Social Recommender systems:
Improving recommendations through personalization

Written by Tanvi Surti
Under the direction of Professor Steven Lindell
# Table of Contents

Abstract .............................................................................................................................................. 3

Personalizing a Crowdsourced Internet .......................................................................................... 4

The Scope of a Recommendation System ......................................................................................... 7

Types of Recommendation Systems ............................................................................................... 10
  1. Content Based Algorithm ......................................................................................................... 10
  2. Collaborative Filtering Algorithms ......................................................................................... 11
  3. Comparing Algorithms ........................................................................................................... 15

Social Networks meet Recommender Systems .................................................................................. 17

Social Data: Availability and Limitations ......................................................................................... 21

Social Algorithms: Clustering and Social Network Analysis .............................................................. 24
  1. Social Compatibility Method ................................................................................................. 24
  2. Top Friends Method ............................................................................................................... 27

Putting it together: A Social Recommendation Engine ................................................................. 29
  1. Feature Combination and Augmentation ............................................................................... 29
  2. Cascade .................................................................................................................................. 30
  3. Mixed/Weighted ..................................................................................................................... 31

Conclusion ......................................................................................................................................... 36

Appendix ........................................................................................................................................... 38

References ........................................................................................................................................ 40
Abstract

The vision for Web 3.0 (*popularly referred to as the Semantic Web*) is the ability to create meaning out of a deluge of qualitative data. This paper explores a very specific instance of the Semantic Web – Social Recommender Systems. This paper discusses the possibility of converting social crowdsourced data into quantitative information and using this information to power social recommendations. Over the course of this paper, we discuss five important recommender algorithms. This paper first outlines the importance of recommendations and elaborates the different types of recommendation algorithms used widely. We then discuss the potential to further personalize these recommendations by trying to identify user ‘taste’ by capitalizing on the social data available about the user. Next, we discuss the availability and applicability of data from social networks and how this data may be processed into quantitative input. This data is then used as input for two different social algorithms and their merits are discussed. And lastly this paper covers the topic of creating hybrid systems out of a wide range of recommendation algorithms so as to create social systems which are able to give diverse and personalized recommendations.
Personalizing a Crowdsourced Internet

The World Wide Web is omniscient. Within several decades of its existence, the information it holds encompasses all subject matters and presents a wide variety of opinions on them. It also enables the sharing of data which surpasses the reach of other sources of extensive information such as Encyclopedias and Expert Systems. The Internet therefore has two key advantages – Extensibility and Accessibility.

Both these advantages can be linked to the phenomenon of crowdsourcing. Crowdsourcing describes the phenomenon of collecting information from the masses – anything ranging from a government census to Yahoo Answers falls under its broad category. In the context of the World Wide Web, crowdsourcing describes the collation of information from a vast range of contributors, regardless of their qualification, age or any other distinguishing feature. Therefore while the internet consists of its share of credible sources, the internet also manages to contain a lot of excess noise and misdirected information.

Crowdsourced data, put together by a diverse set of Internet users, is disorganized and its quality is completely unchecked. To transform this data into accessible knowledge, it has to be indexed and formatted. This is called Collective Knowledge\(^1\). A great example of this is Wikipedia. Wikipedia allows for the collation of the knowledge of the collective in one source, within a specific format, but it also has the ability to remove low quality data.

However, crowdsourced data can be further utilized beyond searchable, text-based collective knowledge. It is possible to create emergent knowledge and add a layer of understanding to this collective knowledge and thus change it from collective knowledge to collective intelligence.

Tim O’Reilly presented this as the aim for Web 3.0 at the recent W3 conference. He hoped that the Web 3.0 would be more than a collation of various sources of information, but it would be an organized way of asking questions and producing answers. This would involve having the internet not only store, but also understand the data it contains to a certain extent. This third layer of understanding of data collected from the masses is called Collective Intelligence. Collective Intelligence is the label applied to any Crowdsourced data which is quantified by applying data mining and machine learning techniques resulting in discernable patterns. These patterns could be in any form – an understanding of mass opinions about a recently release movie, personality analysis of bloggers or personalized recommendations to individuals.

The focus of this paper will be a specific form of the Semantic Web – Recommender Systems. Recommender Systems are algorithms which are able to provide recommendations to a user based on the previous behavior of that particular user and other users within the system. First introduced as a primitive filter for email, recommender systems are now applied in several web-based services. Amazon uses a recommender system which provides users with suggested items to buy based on the user’s previous purchases on the website and the purchases of other users who have bought the same items. Similarly, YouTube provides a very effective recommendation system which suggests to viewers what videos they should watch next. Facebook, Netflix, Google Ads and several other prominent websites use the power of Recommender Systems to assist a user’s browsing experience. Instead of overwhelming the user with the immense amount of information on the Internet, good Recommender Systems are able to capture the essence of a user’s taste and use that to significantly narrow down choices presented to the user.

Therefore Recommender Systems are an interesting area of study. They attempt to make personalized suggestions by analyzing the public, crowdsourced pool of data. A very basic example of such understanding is the CNN World News website. Depending on which links get clicked on the most, the website is able to understand that certain stories are more popular in your geographical location and amongst your Facebook Friends. It then displays these stories more prominently on the home page because it knows that readers are most likely to be interested in those stories. The website is able to change the collective knowledge about which stories are most viewed into collective intelligence – a prediction about which stories will be the most interesting to future users.

We therefore understand that personalizing recommendations does not just involve an understanding of an individual’s past preferences, but it requires the understanding of the choices made by other internet users who are similar to the current user; therefore identifying patterns amongst users
as well as between items which we need recommendations for. We must also note that recommendations are unique for every user in a system, regardless of the system’s size and a personalized recommender system should show an understanding of the user’s taste. To extend this line of thought, if recommendations should ideally show an understanding of the user’s personal taste, it is possible to develop a recommender system which uses the plethora of social data on Social Networks available about the user to strengthen the quality of recommendations.

This paper therefore suggests adding one more level of understanding about the user by incorporating social data about this user into the recommender system. Social networks give us unique access to a large amount of obscure data about a user’s profile – such as tags, comments, work networks, age, interests, liked pages and their relationships. These all go towards creating a psychological profile of this person – and we could use this profile to make more accurate recommendations to the user. While data such as the hometown of the user might not seem relevant while recommending which movies the user should watch, it is possible to identify patterns within these seemingly unrelated attributes – for example: most users who’s hometown is Seattle have enjoyed the movie, *Sleepless in Seattle*. This paper discusses how we could integrate social data into a collaborative filtering recommender system and asks if the resulting improvements are significant.
The Scope of a Recommendation System

A Recommender System attempts to transform a large volume of data into smart suggestions for the user. Over the years, different algorithms have had popularity with recommender systems – collaborative filtering algorithms, content filtering algorithms, hybrid approaches and so forth. However, before we attempt to choose an algorithm, it is important to define the nature of the recommendation system and what we mean by the term recommendation.

The choice of algorithm to apply to a Recommender System is heavily dependent on what data is available, and how it is used. If there is a lot of apriori\(^2\) data available about the nature of our content then the algorithm we choose has the advantage of having some background about our items – for example, if we had to create a Recommendation engine for songs and had data about each song such as Artist, Album and Genre then we already have a reasonable advantage in determining the relationships between items. We therefore might be inclined towards an item-based relationship algorithm. On the other hand, if had absolutely no knowledge about the nature of the items but had a heavy usage history for many users then we might be inclined towards a user-based relationship algorithm.

