ASSIGNMENT

User Meta Modeling

Aggregating User Modeling Methods
On a Personal Level

Study how to create composite user models by combining results from numerous complimenting modeling algorithms. Create a flexible algorithm that individually combines the results into one coherent user model. Utilize the resulting composite user modeling algorithm in an information retrieval system to provide personalized search.

Contents

List of Figures

— 1 —
Introduction

In 1971, Herbert Simon said the following on the topic of information overload: "What information consumes is rather obvious: it consumes the attention of its recipients. Hence a wealth of information creates a poverty of attention and a need to allocate that attention efficiently among the overabundance of information sources that might consume it." Greenberger (1971).

In 2009, Alon Halevy, Peter Norvig, and Fernando Pereira, wrote: "Perhaps when it comes to natural language processing and related fields, we're doomed to complex theories that will never have the elegance of physics equations. But if that's so, we should stop acting as if our goal is to author extremely elegant theories, and instead embrace complexity and make use of the best ally we have: the unreasonable effectiveness of data." Halevy and Norvig (2009).

Higher orders of training data: Some algs perform better than others, depending on amount of data. Banko and Brill (2001)

Simple algorithms, lots of data, many algorithms, user-combined.

Data-driven learning: http://archive.ecml.at/projects/voll/rationale_and_help/booklets/resources/menu_booklet_ddl.htm

"Predictive accuracy is substantially improved when blending multiple predictors. Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique. Consequently, our solution is an ensemble of many methods." — Bell et al. (2007b) Teams with different insights, each team votes on answers. Breakthrough in netflix prize challenge – race of ensembles.

Previously, something else was the problem

Today, biggest problems on the web

Information overload

Content discovery

Search

User modeling

Often generic methods

the modeling problem: model+prediction

the core problem: estimating preferences

getting past 80%

the efficiency of data

This paper: A more personal approach

Aggregated user modeling methods for truly personal predictions.

Hypothesis

Contributions

Outline

— 2 —
Background Theory

This chapter will introduce some basic theory needed to develop our approach to user modeling. We will first describe our stated enemy, the information overload problem, before delving into how user modeling and, more specifically, recommender systems, is currently used to solve this problem.

This chapter will also introduce the notion of personalized search, a field where our user modeling method will be especially applicable. The next chapter will use these theories to build an *even more personalized approach* to user modeling.

2.1 Information Overload

Information overload conveys the act of receiving *too much information*. The problem is apparent in situations where decisional accuracy turns from improving with more information, to being hindered by too much irrelevant data (Bjorkoy, 2010, p13). Needness to say, this is a widespread phenomenon, with as many definitions as there are fields experiencing the problem. Examples include *sensory overload*, *cognitive overload* and *information anxiety* (Eppler and Mengis, 2004).

Two common tasks quickly become difficult in this situation: Consuming content that is known by the user to be relevant can be drowned out by irrelevant noise. Orthogonally, discovering new, interesting yet unknown content also becomes difficult because of the sheer amount of available content. Finding contemporary examples is not difficult:

- Missing important news articles that get drowned out by irrelevant content.
- Forgetting to reply to an email as new messages keep arriving.
- Discovering sub-par movies because those most relevant are never discovered.

The overload is often likened to a *paradox of choice*, as there may be no problem acquiring the relevant information, but rather identifying this information once acquired. As put by Edmunds and Morris (2000): "The paradox — a surfeit of information and a paucity of useful information." While normal cases of such overload typically result in feelings of being overwhelmed and out of control, Bawden and Robinson (2009) points to studies linking extreme cases to various psychological conditions related to stressful situations, lost attention span, increased distractibility and general impatience.

Kirsh (2000) argues that "the psychological effort of making hard decisions about *pushed* information is the first cause of cognitive overload." According to Kirsh, there will never be a fully satisfiable solution to the problem of overabundant information, but that optimal environments can be designed to increase productivity and reduce the level of stress through careful consideration of the user's needs. In other words, to solve the problems of information overload and content discovery, applications must be able to individually adapt to each user.

An insightful perspective on information overload comes from the study of attention economy. In this context human attention is seen a scarce commodity, offset by how much irrelevant noise is present at any given time. Attention can then be defined as "... focused mental engagement on a particular item of information. Items come into our awareness, we attend to a particular item, and then we decide whether to act" (Davenport and Beck, 2001). To evade information overload is then to maximize available attention, allowing more focus on the most important items of an interface.

Conceptual models used in interaction design can help us see when and where information overload interferes with the user experience. Norman (1988) advocates a model called the seven stages of action, describing how each user goes through several states while using a system (see Figure 2.1, adapted from Norman). First, the user forms a goal and an intention to act. The user then performs a sequence of actions on the world (the interface) meant to align the perceived world and the goals. After performing a set of actions, the new world state is evaluated and perceived. At last, the user evaluates the perception and interpretation of the world in accordance with the original goal.

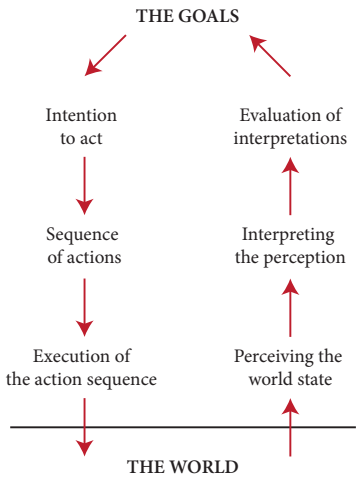


Figure 2.1: Stages of Action

As apparent from this model, information overload can interfere both before and after any action is taken. For example, if the application presents too much content, or presents content in a confusing manner, it can be difficult for the user to identify which actions that would help achieve the current goal. Likewise, after actions are taken, the new world state can suffer the same shortcomings of overwhelming scope or lack of presentations, leading to information overload. This precludes the user from properly evaluating the resulting application state.

