Recommendation Systems and Graph Theory Applications

December 15, 2014

Vela Dimitrova Mineva
St. Lawrence University
Department of Mathematics, Computer Science, and Statistics
vdmine11@stlawu.edu

Introduction

In his article “The Race to Create a ‘Smart’ Google”, the author Jeffrey M. O’Brien writes, “The Web, they say is leaving the era of search and entering one of discovery. What’s the difference? Search is what you do when you are looking for something. Discovery is when something wonderful that you didn’t know existed, or didn’t know how to ask for, finds you.” In that sense, the goal of a recommendation system is to generate meaningful recommendations to users for items, products or services that might interest them. Today, recommendation systems (RS) are a problem-rich research area because of the abundance of applications that help us handle large amounts of information. In particular, artificial intelligence, information retrieval, data mining, security and privacy, and business and marketing are fields conducting more extensive research in the domain of recommender engines (Jannach et al. 2011).

In this paper we are going to look at the motivation behind RS and give a history overview of the field. Then, we are going to elaborate on popular recommender system approaches, namely collaborative filtering and content-based filtering, and only briefly mention the hybrid technique. This general overview of the field will put into perspective some of the common problems in the field and allow us to explore several graph theory applications in the domain of recommendation engines that tackle those issues. We are going to conclude with introducing a couple of innovative graph-based software tools.

Motivation

Recommender engines have two main purposes. First, they encourage users to undertake certain activities that might interest them, such as purchasing a product or visiting a restaurant. Second, in the current age of information overload recommender systems can be viewed as a tool for filtering information based on user’s individual preferences (Jannach et al. 2011).

History

As Joseph A Konstan, a professor in the University of Minnesota, highlights, the concept of recommender systems grows out of the idea of information reuse, and persistent preferences. This is an idea that does not emerge from technology but can be found in nature. Ants, for instance, often follow a line because they leave markers to notify other ants behind them when they find food. This system of following the steps of others to find what we need is knows as social navigation and it is in the base of social information reuse. Similarly, recommender engines collect the opinions of millions of people and reuse the accumulated data to predict our tastes and help us make better decisions.

Predecessors of recommendation systems, information retrieval and information filtering are technologies that emerged in the 1960’s. Information retrieval is the process of obtaining data relevant to a dynamic information need from a static collection of sources. Conversely, in information filtering we have a static information need and a dynamic content base. More specifically, the idea is that tastes and preferences are relatively static. Hence, the main objective is to model user’s needs and match those needs against the incoming information.

A system called Tapestry was the first recommender system (published in 1992). Given a database of content and comments on that content, recommendations were made manually based on that content and on collaborative features, namely what other people did with that content. In a nutshell, Tapestry introduced the idea of collaborative filtering and showed that both explicit annotation data and implicit behavioral data could be collected and used to create personals filters (Jannach et al. 2011).

In one of his introductory lectures on recommender systems, Konstan notes that the start of automated collaborative filtering was given in 1995 by the GroupLens Project. It was a system for filtering Usenet News, i.e. discussion list postings that flow around the Internet. The idea was to rate the articles we read. Then, the system automatically predicted for us what articles we would like to read based on a match between us and other people who shared our taste. In particular, ratings of users with similar tastes were combined to form a personalized weighted average (Jannach et al. 2011). This method is known as the nearest – neighbor approach. It is based on the assumption that taste does not change and people with history of agreement have the same preferences.

Recommendation systems became quickly commercialized. When recommender engines were introduced to the business, problems such as efficiency and scalability became priorities. As a result, new algorithms were developed to reduce computation time. Further, topics such as implicit ratings, issues for new users and items and user experience concerns are an active field of research (Jannach et al. 2011).

Personalized Recommender System Prerequisites

Prior looking at different recommendation system techniques, we have to note that every personalized recommendation system requires at least some information about every user. Hence, recommender engines build and constantly update user profiles containing each user's individual preferences. The system could, for instance, remember which products a visitor has viewed or bought in the past to predict which products might be of interest. The information can be acquired either implicitly by monitoring user behavior or explicitly by asking users for their preferences (Jannach et al. 2011).