The next ambiguous aspect of Recommendation Systems is to define what a recommendation is. Unlike other machine learning algorithms, there isn’t one correct answer to what is a good recommendation. For example, a social bookmarking website such as Digg.com analyzes patterns in what is currently being read by Internet users and recommends ‘Popular’ articles to a user. Let us suppose digg.com has access to the following information

<table>
<thead>
<tr>
<th>Article</th>
<th>Reads in the last week</th>
<th>Reads in the last hour</th>
<th>‘Likes’</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to whip up a quick tiramisu</td>
<td>3,234,999</td>
<td>504</td>
<td>1,026</td>
</tr>
<tr>
<td>Literary comparison of the Ode to Joy and Ode to a nightingale</td>
<td>2,349,349</td>
<td>30,583</td>
<td>920</td>
</tr>
<tr>
<td>Best horror movies of the century</td>
<td>1,203,304</td>
<td>2,203</td>
<td>1,293</td>
</tr>
<tr>
<td>Google Chrome might get rid of the address bar</td>
<td>1,340,504</td>
<td>30,304</td>
<td>501</td>
</tr>
<tr>
<td>Google introduces new Chrome extensions</td>
<td>2,340,203</td>
<td>20,201</td>
<td>302</td>
</tr>
<tr>
<td>Egypt: Wikipedia Article</td>
<td>670,302</td>
<td>123,829</td>
<td>102</td>
</tr>
</tbody>
</table>

\(^2\) Data which has to be available before the recommender system algorithm can provide results
On simply looking at this table, one realizes the complexity of determining which out of these six articles can be considered the most popular, and what weightage should be given to every attribute of data available about the articles. For example, should we consider the article about tiramisu the most popular on account of its total number of reads or should more weightage be given to the fact that the list of horror movies has the greatest number of likes. Alternatively, should we sort by theme and therefore consider the multiple articles about Google Chrome popular? Or should we look at the sudden jump in the number of reads for Egypt’s Wiki article and therefore deem it important for our users to read? This data therefore demonstrates the subjectivity of a good recommendation. While designing an algorithm, we are also faced with the challenge of selecting what attributes are important towards a recommendation and how each of these attributes should be weighted.

The weightage to be applied to a feature might also not be an explicitly stated value but it might have to be learnt by the algorithm as it processes more and more data. If feature $f$ shows a high correlation with what the user prefers then over time, a negative feedback loop increments the weight $w$ the feature $f$ is multiplied with. This Machine learning algorithm which identifies the importance of a feature with the input of training data is called Linear Regression – it uses Naïve Bayes probability techniques to distinguish the importance of each feature. This outlines an important component of a recommender system which is applied as a learning algorithm – the ability to use the accuracy of prior predictions to improve the performance of future recommendations.

Lastly, in trying to define the scope of a recommender system we must identify that there can be several types of recommendations. It is not sufficient to claim that a recommendation is any item which is similar in nature to the current item or past items of the user. Recommendations, as observed in various popular applications, can be categorized into two wide groups – the homogenous recommendation and the serendipitous recommendation. The homogenous recommendation is the highly correlational item which has the most similar feature set to the viewed item/items. To elaborate, if I was watching Season 3 Episode 1 of the Big Bang Theory on YouTube, the top recommendations shown to me are subsequent episodes of the same series. Why? Because the videos of the subsequent episodes of the Big Bang Theory have a similar title, have been watch by the same users and have similar ratings. Due to the convergence in their attributes, YouTube’s recommendation algorithm determines that Big Bang Theory Season 3 Episode 2 is a good recommendation for a user viewing Big Bang Theory Season 2 Episode 1. The homogenous recommendation is an obvious answer – it doesn’t intend to shock but to give the most natural prediction to the user.
On the other hand, the serendipitous recommendation is an item which doesn’t correlate to the current item/items in an obvious way but is more oriented towards discerning the taste of the user’s choice and making a less evident but surprising recommendation. A great example of the serendipitous recommendation is StumbleUpon – a bookmarking website which suggests articles for the user to read based on the user’s past preferences and past usage. If the interests I listed were Technology, Gardening and Fishing, and if I clicked ‘Stumble!’ then the website attempts to show me articles within the range of these interests, but attempts to avoid obvious recommendations like the Wikipedia article on Gardening. The entire purpose of using a recommendation engine such as StumbleUpon is to discover fresh and unobvious articles, without these articles being completely random. Having outlined the attributes of homogenous and serendipitous recommendations, we must note that they are not mutually exclusive. It is important to think of the nature of a recommendation on the scale between homogenous and serendipitous. The diagram below represents approximately where different recommender systems lie on this dichotomy of serendipitous and homogenous.

To summarize, in this section, we have identified several factors which must be predefined before a recommender system is picked.

1. The nature of the data that is available
2. The relative importance of the various data attributes that have been provided
3. The nature of the output – homogenous versus serendipitous recommendations

The next chapter talks about different types of recommender systems and the kind of data that is used to power them.
Types of Recommender Systems

This section discusses the three main approaches that can be taken to Recommender Systems – content based, item to item collaborative and user to user collaborative. While the idea behind each of these algorithms is defined, it must be noted that there are multiple ways of implementing them. This chapter describes the most intuitive techniques in implementing each algorithm and it must be noted that each of these algorithms can be heavily optimized. This chapter then discusses hybrid functions and the pros and cons of combining these different algorithms.

1. Content Based Algorithm

Consider local music management software – let’s say iTunes. Most local music management systems, iTunes included, have the option to auto-generate playlists which they believe the user will enjoy. In iTunes, this is called the Genius. When a user puts iTunes on Genius Mode, iTunes populates a list of songs which are most like the song the user is currently listening to. iTunes is connected to iStore and therefore has data about the attributes of each song – let’s say song name, album name, artist/band name, music genre, release date and position on music charts. iTunes however, has no data about the user. A recommendation engine which has to be implemented for a single user and which has access to apriori knowledge about the data – in this case, songs, is called a content based recommender system.

A content management system makes predictions based on the relationships between the data. In a content based recommendation system, each item has a feature vector and each feature in this vector is assigned some weight, depending on its importance. The objective is to use each feature to determine the distance between an item and all other items in the feature space, and then assign the k most similar items as recommendations. This can be broken down into several steps:

1. Normalize the feature set to ensure that each feature is assigned a value within a given range.
2. Determine a weight to be applied to each feature.
3. Calculate the similarity between two items $I_i$ and $I_j$ through the formula

$$\text{Similarity}(I_i, I_j) = w_1f(A_{1i}, A_{1j}) + w_2f(A_{2i}, A_{2j}) \ldots w_nf(A_{ni}, A_{nj}) / n$$

---

3 Which is untrue because it has access to user ratings of songs and number of plays but let us ignore that for the sake of this explanation
where $A_n$ represents the nth feature of item I, where $w$ represents the weight of the feature and where $f$ represents the function which calculates the distance between the two normalized attributes. This function might be different for each attribute, depending on the nature of the attribute – whether it is continuous or discontinuous data, whether it is Boolean data and so forth.

A content based recommender system is heavily dependent on apriori data about the items in question. However, a content based recommender system does not require data from the user to provide good recommendations. On the other hand, this algorithm is severely limited by the fact that it is unable to improve the quality of its recommendations over time, with data from the users. It is also unable to capitalize the data previous users might provide which might help improve recommendations from the current user.