In short, an application interface can fail both before and after a user tries to interact with it. Information overload happens throughout the interaction process, which is important to know when considering possible solutions.

2.1.1 Online Overload

The Web is a common source of information overload. As we will use the web as an example throughout this paper, this section describes why the Web is so conducive to information overload.

Online information overload is especially pervasive when considering *content aggregating websites*, i.e. sites that collate information from multiple other sites and sources. Online information retrieval (i.e. search engines), fall into this category, as does online newspapers, feed readers and portal websites. As mentioned, the wealth and scope of data are natural culprits of online overload, as well as the varying qualities of websites publishing the information. However, lessons from graph theory can also help us see why information overload occurs on the Web.

Graph theory presents applicable models of the Web that characterize how people navigate between websites, and show how content aggregators form important hubs in the network. These models also show a theoretical foundation for why information overload occurs. In the Web graph, nodes correspond to websites and directed edges between nodes are links from one page to another. The *degree* of a node is defined as its number of edges.

The Internet has the properties of a *small-world network* (Newman and Moore, 2000), a type of random graph, where most nodes are not neighbors, but most nodes are reachable through a small number of edges (See Figure 2.2). This is because of important random shortcuts differentiating the graph from a regular lattice. The graph is not random, but neither is it completely regular. As described by Barabási (2003, p37), the average number of outbound links from a webpage is around 7. From the first page, we can reach 7 other pages. From the second, 49 documents can be reached. After 19 links have been traversed, about 10^{16} pages can be reached (which is more than the actual number of existing web pages, since loops will form in the graph).

The high degree of the Web graph would suggest that finding an optimal path to your desired page is quite difficult. Yet, while it is true that finding the *optimal path* is hard, finding *a good path* is not that big a challenge. When people browse the Web, links are not followed blindly — we use numerous different heuristics to evaluate each link, often

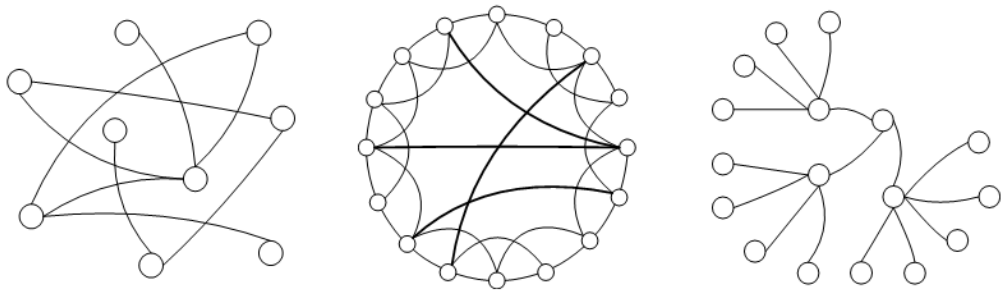


Figure 2.2: Complex Networks, from the left: A random *network, a* small-world *network and a* scale-free *network (which is a type of small-world network). Figure adapted from Huang et al. (2005).*

resulting in a quite good path to where we want to go. So why is the Web still quite challenging to navigate?

As discovered by Albert et al. (1999), the Web also exhibits properties of a *Scale-Free Network* (SFN). They found that in some natural observed networks, there exists a small number of nodes with an extremely high degree. This is also true on the Web — some websites have a huge number of outbound links. For comparison, while a random network is similar to a national highway system, with a regular number of links between major cities, scale-free networks are more like an air traffic system, with central hubs connecting many less active airports (Barabási, 2003, p71).

These highly connected nodes, called *hubs*, are not found in small-world networks or random graphs. As demonstrated by the presence of hubs, the degree distribution of a scale-free network follows a power law, $P(k) \sim k^{-\gamma}$, where $P(k)$ is the probability of a node having k connections and γ is a constant dependent on the type of network, typically in the range $2 < \gamma < 3$. Since the Web has directed edges, we have two power laws: $P_{in}(k) \sim k^{-\gamma_{in}}$ and $P_{out}(k) \sim k^{-\gamma_{out}}$.

Albert et al. (1999) describes a number of studies placing the γ values for the Web in the $[2, 3]$ range, with γ_{out} being slightly higher than γ_{in}. Both these probabilities exhibit power tails (or long tails). In other words, a few important nodes have a huge number of inbound and outbound links — the hubs. Barabási (2003, p86) proposed that hubs emerge in a scale-free networks because of two factors: (1) Growth: Nodes are added to the network one by one, for example when new websites are added to the Internet. (2) Preferential attachment: When new nodes are created, they connect to existing nodes. The probability that the new node will connect to an existing node is proportional to the number of links the existing node has. In other words, older, more established and

central nodes are preferred neighbors.

This is called the Barabási-Albert model (Albert et al., 1999), and the probability for a new node connecting to an existing node is given by $\prod k_i$ in Equation 2.1, where k_i is the number of links pointing to node i.

$$\prod_i k_i = \frac{k_i}{\sum_j^N k_j} \tag{2.1}$$

Another important topological observation of the Web is how fragmented it is. Barabási (2003, p166) describes the Internet as a number of continents:

The first continent is the *central core*, where most important hubs reside. On this continent, most sites are easily reachable through each other. The second is the *in-continent*, and is defined by the group of websites that often link to sites in the central core, but seldom receive reciprocal links from central hubs. The third is the *out-continent*, which are comprised of sites that are often linked to from the central hub, but seldom link back. Finally, the Internet has many *islands*, or dark nets, which are not accessible through links from other continents. The islands of the Web is why search engines allows web masters to manually add their sites to the search engine index — not all sites are discoverable by following simple links. Some of the Web is topographically close, but a lot of it is not.