In addition to users, items are the other required class of entities in a recommendation-system application. Users have preferences for certain items which are stored in a utility matrix. Each user-pair item is associated with a value that represents the degree of preference of that user for that item. In Table 1 we are given an example of a utility matrix representing users' ratings of items on a 1-5 scale, with 5 being the highest rating. As you can notice, many of the pairs do not have a value, which a common problem that we will address in later sections. Generally, the goal of recommendation engines is to predict the blanks in the utility matrix (Rajaraman and Ullman 2011).

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7
Alice	4			5	1		
Lily	5	5	4				
Chris				2	4	5	
Yordan		3					3

Table 1: A utility matrix representing ratings of movies on a 1-5 scale

Now we can get introduced to the basic architectures for recommendation systems, highlighted in the next section.

Techniques Used In Recommender Systems

Recommender Systems typically implement either collaborative or content-based filtering to produce a list of recommendations. There are other options such as hybrid recommendation engines that under certain conditions overcome problem of the approaches mentioned earlier.

Collaborative filtering

Collaborative filtering systems collect user feedback in the form of ratings for items in a given domain and analyze rating behavior patterns amongst multiple users in order to recommend an item (Sindhwani and Melville 2010). Two of the most popular collaborative methods are the user-based and the item-based nearest neighbor recommendations.

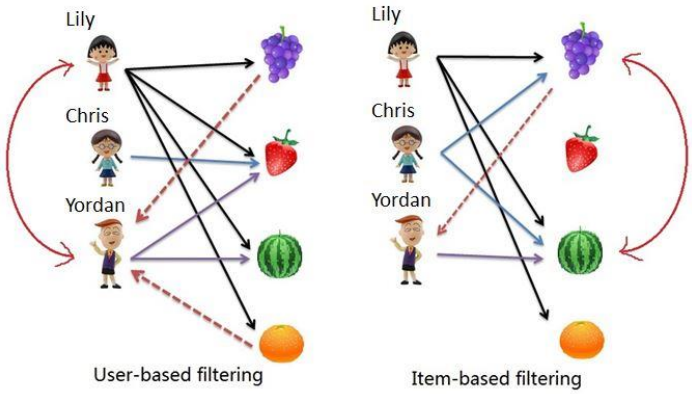

Figure 1: User-based filtering vs. Item-based filtering

User-based filtering

The idea behind user-based filtering is, given a ratings database and ID of a current user, identify other users (nearest neighbors) that have had similar preferences. Then, for each product that the active user has not seen, a prediction is made based on the ratings of the nearest neighbors. As mentioned earlier, this approach relies on the assumption that if users shared the same preferences in the past, they will also have the same preferences in the future (Jannach et al. 2011). For instance, let Yordan and Lily both like watermelon and strawberries.

Further, Lily also likes grapes and oranges but Yordan has not tried those fruits before. Then, the system would recommend those fruits to Yordan.

Today, user-based filtering techniques have been replaced by other approaches since the need to scan a large number of potential neighbors makes calculating predictions in real time impossible (Jannach et al. 2011).

Item-based filtering

Item-based algorithms, on other hand, determine predictions using the similarity between items as oppose to the similarity between users. To simplify the example, let the number of incoming edges to a fruit be our similarity measure/rating. We see that the watermelon has a rating of 3, the grapes - rating of 2 and the rest of the fruits - a rating of 0. This implies that watermelon and grapes are similar fruits. As a result, the prediction made by the item-based approach is that, since Yordan likes watermelon, he would also like grapes.

Notice that the item ratings can be preprocessed offline, which results in a reasonable computation time of recommendations even for significantly larger numbers of items and users (Jannach et al. 2011).

An advantage of the collaborative filtering (CF) approach is that it does not require information about the product which reduces space complexity. However, as Jannach et al. notice, there are many questions that remain unanswered, such as:

- How do we handle items that have not been bought yet?
- How do we deal with users without buying history?
- How do we find users with similar tastes?

Interestingly, the new-item and the new-user problems are collectively referred to as the cold-start problem. Further, in real-life applications many items do not have ratings. This makes computing good predictions extremely hard. There are multiple methods that have been suggested to handle the data sparsity and the cold-start problems (Jannach et al. 2011). In the graph theory applications section, we are going to look at one particular graph-based approach suggested by Huang et al. (2004).