2. Collaborative Filtering Algorithms

Both these shortcomings are resolved in a collaborative filtering recommender system. There are several advantages to using a collaborative filtering recommender system over a content based recommender system – firstly, collaborative filtering algorithms do not require apriori information about the data, it is able to capitalize on the data collected from other users while making recommendations and the algorithm is able to make better predictions as time passes because it collects more data. There are two ways to approach collaborative filtering recommender systems – the item-based technique and the user-based technique.

The item to item approach to collaborative filtering assumes relationships between each item in our data. For example, if we were to find the similarity between popular TV shows using the

![Figure 3: Diagram representing the placement of items with respect to others on a 2-axis feature space](image)
aggregate data about how users have rated these shows, we could imagine each show as a point on an N-dimensional graph where the Euclidian distance between two points represents the similarity between them. Therefore the closer two points are, the more similar they are and could be used as recommendations for one another.

Popular recommender systems such as those on YouTube and Amazon use item based collaborative filtering systems to display video and product suggestions to users, based on the relationships or similarity between two items. Let us suppose that a website such as YouTube has no knowledge about the nature of a video uploaded and therefore has no apriori information about the features of this item. However, what YouTube does have access to is a record of users who have watched videos, some ratings of these videos and the order in which users watch these videos. The objective of the item based collaborative system is similar to that of the content based system in trying to find the relationships of items relative to one another – however while the content based algorithm used apriori data about the items to make recommendations, this algorithm makes recommendations based on aposteriori data collected from user behavior. The algorithm for an item based collaborative filter system is as follows\(^5\)

1. For two items i and j, determine common users i.e. users who have rated/viewed both items i and j.
2. Convert the set of common users into two vectors u and v for items i and j respectively where \(v_{i,n}\) represents the rating \(R\) given by the nth user to item i. If the user has only viewed the item but not rated it, assign some constant to the unrated item.
3. Calculate \(R_i\) and \(R_j\) which is the average rating which is given to items i and k. This will be used as the normalizing factor to remove the bias from skewed rating patterns.
4. Calculate this distance between item i and j by applying the Pearson’s correlation based similarity given by applying the formula where \(R_{u,i}\) represents the rating given by user U to item i. The Pearson’s correlation creates a summation of the product of the ratings and then divides it by approximately the distance between the averages and the ratings.

\[
\text{Similarity}(i,j) = \frac{\sum_{u \in U} (R_{u,i} - R_i)(R_{u,j} - R_j)}{\sqrt{\sum_{u \in U} (R_{u,i} - R_i)^2} \sqrt{\sum_{u \in U} (R_{u,j} - R_j)^2}}
\]

5. This resulting similarity can be used in multiple ways
   a. It can be used to find the x most similar items to a current item being viewed
   b. The similarities between all items the user has viewed/rated and an item k can be used to approximate what rating the user might give this item.

The item based approach is a very popular approach amongst recommender systems is because it is a very consistent approach – if two items i and j begin to show similarity, with the collection of more and more data, this relationship is likely to be strengthened. If the data on Amazon Books shows some similarity between *Freakonomics* and *The World is Flat* this week, this similarity is not going to significantly diminish over time because the relationships between objects is likely to be stable.

On the other hand, the user based approach of collaborative filtering systems is not as preferred as an item based approach due to the instability in the relationships between users. For a system which handles a large user base, even the smallest change the user data is likely to reset the entire group of similar users. However, the conceptual advantage of taking the user to user route is that it allows for personalization of recommendations for the user, based on other attributes of the user – a concept which will be explored in the next several chapters.

The implementation of a naïve user based collaborative filtering system is not significantly different from an item based system. The objective of this algorithm is to calculate the similarities between users based on items they have commonly rated.

There are two steps within a user to user recommender system

1. Identify users which are most like the current user
2. Identify the top x recommendations, using data from those similar users

It is possible to apply the same technique for user to user algorithm as the item to item algorithm; however another less trivial approach is using a clustering algorithm. The advantage of using a clustering technique over a deterministic technique is that it iterates through our datapoints repeatedly until all similarity values converge.

---

The clustering algorithm\(^7\) attempts to map a scarce table of \(m\) users and \(n\) items into a more concentrated table of \(c\) clusters and \(n\) items. The rating for cluster \(c\) for any item \(n\) is an average of all the ratings which have been given to the item by the users in that cluster. Admittance to a cluster is determined by similarity to the center of the cluster, which can be thought of the prototype of that cluster – and the number of clusters is determined by picking the \(k\) most active users in the dataset, who have rated the most items. Each user within the dataset is then compared with each of the cluster centers and that user is allocated to the center which is most similar to it. Once all the users are sorted, the ratings within each cluster is averaged. Alternatively, to account for a small number of ratings for item \(i\) within a certain cluster, which implies that this item is not of interest to this cluster, the algorithm could augment the average rating based on the number of users who rated it.

The more detailed algorithm for this basic clustering technique is described as follows –

\[\text{Input: Matrix of user-item ratings}\]

\[\text{Algorithm:}\]

Select user set \(U=\{U_1, U_2, \ldots, U_m\}\);
Select item set \(I=\{I_1, I_2, \ldots, I_n\}\);
Choose the \(k\) users who rate the most within the dataset \(CU=\{CU_1, CU_2, \ldots, CU_k\}\);
The \(k\) clustering center is null as \(c=\{c_1, c_2, \ldots, ck\}\);
Do
\[\text{for each user } Ui \in U\]
\[\text{for each cluster center } CUj \in CU\]
\[\text{calculate the sim}(Ui, CUj);\]
end for
\[\text{sim}(Ui, CUm)=\max\{\text{sim}(Ui, CU1), \text{sim}(Ui, CU2), \ldots, \text{sim}(Ui, CUk)\};\]
\[\text{cm}=\text{cm } Ui\]
end for

```latex
\begin{center}
\begin{tabular}{|c|}
\hline
for each cluster $c_i \in c$
for each user $U_j \in U$
\hspace{1em} $C_{U_i} = \text{average}(c_i, U_j)$;
\hspace{1em} end for
\hspace{1em} end for
\hline
while (C doesn't change)
\end{tabular}
\end{center}
```

Next, Step 2 uses each cluster to calculate the average cluster rating for each item. This is done through a simple mean of all the ratings for this particular item in the cluster. The top $k$ highest rated items are pulled out from this cluster and used as recommendations. Once the rating for each item is calculated, it is possible to pull out the $y$ highest probable items within this cluster as recommendations.