Returning to our main topic, the *hubs* often represent the previously mentioned *content aggregating websites*. Search engines, social link aggregators, news portals, et cetera are all hubs of the Internet, emerging from the preferential link attachment of newly created nodes. Factor in the existence of multiple sub-graphs, or continents, and we can intuitively see that navigating the Web is not as easy as it might appear from simple models.

What does seem clear is that these content aggregating hubs are prime candidates for overwhelming their users with information. The fundamental observed structure of the Web creates the need for information brokers that link the net together, and the need for techniques to display a lot of data.

So far we have established that information overload is a pervasive problem, especially on the web. The question now becomes how to best solve this issue. This is where user modeling comes in.

2.2 User Modeling

The term *user modeling* (UM) lacks a strict definition. Broadly speaking, when an application is adapted in some way based on what the system knows about its users, we have user modeling. From predictive modeling methods in machine learning and how to implement these methods, to how interface design is influenced by personalization — the field covers a lot of ground.

It is important to differentiate between adapting the interface of an application and the content of an application. Many user modeling methods strive to personalize the interface itself, e.g. menus, buttons and layout of interface control elements (Jameson, 2009; Fischer, 2001). Adapting the application content, on the other hand, means changing how and what content is displayed For instance, interface adaption might mean changing the order of items in a menu, while content adaption might mean changing the order and emphasis of results in a web search interface.

We are interested in adapting the content of an application since the source of our overload problem often comes down to a mismatch between presented content and desired content. Examples of such user modeling include:

- Translating content based on a user's geographical location.
- Suggesting interesting items based on previous activity.
- Reorganizing or filtering content based on predicted user relevance.
- Changing the presentation of content to match personal preferences or abilities.

The fields of Artificial Intelligence (AI) and Human-Computer Interaction (HCI) share a common goal solving information overload through user modeling. However, as described by Lieberman (2009), they have different approaches and their efforts are seldom combined: while AI researchers often view contributions from HCI as trivial cosmetics, the HCI camp tends to view AI as unreliable and unpredictable — surefire aspects of poor interaction design. Luckily, according to Kobsa (2001), recent research has blurred the lines between the AI and HCI in user modeling.

In AI, user modeling refers to precise algorithms and methods that infer knowledge about a user based on past interaction (e.g. Pazzani and Billsus (2007); Smyth (2007); Alshamri and Bharadwaj (2008); Resnick et al. (1994)). By examining previous actions, predictions can be made of how the user will react to future information. This new knowledge is then embedded in a model of the user, which can predict future actions and reactions. For instance, an individual user model may predict how interesting an

unseen article will be to a user, based on previous feedback on similar articles or the feedback of similar users.

HCI aims to meet user demands for interaction. User modeling plays a crucial role in this task. Unlike the formal user modeling methods of AI, user models in HCI are often cognitive approximations, manually developed by researchers to describe different types of users (e.g. Fischer (2001); Jameson (2009); Cato (2001)). These models are then utilized by interaction designers to properly design the computer interface based on a models predictions of its user's preferences. Totterdell and Rautenbach (1990) describes user modeling in interaction design as a collection of deferred parameters: "The designer defers some of the design parameters such that they can be selected or fixed by features of the environment at the time of interaction [...] Conventional systems are special cases of adaptive systems in which the parameters have been pre-set."

This paper is concerned with the AI approach to user modeling, and in particular, the use of *recommender systems (RS)*. As our goal is to combine different RSs into one coherent user model, we will now describe what an RS entails, and introduce some of the many algorithms they employ.

2.3 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 $f : U \times I \to S$. 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.

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, 2007a, 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.

2.3.1 *Predicting Ratings*

The crucial 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 2.1). 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

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

Table 2.1: A taxonomy of recommender systems. From Bjorkoy (2010).

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 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. (2007b); 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))

2.3.2 *Examples of Recommender Systems*

Because our solution will combine different recommender systems, we need a short introduction to some of the approaches we will combine. Let us take a closer look at (1) *baseline ratings*, (2) *neighborhood estimation*, (3) *dimensionality reduction*, and (4) *network traversal*. This is by no means an exhaustive list, but rather a quick rundown of common approaches in recommender systems, that we will use in the next chapter. 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 (by punishing greater values, 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) *Neighborhood estimation* is part of many recommendation systems. It is the core principle behind most collaborative filtering algorithms, that estimate an unknown rating by averaging the ratings of similar items or users, weighted by this similarity. These approaches often work in two steps: First, a neighborhood if similar elements is computed. Second, the similarities and connections within this neighborhood is used to produce a prediction.

The principal method for computing user similarity is the *pearson correlation coefficient* (PCC) (Segaran, 2007, p11). While simple, the PCC compares favorably to more complex approaches, and is often used as a benchmark for testing new ideas (e.g. in Lemire and Maclachlan (2005); Ujjin and Bentley (2002); Konstas et al. (2009)).

The PCC is a statistical measure of the correlation between two variables. In our domain, the variables are two users, and their measurements are the ratings of co-rated items. The coefficient produces a value in the range $[-1, 1]$ where 1 signifies perfect correlation (equal ratings), 0 for no correlation and -1 for a negative correlation. The negative correlation can for example signify two users that have diametrically opposing tastes in movies. We compute PCC by dividing the covariances of the user ratings with their standard deviations:

$$\text{pcc}(u,v) = \frac{cov(R_u, R_v)}{\sigma_{R_u} \sigma_{R_v}}.$$

When expanding the terms for covariance and standard deviations, and using a limited neighborhood size n, we get

$$\text{pcc}_n(u,v) = \frac{\sum_{i\in K}^{n}(R_{ui} - \bar{R}_u)(R_{vi} - \bar{R}_v)}{\sqrt{\sum_{i\in K}^{n}(R_{ui} - \bar{R}_u)^2}\sqrt{\sum_{i\in K}^{n}(R_{vi} - \bar{R}_v)^2}}.$$

The limited neighborhood size becomes the statistical sampling size, and is a useful way of placing an upper bound on the complexity of computing a neighborhood. n does not have to be a raondom sampling — it can also be limited by the number of ratings the two compared users have in common, the number of ratings each user have, or something similar, as denoted by K in our the formula.