Popular collaborative filtering recommenders are Last.fm, Amazon, Facebook, LinkedIn and Twitter (“Recommender Systems” 2014).

<u>Content-based filtering</u>

Content-based filtering focuses on discriminating between relevant and irrelevant suggestions and ranking them. As a result, this approach is heavily dependent on the amount of data available for each product. As Jannach et al. explain, the data is used to create item descriptions and, then, a profile assigns importance to these characteristics. If, for instance, we have a book recommendation engine, characteristics may include author, topic, or title.

An advantage of this approach is that it does not require a large group of users to make accurate predictions and new items can be recommended provided their characteristics are available. Item descriptions, however, are not always automatically acquired. Hence, data collection might require expensive and error-prone processes. Further, another issue is overspecialization, i.e. engines do not recommend items that are different from what users have already seen. In reality, users need to be provided with a wider range of options in case they want to be adventurous and try a new item (Adi Lakshmi and Sree Lakshmi 2014).

Pandora Radio is an example of a content-based recommender system that plays music with similar characteristics to the music that was initially selected by the user. Other popular examples are the Internet Movie Database and Rotten Tomatoes ("Recommender Systems" 2014).

Hybrid Approaches

We saw that both collaborative filtering and content-based approaches have disadvantages. We can, however, minimize the weaknesses and leverage strengths of those approaches by combining them. One option is to use content-based approaches where collaborative filtering is not applicable as in the case of new users and new items (Jannach et al. 2011).

Netflix is an example of a hybrid system. Recommendations are made by analyzing the watching and searching patterns of similar users (collaborative filtering) and by offering movies that share characteristics with films that are highly rated by the same users (content-based filtering) ("Recommender Systems" 2014).

Common Challenges Across Different Approaches

In the today's world of Big Data, recommender engines have become a necessity. As Konstan notes in one of his lectures on recommendation systems, however, there are challenges that remain persistent despite the various approaches, namely:

- Collecting opinion and experience data
- Finding the relevant data for a purpose
- Computing recommendations
- Presenting the data in a useful way

Various applications of graph theory have been exploited to tackle some of those problems, as we can see in the next sections.

Graph Theory Applications in the Domain of Recommendation Systems

Next, we are going to look at two applications of graph theory in the field of recommendation systems that address common problems in the field.

Suggested Solution to the Cold-Start and Data Sparsity Problems

As we already mentioned, in real-life applications the user-item rating (utility) matrices tend to be very sparse since users are generally not accustomed to rate items or data is just not available. A popular solution is to utilize additional information about the user, such as age, education and interests, to classify them and determine their preferences. As a result, informational external to the ratings matrix is incorporated in analyzing users. Then, the question that arises is how to acquire and classify the additional information (Jannach et al. 2011).

A graph-based method that tackles the cold-start and data sparsity problems is proposed by Huang et al. (2004) and explained by Jannach et al. (2011). The approach exploits the "transitivity" of customer tastes and augments the matrix with the newly acquired data. Note that such transitivity associations are typically implemented through a graph-based model because it is clear to interpret and there is an abundance of algorithms applicable to the domain of recommendation engines (Huang et al. 2004).

First, we introduce the notation provided by Huang et al. (2004). Let C denotes the set of users and P – the set of items. Then, we define the utility matrix by a $|C| \times |P|$ matrix A $= (a_{ij})$, where

$$a_{ij} = \begin{cases} 1, if\ user\ i\ purchased\ item\ j \\ 0, if\ user\ i\ did\ not\ puchase\ item\ j. \end{cases}$$

Note that the cold-start problem is a special case of the sparsity problem, where most of the entries for a certain column or row are 0.