3. **Comparing Algorithms**

This section describes the three most common techniques used by recommender systems – content based, user-based collaborative filtering and item-based collaborative filtering. The latter two techniques face two prominent problems, data sparsity and cold start. Data Sparsity is the problem produced out of the limited amount of data for a large number of users and items. A user on a movie recommendation site with an inventory of over ten thousand movies will visit at most two hundred movies. It is hard to use these limited ratings to predict the ratings towards all other possible combinations of movies the user might be interested in. Cold Start is the limited amount of information available about a user when he or she first joins an online community. The user’s recommendations will improve with more ratings and visits by the user, but the recommendations given to the user during the first several visits will be of a poor quality. Content based, on the other hand, might give good recommendations from the get-go however the recommendations will get repetitive as time passes. It is unable to draw on the behavior of other users to augment the recommendations of the current user – nor is it able to discern the current users taste. Also, this algorithm is too dependent on the quality of apriori data it is fed. The table below summarizes the attributes of each of these three algorithms.
<table>
<thead>
<tr>
<th>Summary: Recommender System Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content Based</strong></td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Advantages</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Disadvantages</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Recommendations Generated</td>
</tr>
<tr>
<td>Stability?</td>
</tr>
</tbody>
</table>
Social Networks meet Recommender Systems

So far, in this paper, it has been emphasized that the nature of a recommendation is extremely subjective and personal. Therefore, the user’s personal taste plays a very critical role in whether an item is a good recommendation for her. This is perhaps the biggest shortcoming of the recommendation techniques discussed so far – that none of them truly capitalize on identifying a user’s taste. While it can be argued that a user-based collaborative filtering technique attempts to identify taste, it is a very limited approach because a user’s taste is not limited to her choice of last five movies. Taste is a more holistic quality which is consistent through all aspects of her life – such as her friends, family, choice of hobbies, geographic location, age and so forth.

Taste is the basis of how humans go about the process of recommendation – they use information beyond the scope of the current recommendation to make suggestions. To elaborate, if a friend were to ask me for a recommendation for a good book, I might take into account several factors –

![Figure 5: Diagram representing a human recommendation process](image-url)
As seen through the example above, social background about the user is integral to making a personal recommendation to the user. While an algorithm cannot replicate the subtleties of subjective human thought, we can attempt to use implicit social data about the user to replicate an understanding of her general taste and personality. It is important for this data to be implicit because an algorithm which asks the user to fill extensive personality tests and identify friends within the network is cumbersome for the user and would significantly lower the retention rate of our recommendation engine.

The solution to this is the availability of social data from social networks such as LinkedIn, Facebook and MySpace. For the purposes of this paper, we will assume that our user and all her friends are on Facebook, and are reasonably active. Social Networks such as Facebook provide Application Programming Interfaces (or APIs) which allow developers on external sites to look into their large databases and with the user’s permission, pull out her social information. Facebook’s Social API is called the Graph API which allows the developer to extract information about the user such as their Photos, Friends, Likes, Movies, Music, Books and so forth\(^8\). This implicit data can be used to replicate the personalization which can only come from a person who is familiar with your personality.

There are several naïve approaches to how this social data could be used, approaches that this paper does not discuss because they are too specialized to specific knowledge-based recommendations. For example, we could identify six types of movie watching personalities and classify each user as belonging to one of these personality types based on their age, gender, ethnicity, friends, family life and preferences. While such an approach might be successful, it is not generalizable. It is therefore unadvisable to use some kind of finite list of types of tastes and trying to categorize the user into one of these types.

Instead, we try to explore techniques which are generalizable over all forms of recommendations. This paper takes two distinct approaches to incorporating the social aspect into a recommendation engine –

1. Social Compatibility algorithm

2. Top Friends algorithm

The Social Compatibility algorithm works like a dating site. The objective of a dating site is to use social information about two random people and determine how compatible they will be. This algorithm does the same thing – assuming that we have the social background of x users, we could determine compatibility between these users and decide whether they have similar tastes or not. Once it is established that a group of users have similar tastes, items rated highly by some users in this group can be used as recommendations for the others users. Therefore the Social Compatibility algorithm is based on the assumption that people with similar backgrounds will have similar tastes; and therefore like the same things. This algorithm works like the user-based collaborative filtering approach, however the users will be clustered not based on their past item ratings which result in subjective relationships, but based on their personal data which result in much more sustainable similarities.

The second approach, labeled the Top Friends Approach is conceptually more powerful in providing personalized recommendations but is limited due to data sparcity. Top Friends assumes that any item which your closest friends rank highly will be a good recommendation for you – a very simple yet effective idea. However, this approach is limited because users have only a finite number of immediate friends and only a small number of those friends might actively be using our recommendation engine. On the other hand, this approach is heavily personalized because it uses the preferences of your immediate social group to give you recommendations – an idea reiterated by homophily.

Social Networks are subject to the principle of homophily. In his paper, McPherson reports how “similarity breeds connection”, going on to describe that “people’s personal networks are homogenous with regard to many socio demographic, behavioral and intrapersonal characteristics.” What this paper demonstrates is that a person’s immediate social network is homogenous to his own interests and preferences – for two reasons: a person has relationships with people who are most like him and therefore the person’s preferences and his friends’ preferences will be highly correlational; and secondly, a person’s preferences are highly motivated by those around him – so if my friend really enjoys a music band, I am more likely to try it and be motivated to appreciate it too.

---

The implementation of the Social Compatibility technique and Top Friends Method is explored in the upcoming chapters. However, at this point we are able to see the power and the possibility of incorporating a social human-like recommendation approach towards personalizing the current user’s recommendation – and taking a recommendation beyond the explicit relationships between two items and towards simulating human taste.
Social Data: Availability and Limitations

There are two challenges to processing social data – firstly, choosing relevant data to apply to the algorithm by maintaining a balance between using too little (which makes a naïve recommender system) and using too much (which makes a time and space inefficient algorithm) and secondly quantifying the social data. This section deals with these data processing challenges.

Social Networks provide a plethora of qualitative data – in the form of comments, notes, biographies and status updates. An API such as Facebook’s Graph API gives us access to all the data about the current user and all the data about the current user’s friends. This is a lot of information! To simplify this discussion, one can divide social data into three types

1. Biographical Data
   - Example: Gender, no. of Friends, Age
   - Nature – Strictly quantitative, discontinuous

2. Interests
   - Example: Favourite books, favourite artists
   - Nature – Strictly quantitative, discontinuous

3. Transactional Data
   - Example: Comments, Status updates, wall posts
   - Nature – qualitative, continuous

This data could be used towards two ends – if we had a bunch of nodes on an n dimensional graph, firstly we are able to predict links between nodes which don’t have them (aka. Compatibility Algorithm) and secondly, we are able to determine the strength of the links between nodes which are already connected (aka. the Top Friends Algorithm).

In predicting links between nodes which don’t have them, we can assume that two nodes i and j (i.e. users) are most similar if they demonstrate homophily – similar biographical backgrounds and interests. The biographical data is easy to process with because all the attributes can be treated as discontinuous classification problems – the attribute has to belong to one class or the other. Therefore

---

10 Appendix I for exhaustive list
the attribute matches, or it doesn’t\(^\text{12}\). Passing this to an algorithm is made easier by assigning a unique ID to every class, and then checking for a match. With continuous attributes such as age, we can classify the age within discrete age groups and convert them to classification problems too. The resulting attribute array from the biographical data consists of a list of elements, which in turn can be used by the clustering algorithm. However, it must be noted that not all users have all their biographical data listed on Facebook and therefore a null value has to be passed in place of the absentee data.

The second set of data is the Interests. Unlike biographical data, Interests cannot be treated as a classification problem because a user could have interests from millions of options and each user has a different number of interests and ‘likes’. Therefore Interests have to be treated as a regression problem – where we have to see the degree of match. User A, for example, might have listed 20 favorite books and user B might have 30 favorite books – out of which 2 match. These two attribute arrays can be compared with the Jacquet similarity coefficient\(^\text{13}\), which is as simple as taking the attribute array of user A and user B, and creating a ratio of their intersection and their union –

\[
J(U) = \frac{|U_A \cap U_B|}{|U_A \cup U_B|}
\]

The result is a ratio in the range 0 to 1, which can be used as a measure of similarity between two users which is a measure of their common interests as a proportion of their total interests.