After a neighborhood is determined, it is time to predict our desired rating. When using *collaborative* recommenders, this means averaging the neighborhood ratings weighted by similarity (Segaran, 2007, p16):

$$\bar{r}_{ui} = \frac{\sum_{v\in K(u,i)} \text{sim}(u,v) \times R_{vi}}{\sum_{v\in K(u,i)} \text{sim}(u,v)},$$

where $\text{sim}(u,v)$ is the similarity between two users, $K(u,i)$ is the set of users in the neighborhood of u that have rated item i. This is possible the simplest way of computing a neighborhood-based prediction. Most systems use more complex estimations. For instance, Koren (2008) uses the baseline ratings discussed above instead of plain user and item ratings, to remove what they call global effects where some users are generous or strict in their explicit preferences, and some items are consistently rated differently than the average.

Content-based recommenders based on neighborhoods work on similar principles. The difference is that we compute similarities between items, not users. The simplest approach is to find items highly rated by the current user, compute the neighborhood by finding items similar to these, and produce ratings by weighting the initial rating with the similarity of the neighboring items.

The PCC is but one of many methods used to compute neighborhoods. Other simple measures include the *euclidean distance* (Segaran, 2007, p10), Spearman's or Kendall Tau rank correlation coefficients (Herlocker et al., 2004, p30) — variations on the PCC. Similarity metrics from information retrieval, such as *cosine correlation* from the *vector space model* (VSM) are also popular, often used to compute item similarities (see section 2.4).

Bell and Koren (2007b) shows a more sophisticated neighborhood estimation which computes global interpolation weights, that can be computed simultaneously for all

neirest neighbors. Combinations of metrics are also possible. Ujjin and Bentley (2002) first use a simple euclidean metric to gather a larger neighborhood, which is then refined by a *genetic algorithm*. Another way of computing neighborhoods is by reducing the dimensions of the ratings matrix, as we will now introduce.

(3) *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).

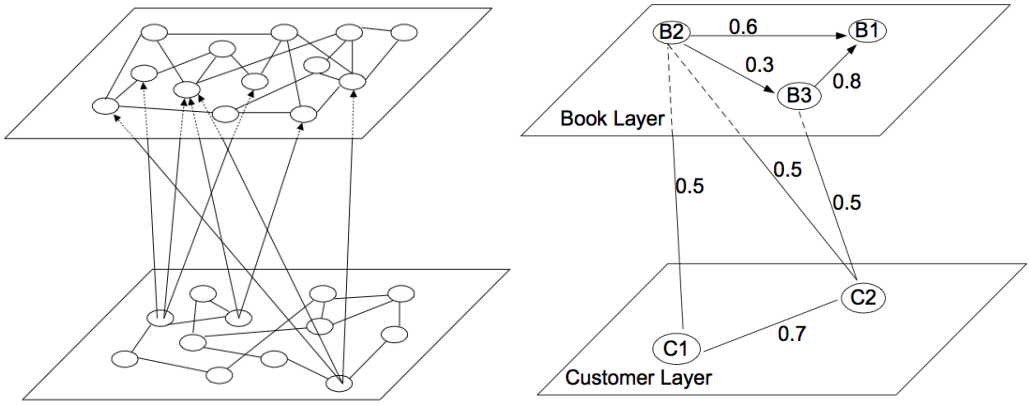
Singular Value Decomposition (SVD) is a common method for such matrix factorization (e.g. Billsus and Pazzani (1998, p5), Sun et al. (2005), Bell et al. (2007b)). This is the same underlying technique used by *latent semantic indexing* (LSI) in information retrieval (Baeza-Yates and Ribeiro-Neto, 1999, 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



(a) *(b)*

Figure 2.3: Network Traversal: (a) A graph with two kinds of nodes, e.g. items and users. (b) A graph with books and customers, where recommendations can be made by traversing the weighted connections. Connections between nodes of the same type represent similarity, while connections between books and customers represent purchases. Figures from Huang et al. (2002).

reduced factors to find similar users by their cosine similarity (see Section 2.4). We can of course also find similarities between items, clusters of items and user and so on, all based on latent "categories" discovered by the automatic identification of patterns in the data. 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.

(4) *Network traversal* refers to estimating predictions by traversing a graph of users and items to provide recommendations. The connections between nodes can be any type of relation that makes sense to the RS. Examples include linking item and user nodes based on purchases or explicit ratings, linking user nodes from asymetrical (directed edges) symetrical (undirected edges) relations, or linking items based on some similarity metric.

Huang et al. (2002) used network traversal to create a simple system for recommending books to customers. Here, edges between items and users correspond to ratings, and edges connecting nodes of the same type are created by connecting elements that have similarity above a certain threshold. Predictions are generated by traversing the graph a preset number of steps starting at the current user, and multiplying the weights of paths leading to the target item (see Figure 2.3).

The complexity of recommender systems based on networks are only limited by the kinds of relations we can produce. For example, recommending other users in social

networks can easily utilize friend or friend-of-a-friend relations to find others the current user might be interested in connecting to. Indeed, any relevant similarity metric can be used to connect nodes of the same type, or nodes of different types.

One variation comes from Walter et al. (2008), who create a network of *transitive trust* to produce recommendations. Here, the neighborhood of users is determined by traversing through users connected by a level of trust. The trust can for example be a function of how many agreeable results the connection to a user has produced. In other words, users trust each others recommendations based on previous experience.

Konstas et al. (2009) takes yet another approach that measures the similarity between two nodes through their *random walks with restarts* (RWR) technique.

