— 1 —
Theory

1.1 Recommender Systems

The name might seem constraining, but recommender systems are incredibly powerful methods in user modeling. Whenever we wish to predict the relevance of an item to a user, recommender systems are the tools to use. Such systems are commonly used on the web to provide a host of predictive functionality, including:

- Recommending products like books or movies based on past purchases.
- Suggesting new social connections based on an existing social graph.
- Recommending items based the activity of similar or like-minded users.
- Ordering news articles by predicted individual relevance.
- Personalizing search results based on the current user.

Common to these examples are a set of users, a set of items, and a sparse set of explicit ratings or preferences. Items can be anything: Documents, movies, music, places, people, or indeed other users. A recommender system is best described by graph and graph operations, even though the underlying algorithms might not use this as the representation. Mirza and Keller (2003) explains how any RS can be expressed as a graph traversal algorithm. Items and users are nodes, while ratings, social connections et cetera are edges between the nodes. An RS performs predictive reasoning on this graph by estimating the strenghts of hypothetical connections between nodes that are not explicitly connected.

For example, if a user has rated some of the movies in a movie recommendation system, we use these ratings to predict how well the user will like unseen movies, based on a movies ratings from users similar to the one in question. In social networks, recommender systems can be used to infer new social relations based on existing connections. The principle is the same: By evaluating current explicit connections, and the connections of similar users, new connections can be predicted. Recommender systems are then powerful methods for user modeling, personalization and fighting information overload, because of their ability to infer how relevant and item (or another user) will be to the current user.

Formally, a recommender system can be seen as a quintuple, $\mathrm{RS} = (I, U, R, F, M)$, where I is the set of items (e.g. products, articles or movies) and U is the set of users. R is the set

of known connections, for example explicit preferences given by users for certain items, or connections in a social graph. F is a framework for representing the items, users and ratings, for example a graph or matrix. M is the actual user modeling method used to infer unknown ratings for predicting a user's preference for an unrated item. This is where AI comes in.

In Adomavicius and Tuzhilin (2005), M is seen as a utility function $f : U \times I \rightarrow S$. Here, f is a function that maps the set of users and items into a fully ordered set of items S, ranked by their utility (i.e. rating) to each user. In other words, S is the completely specified version of R, where each user has either an explicit, implicit or predicted preference for each item in I. To predict the best unrated item for each user, we simply find the item with the highest expected utility:

$$\forall u \in U,\ i'_u = \arg\max_{i \in I} f(u, i)$$

The utility function u depends on the modeling method being used, the active user and the item in question. The *reason* for using a recommender system is that the utility u is not defined for the entire $U \times I$ space, i.e. the system does not explicitly know the utility of each item for each user. The point of a recommender system is then to extrapolate u to cover the entire user-item space. In other words, to be able to rank items according to user preferences, the system must be able to predict each user's reaction to items they have not yet explicitly rated themselves. This is where predictive user models come in handy.

Another popular way of describing, and implementing an RS is using a simple matrix. Here, one dimension represents users, the other dimension represents items, and each cell corresponds to an explicit rating. This matrix then becomes the framework F in our RS quintuple:

$$R_{u,i} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,i} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ r_{u,1} & r_{u,2} & \cdots & r_{u,i} \end{pmatrix}$$

Critically, these matrices are usually extremely sparse (i.e. most of the cells are empty). Consider that while there may be a large number of users and items, each individual user only rates or connects to a few number of items. For example, in the seminal Netflix Challenge movie recommender dataset, almost 99% of the potential user/item pairs have

no rating (Bell and Koren, 2007, p1). In other words, the recommender system must be able to produce results from a matrix where only 1% of the cells have meaningful values.

Naturally, this is the defining characteristic of many recommender systems: the ability to extract meaningful patterns from sparse data, through dimensionality reduction, neighborhood estimation and other methods, as we shall see.

Recommender systems face many challenges other than the sparsity problem. A directly related problem is the need for large datasets. Since the data is often sparse, the systems will most often perform well if used on large numbers of items and users. As in many machine learning methods, concept drift, where the characteristics of a user or item changes over time, is also always present.