The third set of data poses the greatest challenge – comments, status updates, photos and so forth are qualitative data which cannot be easily transformed into a classification or regression problem. Hypothetical, it would be possible to parse all this text and conduct a statistical analysis of the words used by the user to discern some themes. There are two reasons why this is unnecessary – firstly, as previously stated we aren’t as interested in the identifying the personality type of the user as much as the user’s relationships; secondly, the opportunity cost of calculating these subtle themes in the user’s text might be too high as compared to more snap-to-grid information such as Hometown and Age.

However, this data is not rendered completely useless. Though we are not interested in the content of the user’s qualitative data, the total numbers are very useful to our analysis. The Graph API provides us access to the total number of people who have liked, commented on or tagged the current user in photos, statuses or wall posts. Therefore we can simply parse the contents of the transactional data to locate the total number of times the unique ID for a particular friend shows up\(^\text{14}\). This number

\(^{12}\)The SimRank algorithm in the next chapter explains another approach to process this data, other than naïve Classification.


\(^{14}\)For further information, see Appendix I
cannot be treated as an absolute because different users have varying levels of activity on social network. Therefore this number has to be taken as a ratio of the total number of comments, tags, likes and so forth. For example, John is an active Facebook user and he has transactional data with 50 of his 300 friends. We are not interested what John has shared with Bob within this social network, but we are interested to know that out of the 500 comments written in the last 90 days, John has written 150 of them on Bob’s wall. This must mean that John and Bob are really good friends.

After having explored techniques of quantifying all social data, each user on this social network can be assigned a list of quantitative attributes based on his behaviour on the social network. The next section describes the algorithms which can be used to process these social attributes.
Social Algorithms: Clustering and Social Network Analysis

There are two kinds of people within a social network that are of interest to us – first, people who demonstrate social attributes similar to the current user but are not necessarily within the current user’s social graph and second, other users which the current user has already identified to be their friends.

Different social data is important for both these sets of people. The first algorithm, which we have called the Compatibility Algorithm, uses personal data that the user has put up about herself to match it with the other user profiles. Therefore the data that will be used for this algorithm will be 1. Biographical data and 2. Interests. The second algorithm, which is called the Top Friends algorithm uses transactional data to decipher who are the most important friends within the network, and uses their top recommendations towards the current user.

1. Social Compatibility Method

The Social Compatibility Method works on the assumption that when two users have similar attributes then their tastes are similar – which is based, in part, on the principle of homophily. Homophily states that one is most likely to be friends with people with similar profiles, and therefore friends tend to have similar tastes. The Compatibility Method removes the predicate of being explicit ‘friends’ from Homophily – and extends the hypothesis that if you have similar profiles then you have similar tastes.

The algorithm used for this method is a modification of SimRank – a technique which is very similar to Google’s PageRank technique. SimRank[^15] works by propagating relationships from pair to pair within a weighted graph – that is, if two elements a and b are associated with similar neighbours then they must be similar too. To elaborate, let’s suppose Bob is a lot like John, Harry and Ajay while Linda is...

a lot like Harry, Ajay and Isaac. Since Bob and Linda are both like John and Ajay, then there must be some degree of similarity between Bob and Linda.

SimRank uses the same similarity assumption to calculate the proximity between any two nodes in our dataset. The similarity between two elements a and b, denoted by \(s(a,b)\) is signified by the following formula –

\[
s(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(l_i(a), l_j(b))^{16}
\]

where \(s(a,b)\) represents the similarity between two nodes – a and b. The summations represent the total of the similarities between every neighbour of a and every neighbour of b (neighbours represented by \(I(a)\) and \(I(b)\)). This summation is normalized by dividing the total similarities by the total number of comparisons made – which is the product of the total number of neighbours of a and the total number of neighbours of b. Lastly, the entire summation is then multiplied by \(C\), which is a decay factor between 0 and 1 which is a measure of how confident we are of the similarity calculated. To summarize, the similarity between a and b is the average of the similarities between all neighbours of a and b.

To summarize, SimRank consists of the following calculations –

1. Initialize each user with a feature set of his biographical data and interests
2. For every user in social graph
   2.1. Use the Jacquard index to calculate the similarity between this user and all other users in the social graph. These are our initial \(s(a, b)\) values
   2.2. For all \(s(current\ user, other\ user)\) values which are non-zero, other user is a neighbour of current user
3. Use Simrank to iterate over all users several times and produce new \(s(a,b)\) values
   3.1. As \(s(a,b)\) values change, so does the set of neighbours of every node
4. Use the neighbours’ preferences to recommend items to current user

The following pseudocode demonstrates the iteration over similarity values, using a matrix datastructure –

\[\text{Remember to not think of similarity as distance. The closer the similarity is to 1, lesser the distance is (closer the distance is to 0)\]
**Modified SimRank**

\[
C = 0.8\hspace{2cm} //\text{pre-chosen decay factor}
\]

Until several iterations

\[
\text{For } i = 0 \text{ to } n
\]

\[
\text{For } j = 0 \text{ to } n
\]

\[
\text{sum } = 0
\]

\[
\text{For } a = 0 \text{ to neighbours of } i
\]

\[
\text{For } b = 0 \text{ to neighbours of } j
\]

\[
\text{sum } = \text{sum } + \text{matrix}[a, b]
\]

\[
\text{matrix}[i, j], \text{matrix}[j, i] = (C \times \text{sum}) / (a \times b)
\]

//initialize similarity measure

//for every user

//iterative chosen decay factor

//for all the neighbours of i and j

//normalize over total number of comparisons made

\[
\text{The last step is to look at the finished matrix and then pull all the rating histories from the k closest neighbours and use those towards recommendations for the current user, thereby giving the current user the recommendations of users which are most like him.}
\]

This algorithm could be made more effective by taking a bipartite approach to the SimRank logic. If we assumed that each attribute of a user was an item and that items too could cluster like users could cluster, we might be able to get better results than the oversimplified Jacquard index which weighs every important and unimportant attribute equally. This involves just a simple change to the current SimRank – it relies on two formulae

\[
s(a, b) = \frac{C}{|O(a)||O(b)|} \sum_{i=1}^{\text{O}(a)} \sum_{j=1}^{\text{O}(b)} s(O_i(a), O_j(b)) \hspace{2cm} \rightarrow \text{relationship between two users } a \text{ and } b
\]

\[
s(x, y) = \frac{C}{|I(x)||I(y)|} \sum_{i=1}^{\text{I}(x)} \sum_{j=1}^{\text{I}(y)} s(I_i(x), I_j(y)) \hspace{2cm} \rightarrow \text{relationship between two items } x \text{ and } y
\]

The first equation measures the relationships between two users, where \(O(a)\) represents each attribute associated with user \(a\) and \(O(b)\) represents every attribute associated with user \(b\). \(\text{Sim}(a,b)\) now calculates the similarity between all attributes associated with \(a\) and \(b\). The same logic applies to the two attributes \(x\) and \(y\) – a comparison is made between two attributes where \(I(x)\) represents all the users which has this particular attribute. Therefore \(S(x,y)\) calculates the average of the similarities between all the users who have attributes \(x\) and \(y\).