Starting from a node x, the RWR algorithm randomly follows a link to a neighboring node. In every step, there is a probability α that the algorithm will restart its random walk from the same node, x. A user-specific column vector $\mathbf{p}^{(t)}$ stores the long term probability rates of each node, where $\mathbf{p}_i^{(t)}$ represents the probability that the random walk at step t is at node i. $\mathbf{S}$ is the column-normalized adjacency-matrix of the graph, i.e. the transition probability table. $\mathbf{q}$ is a column vector of zeroes with a value of 1 at the starting node (that is, $\mathbf{q}_i$ is 1 when the RWR algorithm starts at node x). The stationary probabilities of each node, signifying their long term visiting rate, is then given by

$$\mathbf{p}^{(t+1)} = (1 - \alpha)\mathbf{S}\mathbf{p}^{(t)} + \alpha\mathbf{q}$$

when it is run to convergence (within a small delta). Then, the *relatedness* of nodes x and y is given by $\mathbf{p}_y$ where p is the user model for the user represented by node x. Konstas et al. found that this approach outperformed the PCC, as long as the social networks were an explicit part of the system in question. In other words, the connections between users had to be one actively created by the users to be of such quality and precision that accurate predictions could be made.

After this whirlwind tour of recommender systems, it is time to look at some closely related topics: information retrieval and personalized search. This will form the basis for the case study performed in the next chapter.

2.4 Personalized Search

Personalized search means adapting the results of a search engine to each individual user. As we shall see, this field has a lot in common with user modeling and recommender system. In both situations, we wish to predict how relevant an item will be to each user. Before delving into the techniques of personalizing search results, we present the basics of *information retrieval* (IR).

2.4.1 Information Retrieval

Manning et al. (2008, p1) define IR as "finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)".

How does this relate to recommender systems? There is an important distinction: The purpose of *recommending* is twofold: (1) show the user items similar to another item, and (2) allow discovery of relevant items the user did not know exist. The purpose of *search* is a bit different: allow the user to find the location of information he or she knows (or hopes) exists. In other words, the defining separator is often the knowledge of existence.

However, as we shall see in this chapter, the two fields employ a lot of the same methods and terminology. In the next chapter, we will show how these can work together.

Baeza-Yates and Ribeiro-Neto (1999, p23) presents a formal definition of an IR system: IR $= (Documents, Queries, Framework, ranking(q_i, d_i))$

As evident by the scope of IR literature, these elements can be just about anything that has to do with retrieving information. However, in what is often called *classic IR*, the documents contain free text with little to no describing structure, and the queries are short user-initiated descriptions of an *information need* (Baeza-Yates and Ribeiro-Neto, 1999, p19). Among other domains, this model describes web search engines, where the documents are web pages and queries are short sentences or a few keywords input by users.

The *Framework* in our quadruple refers to how documents are stored and retrieved. Basic approaches to IR split each document into a set of terms (e.g. words), and create an inverted index (Manning et al., 2008, p22) that lists each document that each term appears in. There are numerous extensions to this framework, including:

- Positional information for phrase search (Manning et al., 2008, p39)
- Stop word removal (removing the most common terms) (Manning et al., 2008, p27)
- Stemming (reducing words to their root forms) (Manning et al., 2008, p32)
- Lemmatization (contextual inflection removal) (Manning et al., 2008, p32)
- Query reformulation (Baeza-Yates and Ribeiro-Neto, 1999, p117)

All these techniques help improve (among other things) the *recall* and *precision* of the retrieval engine. Recall, precision and relevance are well defined masures for evaluating the quality of a search engine (Manning et al., 2008, p5):

- A document is *relevant* if it satisfies the user's information need.
- *Recall* is the fraction of relevant documents retrieved by the system.
- *Precision* if the fraction of retrieved documents that are relevant.

There are many more measures, but recall and precision succintly define what a search engine must to to be successfull: retrieve many relevant documents and few irrelevant documents. Failing this test is to neglect the main purpose of an IR: preventing information overload by allowing people efficient access to relevant parts of an otherwise overwhelming information repository.

To us, however, the most interesting part of any IR system is the *ranking function*. This function maps queries to documents by a scalar score, signifying how well a match each document is to a query. The relation to recommender systems should be self-evident, and indeed, IR systems use many of the same metrics to measure query/document similarity.

A common framework for storing and ranking documents is the previously mentioned vector space model (VSM). This model stores documents as term vectors. Each term represents a dimension, and documents are vectors in this term-space. When performing a query, the query terms are also represented as a vector in the same space. By computing the cosine similarity between the query and each document, we get a good estimate of how well a document matches a query (Baeza-Yates and Ribeiro-Neto, 1999, p29).

The next question is what to store at each (document, term) coordinate in the vector space (called the document-term weights). Storing simple 1 or 0 values representing whether or not terms are present gives a model where a document's relevance is proportional to how many of the query terms it includes. However, this is not very precise. For example, by this definition, a document containing every conceivable query term would be the most relevant to any query. A better idea is to use something like the TF-IDF weighting scheme (Baeza-Yates and Ribeiro-Neto, 1999, p29):

$$w_{t,d} = tf_{t,d} \times idf_t = \frac{\text{freq}(t,d)}{\sum_{k \in d} freq(k,d)} \times \log \frac{N}{n_t}.$$

The final weight is computed by multiplying the term frequency score (TF) $tf_{t,d}$ with the inverse document frequency (IDF) idf_t. TF evaluates how well the term describes the document contents, while IDF punish terms that appear in many documents. $freq_{t,d}$ gives the frequency of a term in a document. N is the total number of documents, and n_t the number of documents in which t appears. The effect of the IDF factor is dampened by taking its log-value. Together, TF and IDF ranks documents higher by words that discriminate well within the document corpus, and ignores words that appear in so many documents that they have little to no predictive capacity.

While simple, TF-IDF has proven itself resilient when compared to more complex methods, and many more complex methods have been built on its foundations (e.g. BM25, one of the most successfull probabilistic weighting algorithms (Robertson, 2010)).