The performance of RSs is often closely tied to their computational complexity. Real world usage of the most precise methods is often hindered by the computational power needed to actually put them into production.

Finally, the scale of the data in question should be a concern. If the ratings are ordinal data (e.g. 1-5) input directly by users, the RS should take into account the domain specific meaning of these intervals. For example, in a system for rating movies, the jump between ratings 4-5 might not have the same significance as the jump from 2-3. However, this is a fact seldom mentioned in the literature. Most RSs employ metrics that assume a normal distribution, and even the common evaluation techniques such as RMSE or MAE treat ordinal data as a continous scale.

Prediction

The most interesting and important part of any RS is how it predicts unknown ratings. (Note that altough we use "ratings", "utility", "preference", "relevance" and "connection strength" depending on the context, they all basically mean the same.) Because of this, each method is best categorized based on a few dimensions of its predictive capabilities (see Table 1.1). In our taxonomy, these dimensions are: *data*, *method*, *granularity*, *temporality* and *agents*.

The *data* variable represents what data the RS uses to perform predictions. Content-based methods use only the items, inter-item relations, and an individual user's past history as predictive of future actions (Pazzani and Billsus, 2007). By only considering the individual user in adapting an application, highly personal models can be created. However, such methods often require a lot of interaction before reliable models can

be created (Adomavicius and Tuzhilin, 2005). The problem of having to do complex inference from little data, as is often is in content-based predictions, is often called the *sparsity problem* or the *cold start* problem. This is closely related to the problem of *overfitting* data, where the algorithms creates models that match the training data, but not the actual underlying relationships. A lot of research looks at ways to overcome sparse data, i.e. achieving "warmer" cold start. When using content-based predictions, the utility function $f(u, i)$ of user u and item i is extrapolated from $f(u, i_u)$, where i is an item similar to i_u and $f(u, i_u)$ is known.

Collaborative or social recommendations build predictive models for users based on the actions of similar users (Schafer et al., 2007). The observation is that similar users should have similar usage and action patterns. By using data from more than one user, expansive models may be built. These methods are especially useful when considering new users of a service. A central problem with collaborative methods is that the resulting model is not as individually tailored as one created through content-based prediction. Collaborative models must be careful not to represent the *average* user, but a single individual. When using a collaborative method, the utility $f(u, i)$ of item i for user u is extrapolated from $f(u_j, i)$ where u_j is a user similar to u.

Because of *the new user problem* of content-based prediction and the *average user problem* of collaborative prediction, many systems use a hybrid approach (Burke, 2007). By combining content-based and collaborative methods, systems that properly handle predictions for new users and avoid too much generalization in the models can be achieved.

The *method* variable, is another way to classify recommenders. Orthogonal to what data the method uses, this variable concerns *how* the data is used to produce recommendations. First we have the *model-based* approach, where the recommender system builds predictive models based on the known data. Unseen items can then be fed into this model to compute its estimated utility score. For example, creating a Bayesian networks from past

Variable	*Values*
Data	Content-based | Collaborative | Hybrid
Method	Heuristic | Model-based
Granularity	Canonical | Typical | Individual
Temporality	Short-term | Long-term
Agents	Implicit | Explicit

Table 1.1: A taxonomy of recommender systems. From Bjorkoy (2010).

interaction is a model-based approach. The other category is the *heuristic* or *memory-based* approach. These methods use the raw data of items, users and ratings to directly estimate unknown utility values. For example, recommending items similar to the ones already rated by computing the cosine similarity of their feature vectors is a heuristic approach.

The *granularity* variable tells whether this approach creates models for the canonical user, stereotypical users or individual users. Rich (1979) presented one of the first user modeling systems based on stereotypes, used to predict which books in a library each user would most enjoy. Here, a dialogue between the system and the user was performed to place the user into a set of sterotypes. Each stereotype has a set of *facets* which is then used to match books and users.