The bipartite SimRank is more effective than the naïve SimRank because it realizes that a certain class of features might have nuanced another class of features. It might be more effective to realize that a user from Philadelphia and a user from Pittsburg have a similarity because even though Philadelphia
and Pittsburg aren’t the same class, they are similar to one another. This is a much better technique to use because it looks at each attribute differently instead of just thinking of it as a binary problem.

2. Top Friends Method

This algorithm assumes that ‘the relationship strength directly impacts the nature and frequency of online interaction between a pair of users. Since each user has a finite amount of time to use in the formation and maintenance of a relationship, it is more likely that they direct these resources towards the relationships that they deem more important... The stronger the relationship, the higher likelihood that that a certain type of interaction will take place between a pair of users’\textsuperscript{17}

The Top Friends technique is a Social Network Analysis problem – we are faced with the challenge of finding the relevant nodes within a social graph. We will commence with assuming that every node on our social graph is using our recommendation engine (i.e. we have deleted all nodes which are not), secondly we will assume that all our nodes have respectable amount of transactional data (i.e. all our users are active Facebook users). The Top Friends approach uses the measure of ‘friendship’ and your top friends’ ratings to create a ranked list of k recommendations for the current user.

As described in the previous chapter, transactional data is a measure of the interactions between two users – it is therefore a total of the number of likes, shared wall posts, photos, comments and any kind of communication. It does not treat each form of communication differently, but simply adds up the number of each piece of shared data. Therefore $\text{TransactionalData}(\text{user } a, \text{ user } b)$ represents the number of pieces of communication $a$ exchanged with $b$. However $\text{TransactionalData}(\text{user } a, \text{ user } b)$ does not represent $\text{TransactionalData}(\text{user } b, \text{ user } a)$ because relationships are being treated as directed entities, where the relationship strength from $a$ to $b$ is independent of the relationships strength from $b$ to $a$.

$$\text{Relationship}(a,b) = \frac{\text{TransactionalData}(a,b)}{\sum_{i=1}^{n} \text{TransactionalData}(a,i)}$$

The array of friends is then sorted according to relationship strength and only the top k are taken into consideration. The other relationships are considered too weak to be taken into consideration. Next, the most highly rated items are pulled out as recommendations to the current user. The user, thereby, gets recommendations from her ‘strongest’ friends. This approach is very simplistic and just uses the total number of transactions to differentiate between friends. It is possible to instead

use a Latent Variable Probabilistic model to get a much more elaborate measure of relationship strength however that is insignificant because we are only interested in the relations of the friends with respect to one another; therefore modeling the exact relationship strength is unimportant.

Having described two techniques of capitalizing on a social network to augment recommendations, the next section describes how it can be all put together into hybrid recommender systems which use various recommendation algorithms to create a diverse and varied set of recommendations.
Putting it together: A Social Recommendation Engine

This paper explores the usage of social networks to personalize recommendations to users. However, given the data scarcity in social networks, it is not suggested that the social algorithms be used independently of the non-social approaches described in this paper. This section explores the integration of various algorithms into a hybrid – a process called Ensemble Learning.

A hybrid recommender system is one that combines multiple techniques together to achieve some kind of synergy between them such that the resulting recommendations have high variance and diversity, without the introduction of unwanted noise. Let us assume that we are putting together all five algorithms which have been described through the course of this paper – content based, user-based collaborative filtering, item-collaborative filtering, social compatibility and top friends. The three hybrid techniques discussed in this section are Feature Combination, Cascading and Mixing.

1. Feature Combination and Augmentation

The first technique – Feature Combination, does not treat the different algorithms as independent of one another. It assumes that instead of processing each algorithm, we could just combine their feature sets. For example, if the feature set for John in user-based collaborative filtering was \{Sleepless in Seattle – rating 3.5, Titanic – rating 5, Snakes on a Plane – rating 4\} and his feature set for the compatibility algorithm was \{35, Male, Philadelphia, Haverford College\} then Feature Combination would just combine these two feature sets and use it as input for the user-based collaborative filtering algorithm.

This can be done relatively simply without any mathematical computation. Augmenting the attribute array of a user with social data from the compatibility algorithm strengthens user to user relationships, making them less unstable, which, as discussed previously, was the greatest shortcoming of the user-based collaborative filtering algorithm.

---

Feature augmentation is an alternative to feature combination which is more effective. Instead of the contributing recommender returning a set of features to the actual recommender, the contributing recommender processes its own input features and then assigns a single value to every data entry. It returns this data to the actual recommender which uses this value as a feature. Therefore John’s feature set could now read 

\{Sleepless in Seattle – rating 3.5, Titanic – rating 5, Snakes on a Plane – rating 4, 4.786\}

where the last number represents the numerical value assigned to him by the Compatibility algorithm. The advantage of augmenting a contributing recommender instead of combining it is that we are able to employ the logic of the contributing recommender algorithm and apply its processed output to the actual recommender.

The major disadvantage of this technique is that it is limited by whether the algorithm is based on a dataset of users or items. We cannot combine the features from an item-based collaborative filtering algorithm with a user-based collaborative filtering. Therefore it is perhaps better to use a technique which doesn’t attempt to put together the different features of these algorithms, but instead puts together the resulting recommendations.

2. Cascade

Therefore, the next approach is Cascading. This technique assumes that it is previously known (or can be learnt) which of the algorithms is the strongest. Therefore, the results from the stronger, primary recommender are used as input for the secondary recommender. The secondary recommender then uses this smaller dataset and processes it to produce the final output. The advantage of this technique is that it
allows us to rank which is the more important algorithm, therefore disallowing secondary recommenders from producing poor recommendations and having them ranked at the same weight as the primary recommender.

However, while cascading heavily reduces bias within our results by stopping weak secondary algorithms from giving poor recommendations to users, it also strongly impacts variance within our output – the diversity within the results is heavily minimized.

3. Mixed/Weighted

The Mixed method is perhaps the most intuitive and the most effective given the nature of our problem. This method allows all our models to run in parallel, and then uses the relative importance of each model to produce an intersection or union of the best recommendations. Therefore the resulting recommendations are a mixture of the best recommendations from various models, resulting in high variance.

The objective of the mixed algorithm is to present the $k$ top scored items as recommendations. If we had a dataset of $h$ items (let’s say movies) and $x$ algorithms, each algorithm $x$ must assign a probability score $P(h)$ to each movie. Next we sum up these probabilities and normalize by dividing by the total number of algorithms; lastly pulling out the top $k$ movies with the highest probabilities of being good recommendations. To take into consideration the relative strength of each algorithm, we could additionally multiply each probability score $P(h)$ by the weight assigned to that algorithm – $\beta^{19}$. The table below shows the application of the formula on some items, producing their probabilities in the hybrid

\[
Hybrid \ P(\text{item } i) = \frac{\sum_{j=1}^{x} \beta_j P(i)_j}{|x|}
\]

<table>
<thead>
<tr>
<th>Movie A</th>
<th>Movie B</th>
<th>Movie C</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model A</td>
<td>0.5</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Model B</td>
<td>0.4</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>Result</td>
<td>$[(0.75 \times 0.5) + (0.25 \times 0.4)] / 2 = 0.5$</td>
<td>$[(0.75 \times 0.2) + (0.25 \times 0.8)] / 2 = 0.6$</td>
<td>$[(0.75 \times 0.8) + (0.25 \times 0.9)] / 2 = 0.875$</td>
</tr>
</tbody>
</table>

\[^{19}\text{Calculation of the weights covered later in this section}\]
While this is an efficient technique of creating hybrids, this mixing technique does not work for the algorithms within this paper. This is due to the nature of our input data. To elaborate, the adjoining image is a diagrammatic representation of a dataset of items, semi-sorted vertically by an abstract feature of popularity. If the cloud represents all the available items, the circles represent the datasets for which our various algorithms are able to provide recommendations for. We see that content based algorithm is applicable to the largest dataset but the datasets for the other algorithms which deal with crowdsourced data are heavily dependent on the more popular items amongst the users. For example, it is very unlikely that our users will rate obscure movies – and therefore we might not have any data to produce recommendations for these films. Thus the crowdsourced models might not have enough data to produce results for the entire dataset of items. Therefore it is unlikely that all 5 models produce a sufficient probability for every item $h$ within our large dataset – rendering our previous technique useless.