	Item 1	Item 2	Item 3	Item 4
User1	0	1	0	1
User2	0	1	1	1
User3	1	0	1	0

Table 2: A ratings matrix for spreading activation approach

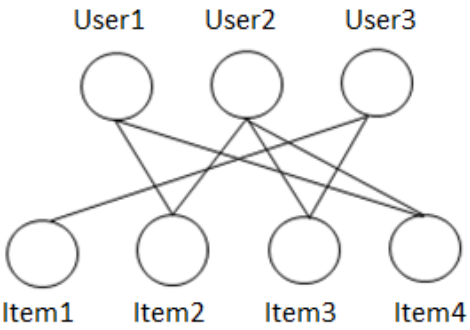

Figure 2: Graph representation of the user-item relationships

Consider the user-item relationship graph (Figure 2) constructed from the binary ratings matrix (Table 2). Note that this is a bipartite graph of users and items and there is an edge between a user and an item (vertices belonging to opposite sets) provided the user has previously purchased the item. We are going to use the spreading activation graph-exploration algorithm. In essence, we first activate a subset of nodes as source nodes and then follow the edges to iteratively activate nodes from nodes activated earlier. If there are no constraints, all connected (reachable) vertices are activated. If we are given an activation level, the process continues up to that level (Huang et al. 2004).

In the implementation of the algorithm we construct a directed graph with n vertices where each node has an associated activation value $a[i] \in [0.0, 0.1]$. An edge $a_i a_j$ connects a source node a_i with a target node a_j. Each edge has a corresponding weight $w_{ij} \in [0.0, m\ 0.1]$. We also have a firing threshold F and a decay factor D with $F, D \in [0.0, 1.0]$. Next, we provide pseudocode for the spreading activation algorithm:

Initialize all activation values $a[i]$ to 0
Set one or more origin nodes to an initial activation value greater than F
For each unfired node a_i with $a[i]$> F
    For each edge $a_i a_j$ adjust $a[j] = a[j] + (a[i] * w_{ij} * D)$
    Mark $a[i]$ as a fired node

Note from the pseudocode that as the weights propagate through the graph, they decay. The procedure terminates when there are no more nodes to fire, i.e. when the values of all nodes are less than the firing threshold ("Spreading activation" 2014).

For example, assume we are looking for a recommendation for *User1* using a collaborative filtering approach. Since *User1* and *User2* have both purchased *Item2* and *Item4*, we consider them nearest neighbors. Consequently, *Item3* is recommended to *User1* since its nearest neighbor *User2* has bought it. Hence, there we can view a recommendation as a path. In this specific example we have a path of length 3 (*User1 – Item2 – User2 – Item3*). In larger sparse databases longer paths should also be considered. The incentive of Huang et al. (2004) to translate the ratings matrix into a graph was the less expensive computation in a graph.

Huang et al. concluded that with standard collaborative filtering the quality of recommendations (including recommendations for new users) greatly improved when the ratings matrix is sparse. At a certain density of the ratings matrix, however, they found that recommendation quality starts decreasing when compared to standard methods. Further, computation remains expensive for distant relationships (Jannach et al. 2011).

<u>The YouTube Graph-Based Recommender System Architecture</u>

As we know, the YouTube community provides a personalized set of videos to signed-in users based on their previous activity on the YouTube website. Davidson et al. (2010) highlight that the design of the recommendation system is determined by the unique challenges that YouTube faces, namely poor video metadata, short and noisy user interaction, as well as short

video life cycle. The set of suggested videos is compiled from the videos a user liked/watched (which are referred to as seeds) and videos generated by traversing a co-visitation based graph of videos (Davidson et al. 2010).

Davidson et al. (2010) explain the process of constructing a set of related videos. First, we define a mapping from a video v_i to a set of videos R that the user would be interested in watching. We count for each pair of videos (v_i, v_j) the number of people who watched both videos within a given amount of time (usually 24 hours). We denote this quantity as co-visitation count c_{ij}. Then, the relatedness score of video v_i to video v_j is defined as follows:

$$relatedness\ score = \frac{co - visitation\ count}{normalization\ function} \leftrightarrow r(v_i, v_j) = \frac{c_{ij}}{f(v_i, v_j)}$$

Note that $f(v_i, v_j)$ is a normalization function. Next, we pick the top N videos ranked by their relatedness score and exclude videos with relatedness score below a given threshold. The generated set of related vides is represented as a weighted directed graph over the entire set of videos. In particular, for each pair of videos (v_i, v_j), there is a directed edge $e_{ij} = (v_i, v_j)$ if and only if $v_j \in R_i$, where R_i is the set of related videos of v_i and the weight of the edge is the relatedness score $r(v_i, v_j)$. As expected this is a simplified explanation of the algorithm which ignores factors such as video metadata and time stamps of video watches.

