The complexity is $O(|V|^3)$ which is expensive to obtain given the time we have. Therefore, we use Monte Carlo Hitting Time to estimate for the true hitting time. **Naive Monte Carlo Hitting Time** is to simulate random walk from i for k times and estimate $h_{i,j}$ = (# random walks that reached j)/ k. Notice that since we need to simulate many many times in order to reach rather accurate estimation, the naive approach may still seem expensive given our network. For example, as experiment, we simulate 1000 times from the same node and observe that the number of same hitting node is very small(less than 3) thus could not give us stable result. So we still need to improve algorithm. A useful observation is that each random walk can also be used to estimate sub-walks. This is formally known as **Multiwalk Monte Carlo Hitting Time** proposed by Liu(2014). An illustrative example is shown in **Fig5**. Suppose we simulate a random walk from A to D. We can use this one simulation to also calculate hitting time from B to C, B to D and C to D. This extra gain helps us tremendously reduce the computational time. In our simulation, we simulate 5 random walks for each user's each listenned song and in the end obtain 31874 walks.

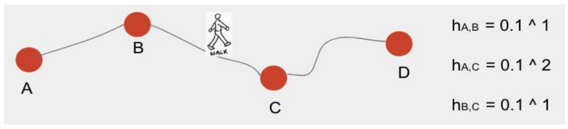


Fig5: Example of Random Walk and hitting time when beta = 0.1

The detailed simulation set up is as follows: for each user i in test data, we find all song she has listened to. We set each of the listened songs as an initial random walk starting node. Then simulate ***α*-terminating random walk** k times. Record paths and the corresponding hitting time. Then for all paths obtained from simulation, we find for all subpaths and add hitting time to each (i,j).

Here is how ***α*-terminating random walk** works: For each round at some node i, there's ***α*** probability to terminate the random walk. If not terminated, the next node is picked among node i's direct neighbours with probability proportional to the edge weights. This makes sense, because we want the more co-listened songs to have higher probability to be chosen to hit. It's formally defined as follows:

$$P(X_{t+1} = j | X_t = i) = \mathbb{1}[(1-\alpha)\frac{w_{i,j}}{\sum w_{i,j}}] \text{ if node } i, j \text{ is connected}$$

The hitting time we use is inspired from ***β*-discounted length of Random Walk**(Liu 2014), which originally calls the parameter ***α*** but to avoid confusion we denote as ***β***. The hitting time is formally defined as:

$$h(i,j) = \beta \text{\^{}step\_size}(i,j)$$

In the simulation we use, we set k=5 as mentioned earlier and ***α***=0.4 because we do not want the walk path to be too long as it's going to be eventually penalized by ***β***. We simulate the result for different ***β*** values: 0.01, 0.05, 0.1, 0.2 and 0.3.

VI. Evaluation and Result

6.1: Non-network based CF Evaluation

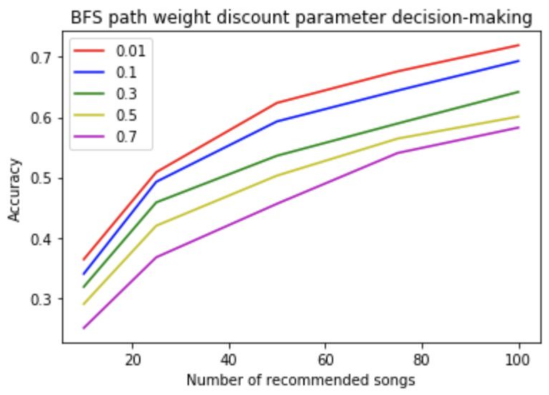
As we do not care the exact times the user has listened to a specific time, but rather we are interested in knowing how much the user likes the song, we binnize the listening count into three groups: Low likeness([1,2]), Medium likeness([3,4]) and High likeness([5,)) based on quantile distribution of overall songs' listening counts. For CF, we achieved

The result of CF based method is unpromising in terms of several ways. First about half of the time, the algorithm fails to detect near neighbours because of the pruning. Even if without pruning, the number of nearest neighbour can be still very low, and CF cannot predict well. Overall accuracy was 31% and High likeness detection accuracy was 13%.

6.2: Network based BFS and HT

Notice that in real world platform, recommendation system can be accurately evaluated with observing user behavior as time passes. However in this study, we do not have timestamp and the normal accuracy is not accurate in reflecting our system performance. One concern is that we cannot assume the user knows the recommended songs. If the user has never listened to the song we provide, this may not imply that the system performs badly, but it can be just because the user has never heard of the song. If the user ever knows about it, she may like it a lot. To make our evaluation easier, we use recall as accuracy approximation. Notice that in our network representation we do not have listening counts embedded in our network so we do not predict for listening count and thus adopt different evaluation method from CF. For network based method we evaluate as follows: Given top N recommendations for a user, we find in the test data whether the user has listened to the song. If the user has listened to, then this record is tagged as correct prediction. The accuracy of BFS and HT for different parameters is shown in **Fig6** and **Fig7**.