There are as many IR models as there are domains that need search, and even the basic vector space model can be constructed in a myriad of ways. There is also the simpler *boolean search model*, where queries are based on boolean algebra. Probabilistic models frame the similarity question as the probability that the document is relevant. Latent Semantic Indexing (LSI), the application of SVD to IR by performing dimensionality reduction of the term-space into concept-space is another approach. See Manning et al. (2008), Robertson (2010) or Baeza-Yates and Ribeiro-Neto (1999) for a more comprehensive introduction to models in IR.

The important take-away is that, while serving different use cases, RSs and IR systems employ much of the same technology with different input and expected output.

2.4.2 Ranking Signals

Modern web search engines have long ago moved on from simple ranking metrics such as TF-IDF. While similar traditional metrics may form the foundation of modern search engines, a lot more thought go into the final results. Different types of rankings are combined to produce the final *search engine results page* (SERP), with each ranking function often being referred to as a *signal*. Alternate names include *reranking* or *rescoring* functions.

Google, the company behind the popular online search engine, writes: "Today we use more than 200 signals, including PageRank, to order websites, and we update these

algorithms on a weekly basis. For example, we offer personalized search results based on your web history and location."[1] Bing, another popular search engine, uses the same terminology: "We use over 1,000 different signals and features in our ranking algorithm."[2]

Signals are often products of the document structure of the current domain. Sergey and Lawrence (1998, p5) points to the use of the proximity of query terms in matching documents. Those where the terms appear close together are natural candidates for a higher ranking. Other signals, still reliant on the documents themselves, are more domain oriented. Another signal they point out is how words in a larger or bold font can be weighted higher than normally typset words.

Signals can also depend on the query. Manning et al. (2008, p145) describes a system that takes multi-word queries, breaks them up into different permutations and runs each new query against the same document index and ranking function. Each query corresponds to its own ranked set of results, which are sent to a *rank aggregation function* which turns the accumulated ranking evidence into one coherent result. We will have more to say on rank aggregation in Section 2.5.

Signals can also be external to the collection or relational within the collection. PageRank (Sergey and Lawrence, 1998, p4) is perhaps the most known of the relational signals. The algorithm forms a probability distribution over web pages, ranking their percieved authority or importance according to a simple iterative estimation. Each web site is given its rank based on how many pages that link to it. For each page that provides links, the score it contributes to the linked-to page is its own page rank weighted inversely proportional to the number of outbound links the page has. Another intuitive justification for a site's PageRank is the *random surfer model* (Sergey and Lawrence, 1998, p4). The probability that the random surfer visits a page is its PageRank. The "randomness" is introduced by a damping parameter d, which is the probability that a user will stop browsing and start at a new random page:

$$\mathrm{PageRank}(x) = \frac{1-d}{N} + d \sum_{y \in B_x} \frac{\mathrm{PageRank}(y)}{\mathrm{Links}(y)},$$

where B_x is the set of pages linking to page x, and $\mathrm{Links}(y)$ is the number of outbound links from page y. The first term distributes an equal pagerank score to all pages that have no outbound links, as N is the total number of pages. This iterative algorithm is run until convergence inside a small delta.

(1) google.com/corporate/tech.html — accessed 11/04/2011
(2) bing.com/community/site_blogs/b/search/archive/2011/02/01/thoughts-on-search-quality.aspx — accessed 11/04/2011

Let us now finally take a look at one of the main uses of signals: *personalized search*.

2.4.3 *Personalizing Search Results*

Search engines, especially online search engines, face a huge challenge. In addition to the wide range of websites, the ambiguity of language, the restricted nature of queries, comes the wildly differing users. Each user is unique. Even when considering one user, there might be many different use cases, for example when using the same search engine at work and at home. Another identified problem is that users use search engines for navigation as well as pure search. Teevan et al. (2007) found that as many as 40% of all queries to the Yahoo! search engine were "re-finding queries", i.e. attempts to find information the user had accessed before.

Personalized search (PS) attempts to solve these problems by introducing individually catered search results. These techniques are based on user modeling (as introduced in Section 2.2), and attempts to build predictive models based on mined user preferences. Commonly, this is done through query log analysis (e.g. Liu et al. (2002); Sugiyama et al. (2004); Shen et al. (2005); Speretta and Gauch (2000)) In other words, these are often model-based techniques with implicit knowledge gathering agents, that create individual, long-term user models (see Section 2.3).

There are two leading approaches to personalizing search results (Noll and Meinel, 2007, p2). The first method is query reformulation, where the actual user query is enhanced in some way, before traditional IR retrieves and ranks documents. The second method is results re-ranking, where the IR results are sorted based on personalized metrics. This section describes the latter approach.

To demonstrate how these methods compare to traditional recommendation systems, we will explore a few different approaches to personalized search: (1) *personalized topic-sensitive PageRank*, (2) *folksonomy-based personalization* and (3) *social network search ranking*.

(1) Haveliwala (2003) introduced a topic-sensitive PageRank algorithm, that they found to be "generate more accurate rankings than with a single, generic PageRank vector". In essence, they create topic-specific PageRank vectors for a number of pre-set topics, creating many rankings per page, one for each topic. This new PageRank is computed based on an existing set of websites that belong to each topic. Qiu and Cho (2006) achieved "significant improvements" to this approach by adding a personally adaptive layer to the topic-sensitive PageRank algorithm, creating a *personalized PageRank algoriithm*.