We continue the process with generating personalized recommendations from the set of related videos combined with the user's personal activity. Specifically, we consider the seed set S , i.e. the union of videos that a user watched/liked/rated.

Now for each video $v_i \in S$ in the seed set we consider its related videos R_i and denote the union of those sets as the set C_1:

$$C_1(S) = \bigcup_{v_i \in S} R_i$$

The set C_i often generates only videos that are very similar to the videos in the seed set and fails to recommend more diversified options that would be of interest to the user. Hence, to broaden the set $C_1(S)$, we define a new set $C_n(S)$ of videos reachable within a distance n. (Davidson et al. 2010). The set $C_n(S)$ is given by the recursive definition

$$C_n(S) = \bigcup_{v_i \in C_{n-1}} R_i$$

To explain the definition, note that $C_0(S) = S$ is the seed set. Then, the set $C_1(S)$ is defined by the formula above, i.e. the set of videos that are within a distance of 1 from any video in the seed set. Analogously, $C_2(S)$ is the set of videos within a distance of 2 from any video in the seed set, etc. The final candidate set C_{final} is defined as the union of the sets $C_1, C_2, \dots, C_n$, where $N$is upper limit for the distance from any video in the seed set:

$$C_{final} = (\bigcup_{i=1}^{N} C_i)$$

Davidson et al. highlight that the high branching factor (out-degree) of the related videos graph results in a diverse set of recommendations even over a small distance. The videos in the resulting set C_{final} are then ranked according to a variety of factors such as video quality (ratings, upload time, etc.), user specificity (consider properties of seed videos) and diversification.

YouTube's recommendation quality is determined by metrics such as session length, and click-through rate (the number of clicks over the number of times a video was displayed on a page).

In an earlier paper on the video suggestion and discovery for YouTube by Baluja et al. (2008) a different approach is described using the adsorption algorithm.

The adsorption algorithm is a framework for classification and learning given a set of labeled objects and a graph structure of labeled and unlabeled objects. The goal is to label all unlabeled nodes in the graph. The algorithm propagates label-information from the labeled nodes to the entire set of vertices via the edges. The labeling is implemented with a weight score, where higher score indicates a higher association of the node with that label. More specifically, the algorithm is given by an undirected graph $G = (V, E, W)$, where node v corresponds to an item (video in the case of YouTube), an edge $e = (a, v) \in V \times V$ indicates that the label of the two vertices $a, b \in V$ should be similar and the weight $w_{ab} \in \mathbb{R}_+$ reflects the strength of the similarity (Talukdar 2009).

Choosing a suitable classifying function is essential for the algorithm because computation on large graphs is expensive and simple ones do not yield optimal recommendations. Consider a user-video view bipartite graph where there is an edge between user u and video v provided the user has viewed the video. A metric such as shortest distance is not advisable because we might recommend nodes (videos) that can be reached only by high-degree nodes. That is, assume users u_1 and user u_2 have both watched a popular video v_1, i.e. v_1 is a node of a high degree, even though they have different tastes. As a result, another video v_j watched by u_2 might be recommended to u_1 only because of the short path between u_1 and v_j ($u_1 v_1 u_2 v_j$ is a path of length 3). Thus, when selecting a classifying function, the YouTube recommender engine considers the following factors when classifying a video v as relevant to user u:

- A short path connects u and v
- Multiple paths connect u and v
- There is a path connecting u and v with no high-degree nodes

For implementation details on different versions of the adsorption algorithm, we can refer to the paper on video suggestion and discovery for YouTube by Baluja et al. (2008).

In addition, eBay is another well-known company that has implemented a graph-based recommendation system. In a talk at the NYC* Big Data Tech Day in 2013, Thomas Pinckney,

senior director of engineering at eBay, highlighted that one of the challenges that his team has is building a taste profile for a user. In fact, he noted that eBay believes that it is more useful to understand the user as oppose to the item because otherwise the personalization component is left behind. Further, their core thesis is that there is a correlation between items that users like and do not like. Thomas Pinckney explained that it is hard to ask someone whether they are an extrovert or an introvert but what they like and dislike shows where they fit on that scale. Thus, eBay plots every user in a high dimensional taste space, which is represented as a bipartite directed weighted graph. Positive weights represent likes and negative weights – dislikes. Further, a user and an item connected with an edge have a dot product of their coordinates that equals the edge value. In essence, the edges give constraints on where the coordinates are in space and the dot product is one way of expressing that constraint. As a result, the coordinates of the products are picked in a way that gives insight about their similarity and dissimilarity. This taste graph of every user, every item and all the connections between them is stored in Cassandra, which is an open source distributed database management system (Pinckney 2013).