For BFS, we can see that the overall hitting accuracy is the highest when discount parameter $\boldsymbol{\alpha} = 0.01$. Under the choice of this parameter, we plot for the average rank of songs in our recommendation list. We can see that songs we recommended mostly lie on the first 20%-30% of the list, which indicates the quality of our BFS recommendation system is high other than its high accuracy.



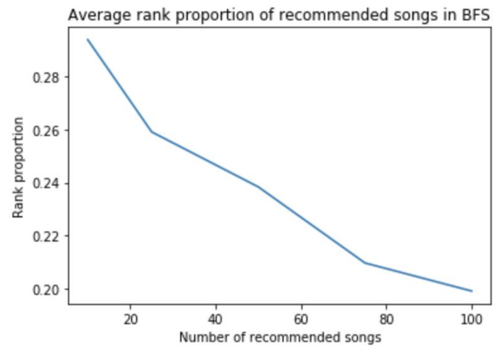


Fig6. Accuracy for BFS

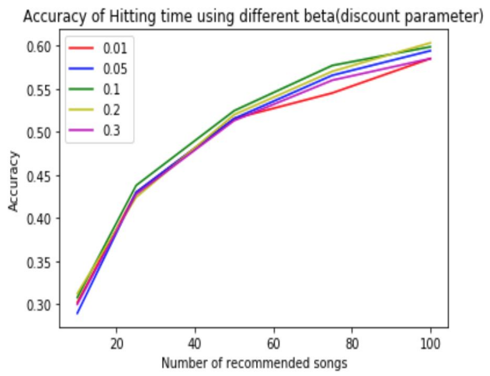


Fig7. Accuracy for HT for different parameter value

For Hitting Time, we see that overall accuracy is highest when discount parameter $\boldsymbol{\beta} = 0.1$ and number of recommended songs are less than 80. The lower the $\boldsymbol{\beta}$ value, the harsher it's going to penalize for far away neighbours. We observe a sharp increase in accuracy from 10 to 30. This implies that the rank of many correctly predicted songs are within 10-30, implying very satisfactory system performance.

VII. Project Challenges and Highlights

In addition to some of the challenges already mentioned in previous sections, we also encountered lots of problems when we create network representation, catch all corner cases for CF and finding a good way to evaluate our system. We are able to represent full data set in networkx so we need some sort of sampling heuristics that satisfies several conditions: first, to make one mode projection, the bipartite graph needs to be all connected; test data's songs need all to be in the projected graph and test user's listened songs need also be included. To make this work, we focus on set of people who have listened both to the test songs and test user's listened songs. This gives us 286,480 records of 105,986 users. We used this sub data to do network recommendation. We acknowledge that this sampling is very biased in terms that it does not include other arbitrary songs and the edge weights are biased towards connection between a test song and a listened song, so in random walk, the walker is more likely to go to songs that appeared test data. When dealing with CF, there were many edge cases such as not enough neighbours found so we need to detect whether it was because it didn't find enough users or enough songs and figure out how to fix the issue. We solved this by not looking at whether the user has listened to the exact song, but also leveraging the clustering result and find the listening count for similar songs and use that information into our final prediction. The evaluation method tricked us as well because we think the traditional method for machine learning evaluation was not suitable in evaluating our recommendation system so we tailored different evaluation methods for network based and non-network based method. This makes it hard to compare between methods but it gives us an idea of how the method is performing.

The biggest highlight of this project is that we were able to design the three algorithms based on existing methods and implemented the details all by ourselves. More importantly, we were able to draw insights from related paper and accommodate that to our project such as how we came up with hitting time and scoring for BFS using discounted length from Liu(2014). Additionally we were able to analyze the limitation of our models as mentioned earlier.

The github link for all our code is **https://github.com/Cathy97/MSD**.

VIII. Conclusion and Discussions

For finding the parameters in CF, we did not have the time to tune our parameters such as size of filtering band. The result may improve after some tuning, but considering the floating range of the accuracy for CF model, its performance is still even lower than the random guess.

Therefore, we have the following conclusions based on our study results. First, network based model performs better than non-network based ones in our recommendation mechanism, which is caused by the fact that CF fails to find very similar users with similar songs when data is sparse. Second, BFS performs better than HT between two network based models we study. Third, the highest accuracy of a network-based model can reach in our study is by BFS with a value around 70% when $\boldsymbol{\alpha} = 0.01$. The accuracy score may be further increased if we try more scoring method or tune more parameter cases in BFS, or may be increased if we try other network based models for recommendation.

Overall, network based models prevail a lot over non-network based models in our study in terms of implementing complexity and hitting accuracy.

References:

Zhang, Haiyang, et al. "Hybrid Recommendation for Sparse Rating Matrix: A Heterogeneous Information Network Approach." 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2017, doi:10.1109/iaeac.2017.8054114.

Singh, Ranveer, et al. "A Complex Network Approach for Collaborative Recommendation." 2 Oct. 2015.

*Cheng,Heng-Tze, et al. "*Wide & Deep Learning for Recommender Systems*" 2016*

Liu, Brandon. "Honor Thesis: Better than PageRank: Hitting Time as a Reputation Mechanism", 2014