Temporality refers to how volatile the gathered knowledge will be. While most RSs produce long term, relatively stable knowledge based on lasting user preference and taste, some systems use fluctuating parameters such as the time of day, exact location and the current context to produce recommendations. For example, Horvitz et al. (2003) used clues from a user's calendar, camera and other sensors to determine the attentional state of the user before delivering personalized and contextual notifications.

The *agents* variable signifies whether the knowledge gathering and presentation is implicit and opaque, or explicit and requires dedicated user interaction. Explicit feedback through ratings is common in movie, product or music rating services (e.g. Bell et al. (2007); Basu et al. (1998); Hotho et al. (2006)). However, for other services such as personalized search, implicit mining of query logs and user interaction is often used to build user models (e.g. Shen et al. (2005); Agichtein et al. (2006); Speretta and Gauch (2000); Teevan et al. (2005))

Approaches

Because our solution will combine different recommender systems, we need a short introduction to some of the approaches we will combine. See Adomavicius and Tuzhilin (2005), Pazzani and Billsus (2007), Schafer et al. (2007) or Bjorkoy (2010) for a more comprehensive exploration of different types of recommenders.

(1) *Baseline ratings* are the simplest family of recommender systems, based on item and user rating averages. The data is content-based, and used to compute heuristic predictions. This is done on a per-user, individual basis and collects long term knowledge so far as the user is rational in most of his or her ratings. While simple in nature, they are often helpful as starting points for more complex systems, or as benchmarks for

exploring new approaches. (Koren, 2008, p2) computes the baselines for items and users, and use more involved methods to move this starting point in some direction. The baseline for a user/item pair is given by

$$b_{ui} = \mu + b_u + b_i$$

where μ is the average system rating, b_u is the user baseline and b_i is the item baseline. The user and item baselines correspond to how the user's and item's ratings deviate from the norm. This makes sense as some items may be consistently rated higher than the average, some users may be highly critical in their assesments, and so on. Koren computes these baselines by solving the least squares problem

$$\min_{b*} = \sum_{(u,i)\in R} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(\sum_u b_u^2 + \sum_i b_i^2)$$

which finds baselines that fit the given ratings while trying to reduce overfitting (as weighted by the λ parameter). By using baselines instead of simple averages, more complex predictors gain a better starting point, or in other words, a better average.

Another approach based on simple averages is the *Slope One* family of collaborative filtering algorithms. As introduced by Lemire and Maclachlan (2005), these algorithms predict unknown ratings based on the average difference in ratings between two items. For example, if item i is on average rated δ points above item j, and the user u has rated item j, that is, we know r_{uj}, the predicted rating of i is simply $r_{uj} + \delta$, for all ratings that match this pattern. In other words,

$$\hat{r}_{ui} = \frac{\sum_{j\in K_u} \text{ratings}(j) \times (r_{uj} + \text{diff}(i,j))}{\sum_{j\in K_u} \text{ratings}(j)},$$

where $\hat{r}_{ui}$ is the estimated rating, K_u is the items rated by user u, $\text{ratings}(i)$ is the number of ratings for item i, and $\text{diff}(i,j)$ is the average difference in ratings for items i and j. While simplistic, Slope One is computationally effective and produces results comparable to more complex methods (Lemire and Maclachlan, 2005, p5).

(2) *Dimensionaliy reduction* is an oft-used technique when creating recommender systems. The ratings matrix is factored into a set of lower dimension matrixes, that can be used to approximate the original matrix. Since this has the added effect of trying to approximate

unknown cells, the lower dimension matrices can be used to predict unknown ratings (a type of least squares data fitting).

Singular Value Decomposition (SVD) is a common method for such matrix factorization (e.g. Billsus and Pazzani (1998, p5), Sun et al. (2005), Bell et al. (2007)). This is the same underlying technique used by *latent semantic indexing* (LSI) in information retrieval (Baeza-Yates and Ribeiro-Neto, 1999, p44). Formally, SVD is the factorization $M = U\Sigma V^*$. M is an $m \times n$ matrix, in our case the ratings matrix, with m users and n items. U is an $m \times m$ factor (sometimes called the "hanger"), V^* (the conjugate transpose of V) is an $n \times n$ factor ("stretcher"). Σ is a $m \times n$ diagonal matrix ("aligner").