Implementing a Graph-Based Recommendation System

Today there are many different graph-based software platforms that allow us to store data, traverse it efficiently and analyze it. One such framework is GraphLab.

GraphLab

The GraphLab project was started in 2009 by Carlos Guestrin, a professor in Carnegie Mellon University. It is written in the C++ programming language and it was originally developed for Machine Learning tasks (“GraphLab” 2014). Further, GraphLab Create is an open-source Python package for handling data and performing tasks in various domains such as graph analytics, graphical models and computer vision. The package has a recommender toolkit which allows a variety of recommender models to be trained and used to make recommendations. One of the methods used to implement the recommender object is collaborative filtering (Dubois 2014). The toolkit is very well-documented and intuitive, which allows even beginners in programming to create their own recommender system. A downside is that if we want to run GraphLab Create on a Windows platform, we need to run it in a Virtual Machine. Installation is straightforward on Mac OS X and Linux. Snippets of example code from the GraphLab Conference in 2014 presented by the data scientist Chris Dubois are provided on Figure 3.

```
# Creating a recommendation system in GraphLab Create
import graphlab
my_recommender = graphlab.recommender.create(data)
recommendations = m.recommend()

#Getting recommendations for a set of users
personalized_recs = my_recommender.recommend(users=my_user)

#Restricting recommendations to a particular set of items
constrained_recs = my_recommender.recommend(items=candidates)

#Excluding previously seen observations
unseen_recs = my_recommender.recommend(exclude=ignore_these)
```

Figure 3: Example code to create a recommender in GraphLab Create (Dubois 2014)

In addition, among other tools and technologies that can be used to develop a recommender systems are the leading open-source graph database Neo4j and the graph traversal language Gremlin, which is part of the TinkerPop open source graph computing framework. As Marko Rodriguez and Peter Neubauer note, a single-relational graph maintaining a set of edges that are all homogeneous in meaning is not the optimal choice to model a complex real-life domain (Rodriguez and Neubauer 2010). In fact, Gremlin was designed to work with a specific kind of graphs, called multi-relational property graphs (Figure 4).

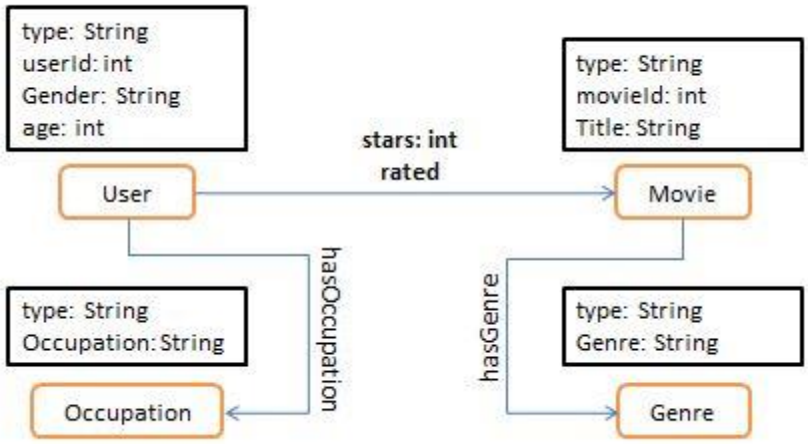

Figure 4: An example of a property graph (Rodriguez 2011)

The edges in a property graph are labeled and, hence, heterogenous in meaning. For example, on the property graph on Figure 4, the possible labels that an edge could have are rated, hasOccupation and hasGenre. The label of an edge describes the relationship between the vertices that are connected with that edge. Further, vertices and edges in a property graph maintain a set of key/value pairs (properties) that allow for representing non-graphical data.