To resolve this problem, we can modify a supervised learning algorithm to produce an effective Mixed Hybrid. In this hybrid method, we don’t require each model to produce a probability for every item $h$ in the dataset, instead we simply need a ranked array of top $k$ recommendations from each algorithm. We use the weights assigned to each algorithm to pull some number of recommendations from each of these arrays and put all these recommendations together to find a resulting $k$ total recommendations.

To elaborate with an example, lets suppose we have 2 algorithms which return 4 recommendations each. These 4 recommendations are ordered such that the first element is most preferable, the second is second most preferable and so forth. Each of these elements have a rank associated with them, starting with 4 and then moving in descending order. The rank is a measure of how strong the recommendation is and the weight is a measure of how strong the algorithm is, therefore the product of the rank and the weight is a measure of the total value of the element. If an element shows up in more than one array, then the multiple products are simply added. The top 4 most highly valued elements in total are then returned as recommendations –
### Ranked Array

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model A</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>Model B</td>
<td>E</td>
<td>D</td>
<td>F</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>4 x 0.7 = 2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3 x 0.7 = 2.1</td>
</tr>
<tr>
<td>B</td>
<td>2 x 0.7 = 1.4</td>
</tr>
<tr>
<td>C</td>
<td>1 x 0.7 + 3 x 0.3 = 1.6</td>
</tr>
<tr>
<td>D</td>
<td>4 x 0.3 = 1.2</td>
</tr>
<tr>
<td>E</td>
<td>2 x 0.3 = 0.6</td>
</tr>
<tr>
<td>F</td>
<td>1 x 0.3 = 0.3</td>
</tr>
</tbody>
</table>

If we multiply each of their ranks by their weights, we get their final value. On picking the top 4, our final recommendations are [A, B, D, C] in that order because they are the four highest values. This entire process is elaborated in the algorithm below.

---

**Mixed Hybrid Algorithm – Calculate top k recommendations**

\[ x = \text{number of algorithms} \]
\[ k = \text{total number of recommendations required} \]
\[ \text{array}_x = \{ \text{item 1, item 2, item 3... item } k \} \]
\[ \beta_x = \text{weight for algorithm } x \]
\[ \text{final recommendations} = \{ \} \]

for \( i = 1 \) to \( x \) algorithms
  for \( j = 1 \) to \( k \) recommendations
    rank = \( k - j + 1 \)
    weight of \( \text{array}_j[j] \) = \( \text{rank} \times \beta_x \)
    if \( \text{array}_j[j] \) in \( \text{final recommendations} \)
      add weight of \( \text{array}_j[j] \) to total weight of \( \text{array}_j[j] \) in final recommendations
    else
      insert \( \text{array}_j[j] \) into \( \text{final recommendations} \)

sort \( \text{final recommendations} \) by total weights of elements
print top \( k \) elements of sorted \( \text{final recommendations} \)

---

In the pseudocode, we assume that the weight \( \beta_x \) for each algorithm is pre-calculated. These weights are a measure of the strength of each algorithm. Most machine learning algorithms calculate weights based on aposterior data about how correct the results from an algorithm are. However because this is an unsupervised learning problem, we do not have a measure of how correct our recommendations are. But it is possible to modify the problem so as to keep track of the recommendations most prefered by the users, and then use these preferred recommendations to

---

\(^{20}\) Representing the relative strength of the algorithm
augment the weights of the algorithms they came from. For example, if user John is presented with four movie recommendations – *Slumdog Millionaire, The Notebook, Terminator 3 and the GodFather*; out of which John rates *The Notebook* highly then we can store data saying that that *The Notebook*, which was recommended by the Social Compatibility algorithm was a good recommendation. Over many, many iterations of users rating data over time, we are able to modify the weights of each algorithm.

The modified version of the Weighted Majority Algorithm (WMA)\(^\text{21}\) demonstrates this idea pseudocode below. In our modified WMA we assume that the weights of all the algorithms add up to 1 and therefore maintain that the strengths of each algorithm are relative to one another. These are the basic ideas associated with the WMA

1. We therefore initialize each of the algorithms’ weights to $1/|\# \text{ of algorithms}|$.
2. We then move through each rated item within our dataset $h$, penalizing all the algorithms which did not recommend it and rewarding all the algorithms which did. *(We will assume that the $k$ (no. of items returned by each algorithm) is large enough that when an algorithm doesn’t return an item within its top $k$, we can state that the said algorithm didn’t recommend the item at all.)*
3. We continue an endless iteration of step 2 for all users who receive recommendations and in turn, rate them.

   These formulae behind these steps are elaborated in the pseudocode below –

**Mixed Hybrid Algorithm – Calculate weights of x algorithms**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = \text{all rated items in database}$</td>
<td>//all recommendations which have been rated</td>
</tr>
<tr>
<td>$\text{algorithm}_i = \text{algorithm/s which recommended item } i$</td>
<td>//all algorithms which recommended this item</td>
</tr>
<tr>
<td>$\text{rank}_i = \text{rank algorithm/s have item } i$</td>
<td>//rank which aforementioned alg. gave this item</td>
</tr>
<tr>
<td>$\phi = \frac{1}{</td>
<td>d</td>
</tr>
</tbody>
</table>

//we subtract the average rating by this user to normalize rating

rating$_i = (\text{rating associated with item } – \text{average rating by user who rated the item})$

** β$_x = \text{weight associated with algorithm } x$ //initialize all weights to 1/# of alg so that**

Initialize all β values to $\frac{1}{x}$ //they add up to 1

For every rated item $i$ in database //infinite iterations (assuming infinite ratings)

For every algorithm, which recommended $i$

//award this algorithm for providing a good recommendation
//this award is a product of rank this algorithm gave this item and rating which the user
//gave this item; and the normalizing factor

$\beta_{\text{algorithm}_i} = \beta_{\text{algorithm}_i} + (\phi \times \text{rank}_i \times \text{rating}_i)$

//to maintain the condition that all the algorithms’ weights add up to 1, we penalize all
//other algorithms for not recommending this item

For every other algorithm $j$ (which didn’t recommend $i$)

$\beta_{\text{algorithm}_j} = \beta_{\text{algorithm}_j} - \frac{\phi \times \text{rank}_i \times \text{rating}_i}{|x| - 1}$

Out of the three hybrid techniques explored within this section, Mixing does the most justice to the serendipitous recommendations which can be returned by diverse algorithms. It introduces variance, without letting bias escalate. It also takes into account that each of our algorithms is working with a slightly different dataset and therefore their scores on the same items aren’t comparable. The weighting ensures that we do not impose inefficient algorithms on the users. Furthermore, this hybrid model could be made much more personalized by basing the weights for algorithms on each user’s own behavior. This could be implemented with an easy change to the WMA algorithm.