The dimensionalty of the ratings space is performed by truncating the factor matrices each to a number of rows or columns, where the number is a paramenter depending on the current domain and data. By truncating the factors, we in essence create a higher-level approximation of the original matrix that can identify latent features in the data. With the factors reduced to k dimensions, the result looks like this:

$$\begin{bmatrix} \\ R_{m,n} \\ \\ \end{bmatrix} \Rightarrow \begin{bmatrix} \\ U_{m,k} \\ \\ \end{bmatrix} \begin{bmatrix} \Sigma_{k,k} \end{bmatrix} \begin{bmatrix} \quad V^*_{k,n} \quad \end{bmatrix}$$

Two important transformations happen in this reduction. First, ratings that do not contribute to any greater pattern are removed as "noise". Second, ratings that in some way correlate to each other are enhanced, giving more weight to the actual predictive parts of the data. This means that the reduced factors can for instance identify features that correspond to correlations between items or users. These features are comparable to the mapping of terms to concepts in LSI.

There are many ways of using the reduced factors. We can for instance use the resulting reduced factors to find similar users by their cosine similarity (explained below). SVD is then an ingenious way of dealing with the commonly sparse ratings data, by identifying latent correlations and patterns in the data, which is exactly what we need to predict unknown ratings or connections.

(3) *Neighborhoods* knn, pearson

(4) *IR methods* vector space, page rank

(5) *Social networks* traversal, transitive trust

— 2 —

Methods

Getting past 80%

Power of data

Hypotheses

2.1 Modeling Phase

$$\text{AM} = (Items, Users, Framework, Methods, Aggregation) \tag{2.1}$$

2.2 Prediction Phase

2.3 Implementation

2.4 Evaluation

Datasets

Metrics

Experiments

References

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.

Agichtein, E., Brill, E., Dumais, S., and Ragno, R. (2006). Learning user interaction models for predicting web search result preferences. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '06*, page 3.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern information retrieval*, volume 463. ACM press New York.

Basu, C., Hirsh, H., and Cohen, W. (1998). Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 714–720. JOHN WILEY & SONS LTD.

Bell, R., Koren, Y., and Volinsky, C. (2007). The BellKor solution to the Netflix prize. *KorBell Team's Report to Netflix*.

Bell, R. M. and Koren, Y. (2007). Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75.

Billsus, D. and Pazzani, M. (1998). Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 54, page 48.

Bjorkoy, O. (2010). User Modeling on The Web: An Exploratory Review.

Burke, R. (2007). Hybrid Web Recommender Systems.

Horvitz, E., Kadie, C., Paek, T., and Hovel, D. (2003). Models of attention in computing and communication: from principles to applications. *Communications of the ACM*, 46(3):52–59.

Hotho, A., J, R., Schmitz, C., and Stumme, G. (2006). Information Retrieval in Folksonomies: Search and Ranking.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM.

Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. *Society for Industrial Mathematics*.

Mirza, B. and Keller, B. (2003). Studying recommendation algorithms by graph analysis. *Journal of Intelligent Information*.

Pazzani, M. and Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer-Verlag.

Rich, E. (1979). User modeling via stereotypes. *Cognitive science*, 3(4):329–354.

Schafer, J., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems. *The adaptive web*, pages 291–324.

Shen, X., Tan, B., and Zhai, C. (2005). Implicit user modeling for personalized search. *Proceedings of the 14th ACM international conference on Information and knowledge management - CIKM '05*, page 824.

Speretta, M. and Gauch, S. (2000). Personalized Search Based on User Search Histories. *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 622–628.

Sun, J., Zeng, H., Liu, H., and Lu, Y. (2005). CubeSVD: a novel approach to personalized Web search. *on World Wide Web*, pages 382–390.

Teevan, J., Dumais, S. T., and Horvitz, E. (2005). Personalizing search via automated analysis of interests and activities. *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '05*, page 449.

Appendix

Sourcecode, tables, et cetera.