Referring back to the graph on Figure 4, gender for example is a property of a user. Marko Rodriguez and Peter Neubauer (2010) give a formal definition of a multi-relational property graph as $G = (V, E, \alpha, \mu)$, where

- Edges are directed, $E \subseteq (V \times V)$
- Edges are labeled, $\lambda : E \to \Sigma$
- Properties are a map from the elements of the graphs ($V \cup E$) and keys (R) to their values (S), $\mu : (V \cup E) \times R \to S$

Property graphs enable greater flexibility when storing information and Gremlin facilitates the efficient analysis of that data. As GraphLab Create, Gremlin is also very well-documented and there are tutorial that can found online on building a movie recommender system.

Conclusion

In this paper we looked at the motivation behind recommendation systems and gave a brief history overview of the field. Then, we elaborated on the main approaches used in recommender engines, namely collaborative filtering, content-based filtering and hybrid recommender systems. We also looked at a graph-based solution to the cold-start and sparsity problems as well as examined the architecture of the YouTube video recommender and the eBay e-commerce recommender. My research concludes with introducing several graph-based implementation tools.

As we illustrated, recommendation engines are a critical component of many social and commercial websites because they are of great value to both owners and users (Jones 2013). Further, graph-based recommender engines are scalable, efficient, generate diversified recommendations and successfully overcome issues, such as the cold start problem. Hence, in the today reality of BigData and information overload graph-based recommenders are a valuable tool that we should continue develop.

References

Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

Baluja, Shumeet, et al. "Video suggestion and discovery for youtube: taking random walks through the view graph." *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008.

Bhunje, Sourabh. Collaborative filtering (CF) methodologies. Digital image. *The E Geek*. N.p., 29 May 2014. Web. 15 Dec. 2014.

Dubois, Chris. „Recommender Systems and Text analysis with GraphLab Create“. GraphLab Conference. 21 July 2014. San Francisco, CA. Web. 15 Dec. 2014

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. 2010. *Recommender Systems: An Introduction* (1st ed.). Cambridge University Press, New York, NY, USA.

Davidson, James, et al. "The YouTube video recommendation system." *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010.

Huang, Zan, Hsinchun Chen, and Daniel Zeng. "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering." *ACM Transactions on Information Systems (TOIS)* 22.1 (2004): 116-142.

Jones, Tim. *Recommender Systems, Part 1: Introduction to Approaches and Algorithms Page 1 of 8 Recommender Systems, Part 1: Introduction to Approaches and Algorithms*. Tech. IBM, 12 Dec. 2013. Web. 10 Dec. 2014.

Konstan, Joseph A. “Introduction to Recommender Systems.” University of Minnesota. Coursera. Retrieved from https://class.coursera.org/recsys-001/lecture. Web. 14 Dec. 2014

Lakshmi, Soanpet Sree, and T. Adi Lakshmi. "Recommendation Systems: Issues and challenges.". (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (4), 2014, 5771-5772. Web. 15 December 2014

Melville, Prem, and Vikas Sindhwani. "Recommender systems." *Encyclopedia of machine learning*. Springer US, 2010. 829-838.

O'Brien, Jeffrey M. "The Race to Create a 'smart' Google." *Fortune*. Time Inc., 20 Nov. 2006. Web.15Dec.2014. Retrieved from http://archive.fortune.com/magazines/fortune/fortune_archive/2006/11/27/8394347/index.htm

Pinckney, Thomas. “Graph-based Recommendation Systems at eBay”. NYC* Big Data Tech Day. 20 March 2014. New York City, NY. Web. 14 Dec. 2014.

Rodriguez, Marko A. "A Graph-Based Movie Recommender Engine." Web log post. Marko Rodriguez, 22 Sept. 2011. Web. 10 Dec. 2014.

Rodriguez, Marko A., and Peter Neubauer. "The graph traversal pattern." *arXiv preprint arXiv:1004.1001* (2010).

Talukdar, Partha Pratim, and Koby Crammer. "New regularized algorithms for transductive learning." *Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 2009. 442-457.

Wikipedia contributors. "GraphLab." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 29 Nov. 2014. Web. 17 Dec. 2014.

Wikipedia contributors. "Recommender system." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 15 Dec. 2014. Web. 15 Dec. 2014.

Wikipedia contributors. "Spreading activation." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 10 Nov. 2014. Web. 16 Dec. 2014.