We have now successfully integrated a couple of personalized models within the hybrid recommendation engine, also allowing the hybrid algorithm to learn over time whether the social algorithm is producing good recommendations for the users; and if not, its weight will reduce a negligibly small amount over time. This provides a measure of the effectiveness of social algorithms.
Conclusion

The Paradox of Choice is a phenomenon whereby the availability of an overwhelmingly number of choices ironically leads to reduced happiness and productivity. The opportunity cost of trudging through a large volume of information in return of some insignificant benefit is too low. Therefore there is a strong motivation behind researching recommendations – removing the Paradox of Choice from the internet and presenting users with a smaller but much more robust dataset of choices.

In this paper we have talked about three distinct recommendation topics – first, a discussion of the merits of popular recommender algorithms; second, the availability of social data and how it can be used towards social recommendations and thirdly, the creation of personalized hybrid models. The objective behind this discussion has been to explore further personalization of the web –

1. by capitalizing on crowdsourced data by concurrently running user-based and item-based collaborative filtering systems
2. by allowing the socialization of recommendations through Top Friends,
3. by attempting to decipher a user’s taste through Social Compatibility,
4. by trying to account for the user preferences in algorithm choice in Mixed Hybrids.

I have attempted to focus on the subtleties of user taste instead of attempting to singularly focus on stable similarities between items, and therefore trying to incorporate serendipitous recommendations in the results.

However there is also an ethical and legal side to the topic of Social Recommendations. Social algorithms rely heavily on implicit data that users put out on the internet without the intention of that data being used towards investigating their ‘taste’. Users might not be as comfortable with their personal information being used towards recommendations – the same way one might feel uncomfortable with banner ads capitalizing on one’s web browsing history towards targeted ads. After having booked a flight from Philadelphia to Mumbai one year ago, I continue to get targeted Expedia ads on every blog I visit. While it is technically impressive that Google Ads retained that data and has matched it with a relevant recommendation, the ethical implications of this data being used without my explicit permission are uncertain.

Social recommendation engines such as Hunch and Rapleaf use searchable databases towards aggregating data about users. Rapleaf outputs biographical and geographical information about the
user, given their email address. Hunch provides recommendations on absolutely any subject, given a user’s name. The APIs from Rapleaf and Hunch can be capitalized by many website to personalize a user’s browsing experience – but on the other hand, the user is not explicitly aware that this data is being accessed. It is therefore a fine line between capitalizing on crowdsourced data and infringing privacy laws – and these are lines which social data aggregators such as Facebook and Google have to tread carefully.

Despite the legal risks involved, Social recommendations continue to be very relevant – perhaps because the best replicate the human decision making process. Web startups and giants such as Google Social, Jinni, Last.fm, Pandora, YouTube, StumbleUpon and so forth are currently pushing the personalized and socialized web towards a very tangible reality in the near future. As the expert systems of the world make way for the crowdsourced recommendation systems, British journalist Jemima Kiss’ words ring true –

“If web 2.0 could be summarized as interaction, web 3.0 must be about Recommendation and Personalization”

---

Appendix

This appendix describes how Facebook’s Graph API is used towards collecting Social Data. The Graph API allows the developer access to the user’s personal details, if the user provides the application with the authorization to do so.

A user is an Object within the Facebook Graph API and each Object has a unique ID associated with it. Each object has two kinds of attributes – properties of the Object and connections of the Objects to other Objects. To elaborate, a user ‘Tom’ has properties id, first_name, last_name, education and so forth. The user ‘Tom’ also has a bunch of connections – albums, events, feed, likes and so forth. On referencing these connections, we then enter a new object with a bunch of properties and connections of it’s on.

Instead of trying to populate the database with our recommendation engine with the entire social graph of the user, we must be selective about what information we collect from the Graph API. It would be ludicrous to assume that all the user’s friends are also using our recommendation engine, therefore instead of pulling the names, ids and biographical information about all of the user’s friends, we attempt to only pull out the data about the user and then parse her list of friends to identify which of them are also using our recommendation engine – we are able to do so because Facebook assigns every user with a unique ID. Therefore, once we identify which friends of the current user are also using our recommendation engine, we can then pull transactional data about those friends only, as elaborated by the table below.

For the purpose of our Social Algorithms, we have identified our requirements for three types of data – biographical, interests and transactional. This table is an exhaustive list of all the data we need to collect from the Graph API and the permissions required for it.

<table>
<thead>
<tr>
<th>Data</th>
<th>Link</th>
<th>Object/Property</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Biographical Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Id</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Basic</td>
</tr>
<tr>
<td>First_name</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Basic</td>
</tr>
<tr>
<td>Last_name</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Basic</td>
</tr>
<tr>
<td>Gender</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Basic</td>
</tr>
<tr>
<td>Locale</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Basic</td>
</tr>
<tr>
<td>Username</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Basic</td>
</tr>
<tr>
<td>Birthday</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_birthday</td>
</tr>
<tr>
<td>Education</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_education_history</td>
</tr>
<tr>
<td>Email</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>Email</td>
</tr>
<tr>
<td>Hometown</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_hometown</td>
</tr>
<tr>
<td>Location</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_location</td>
</tr>
<tr>
<td>Political</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_religion_politics</td>
</tr>
<tr>
<td>Relationship_status</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_relationship_details</td>
</tr>
<tr>
<td>Religion</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_religion_politics</td>
</tr>
<tr>
<td>Work</td>
<td><a href="https://graph.facebook.com/*currentuser">https://graph.facebook.com/*currentuser</a>*</td>
<td>Property</td>
<td>User_work_history</td>
</tr>
<tr>
<td>Friends</td>
<td><a href="https://graph.facebook.com/*currentuser*/friends?*AccessToken">https://graph.facebook.com/*currentuser*/friends?*AccessToken</a>?*</td>
<td>Object</td>
<td>Basic</td>
</tr>
<tr>
<td><strong>Interests</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activities</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_activities</td>
</tr>
<tr>
<td>Checkins</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_checkins</td>
</tr>
<tr>
<td>Interests</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_likes</td>
</tr>
<tr>
<td>Likes</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_likes</td>
</tr>
<tr>
<td>Movies</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_likes</td>
</tr>
<tr>
<td>Languages</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_likes</td>
</tr>
<tr>
<td>Television</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_likes</td>
</tr>
<tr>
<td><strong>Transactional Data (after parsing through friend lists to see which friends are using our recommendation engine)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_events</td>
</tr>
<tr>
<td>Photos</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>User_photos</td>
</tr>
<tr>
<td>Posts</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>Read_Stream</td>
</tr>
<tr>
<td>Statuses</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>Read_Stream</td>
</tr>
<tr>
<td>Tagged</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>Read_Stream</td>
</tr>
<tr>
<td>Feed</td>
<td><a href="https://graph.facebook.com/*currentuser*/*token_name?*AccessToken">https://graph.facebook.com/*currentuser*/*token_name?*AccessToken</a>?*</td>
<td>Object</td>
<td>Read_Stream</td>
</tr>
</tbody>
</table>
References