In addition to the topic vector, Qiu and Cho creates a topic-preference vector for each user. When the user has clicked on a few search results, a learning algorithm kicks in and estimates approximately how likely the user is to be interested in each of the pre-set topics, creating the topic-preference vector T. When it is time to rank a page p in response to the query q, they compute the personalized ranking:

$$PersonalizedRanking(T, p, q) = \sum_{t=1}^{m} T(i) \cdot Pr(q|T(i)) \cdot TSPR_i(p)$$

We will not deduce this equation here (see Qiu and Cho (2006, p5)), but let us explain it. T is the user-specific topic preference vector. i is the index of a topic and m the total number of topics. $Pr(q|T(i))$ is the probability that the query belongs in topic i. This can be as simple as the total number of times the query terms appear in websites under topic i. $TSPR_i(p)$ is the topic-sensitive PageRank score for page p in topic i. Basically, this is the normal PageRank vector computed within a pre-set topic i.

The construction of $T(i)$, i.e. the training phase of the algorithm, is performed by mining the query logs for each user. By identifying how many sites the user has visited in each topic, computing T can be done through linear regression or by using a Maximum-likelihood estimator (basically, any method that can fit a curve). Qiu and Cho (2006, p10) reports improvements of 25% to 33% over the Topic-sensitive PageRank approach, which Haveliwala (2003) reports outperformed the original PageRank algorithm.

(2) Web applications often have more information about users and items (documents, sites or articles) than simple ratings. One of these extra resources are tags, simple keywords assigned from users to items. The collection of users, items, tags and user-based assignment of tags to resources is called a *folksonomy*.

Hotho et al. (2006) defines a folksonomy as a tuple $F = (U, T, R, Y, \prec)$. Here, U, T and R are finite sets of users, tags and resources (items), respectively. Y is a ternary relation between users, tags and resources, called tag assignments. $\prec$ is a user-specific tag hierarchy, applicable if the tags are organized as super- and sub-tags. The *personomy* P_u is a user-specific part of F, i.e. the tags, items and assignments related to one user u. In our terms, this personomy would be the user model. Hotho et al. use folksonomies to do information retrieval based on their *FolkRank* search algorithm, a derivative of PageRank.

Bao et al. (2007) shows how folksonomies can be used to personalize search. They first create a topic-space, where every user and document are represented. Each tag in the

system is a dimension in this topic-space, or tag-space. Whenever a new query is issued, two things happen: First, a regular IR method computed a standard, non-personalized ranking of documents. Second, a personalized ranking list is computed by performing a simple vector-space model matching in the topic-space, for example by using cosine similarity (as previously explained). The personalized list is then unrelated to the actual query, and is simply a ranking of the most relevant pages to the current user.

The two ranks are aggregated by a simple consensus-metric, the *weighted borda-fuse* (WBS) aggregator (Xu et al., 2008, p3), which is nothing more than a weighted combination of the rankings:

$$\text{rank}(u, q, p) = \alpha \cdot \text{rank}_{IR}(q, p) + (1 - \alpha) \cdot \text{rank}_{personal}(u, p)$$

Xu et al. tried many combinations of weights, topic selection and datasets, with the general conclusion that folksonomy-based personalized search has great potential. If nothing else, this example shows how easily tags can be integrated to provide an individual searching experience.

(3) Carmel et al. (2009) developed a personalized search algorithm based on a user's *social network*. By re-ranking documents according to their relation to with individuals in the current user's social network, they arrived at a document ranking that "significantly outperformed" non-personalized social search (Carmel et al., 2009, p1). Note, however the qualifier "social search" — their project searches through social data within an enterprise, naturally conducive to algorithmic alterations based on social concepts. However, as social data is data just as well, seeing how a personalized approach improves standard IR in this domain, is helpful.

Their approach: First, documents are retrieved by a standard IR method. Second, the user's socially connected peers are also retrieved. Third, the initial ranked list of documents is re-ranked based on how strongly they are connected to the user's peers, and how strongly those peers are connected to the user. The user-user similarity is computed based on a few signals (Carmel et al., 2009, p2), e.g. co-authoring of documents, the use of similar tags (see (2)) or commenting on the same content. The user model also includes a list of terms the current user has employed in a social context (profile, tags, et cetera). This is all done to infer implicit similarity based on social connections.

The algorithm is quite powerful, and combines three types of rankings: The initial IR score, the social connection score, and a term score, where the terms are tags and keywords used by a user. The user model is $U(u) = (N(u), T(u))$, where $N(u)$ are

the social connections of u and $T(u)$ the user's profile terms. First, the score based on connections and terms is computed, weighted by β which determines the weighting of both approaches:

$$S_P(q, d|U(u)) = \beta \sum_{v \in N(u)} w(u, v) \cdot w(v, d) + (1 - \beta) \sum_{t \in T(u)} w(u, t) \cdot w(t, d)$$

Finally, the results are combined with the ranking returned by the IR method (R_{IR}). A parameter α is used to control how much each method is weighted:

$$S(q, d|P(u)) = \alpha \cdot R_{IR}(q, d) + (1 - \alpha) \cdot S_P(q, d|U(u))$$

This approach, while simple, shows how social relations and social annotations can easily be used to personalize a search experience. However, Carmel et al. (2009, p10) notes that the high quality data in their enterprise setting were important to achieve the improved results.

2.5 Aggregate Modeling

So far we have seen a lot of modeling methods, both for recommender systems (RS) and for personalized search (PS). *Aggregate modeling* (AM) is the act of merging two or more modeling methods in some way. A proper aggregation method creates a combined result that is better than either of the individual methods. In other words, the sum is greater than the parts. We have already seen a few examples of aggregate modeling:

- Koren (2008) aggregates global, individual and per-item averages to a baseline.
- Huang et al. (2002) aggregates different types of graph relations into one prediction.
- Haveliwala (2003) combined their personalized PageRank with another approach.
- Carmel et al. (2009) combined classic IR with social relations and annotations.
- Sergey and Lawrence (1998, p5) aggregates signals masured from website structure.

Clearly, aggregation is an important part of both RS and PS. The reasoning behind combining different approaches is that no one methods can capture all the predictive nature of the available data. For example, Bell et al. (2007a) created a recommender systems where the neighborhood- and SVD-based approaches complement each other.

While the neighborhoods correspond to "local effects" where similar users influence each other's preictions, the dimensionality reduction finds "regional effects", major structural patterns in the data. As they say: "Both the local and the regional approaches, and in particular their combination through a unifying model, compare favorably with other approaches and deliver substantially better results than the commercial Netflix Cinematch recommender system ..." (Bell et al., 2007a, p1)

An interesting question is whether or not all hybrid recommenders are aggregators. This is mostly a question of semantics and implementation. Burke (2007, p4) defines a hybrid system as "any recommender system that combines multiple recommendation techniques together to produce its output." Some hybrid methods combine stand-alone methods, and are definitely aggregations. Other methods merge the methods themselves into one implementation that uses the data in different ways. Burke describes a few types of hybrid recommenders:

- Weighted combinations of recommenders.
- Switching and choosing one recommender in different contexts.
- Mixing the outputs and presenting the result to each user.
- Cascading, or prioritized recommenders applied in succession.
- Augmentation, where one recommender produces input to the next.

However, without being to pedantic, these can all be seen as aggregate approaches: Multiple prediction techniques are used to create a result better than any single methods would provide.

There are two main approaches to model aggretation (Liu et al., 2007, p1):

1. Rank (or *order-based*) aggregation (RA) lets each method produce a sorted list of recommendations or search results. These lists are then combined into one final list, through some aggregation method (see Dwork et al. (2001) or Klementiev et al. (2008)). These methods only require the resulting list of items from each method (Aslam and Montague, 2001, p1).

2. Prediction (or *score-based*) aggregation (PA) works on the item- or user-level by combinining predicted scores one-by-one, creating an aggregated result for each element that should be evaulated. These methods require the actual prediction scores for any item from each method (Aslam and Montague, 2001, p2).

2.5.1 *Rank Aggregation*

RA combines multiple result lists into one list by some metric. Dwork et al. (2001) shows a few metrics applicable to meta-search, the combination of results from multiple search engines. Borda's method (Dwork et al., 2001, p6) is based on positional voting, where each result gets a certain number of points from each result set, based on where it appears in the sorted list. Items at the top gets the most points, while lower items gets fewer points. This is in essence a method where each method has a set number of votes (c, the number of results) that they give to each item.

As we saw in Section 2.4, Xu et al. (2008, p3) used a weighted version of this approach to combine an IR and personal approach to result ranking. Aslam and Montague (2001, p3) calls their version of this *Weighted Borda-Fuse*, where the points given from a method to an item is controlled by the weights estimated for each method. Aslam and Montague (2001, p4) also explain a bayesian approach (*bayes-fuse*), that combined with the *naive Bayes* independence assumption produce the following formula:

$$\text{relevance}(d) = \sum_{i \in Methods} \log \frac{\Pr(r_i(d)|rel)}{\Pr(r_i(d)|irr)}.$$

Here, $\Pr(r_i(d)|rel)$ is the probability that document d is relevant given its ranking by method i. Conversely, $\Pr(r_i(d)|irr)$ is the probability that the document is irrelevant. The probability values are obtained through training, and evaluating the results against known relevance judgements. An interesting note is that the standard Borda method does not require training data, while the weighted version and the bayesian approach do. Aslam and Montague (2001, p1) results with these rank aggregations were positive: "Our experimental results show that metasearch algorithms based on the Borda and Bayesian models usually outperform the best input system and are competitive with, and often outperform, existing metasearch strategies."

Liu et al. (2007) presents a rank-aggregation framework, where the task of estimating a ranking function by using training data. They treat this task as a general optimization problem, with results showing that this framework can outperform existing methods (Liu et al., 2007, p7).

Rank aggregation is a substantial topic, with many approaches. The main take-away is that this approach combines list of results into one single results, and experiments show that results superior to the best of the combined methods are attainable. See Aslam and Montague (2001), Liu et al. (2007) or Klementiev et al. (2008) for more information.

2.5.2 *Prediction Aggregation*

Unlike rank aggregation, prediction aggregation (PA) does not deal with lists of results. PA works on the item-level, collecting scalar predictions of an item's relevance from a number of methods, and combining these predictions into a final score.

Aslam and Montague (2001) describe a number of simple approaches: Min, max and sum models combine the individual predictions in some way, or select one or more of the results as the final prediction. Other models use the average, or log-average of the different methods. The linear combination model trains weights for each predictor, and weighs predictions accordingly. At slightly more complex approach is to train a logistic regression model (Aslam and Montague, 2001, p3) over a training set, in an effort to find the combination that gives the lowest possible error. This last method improved on the top-scoring predictor by almost 11% (Aslam and Montague, 2001, p3), showing that even fairly simple combinations have merit.

Early approaches in recommender systems dabbeled in aggregating content-based and collaborative approaches. Claypool et al. (1999) combined the two approaches in an effort to thwart problems with each method. Collaborative filtering (CF) methods have problems rating items for new users, radically different users or when dealing with very sparse data. Content-based (CB) methods do not have the same problems, but are less effective than CF in the long run, as CB does not tap into the knowledge of other users in the system — knowledge that out-performes simple content analysis. In Claypool et al. (1999), the two types of recommenders were used to create a simple weighted result.

Generally, methods for aggregating predictions in the field of machine learning is called *ensemble methods* (EM) (Dietterich, 2000, p1). While most often used to combine classifiers that classify items with discrete lables, these methods are also used for aggregating numerical values (see the numerical parts of Breiman (1996)). Approaches include *bootstrap aggregation* (bagging) and *boosting* for selecting proper training and testing sets, and creating a *mixture of experts* for combining the predictors (Polikar, 2006, p27).

Bell et al. (2007b) took method aggregation to its logical conclusion when winning the Netflix Challenge, by combining 107 individual results from different recommenders: "We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different, complementing aspects of the data. Experience shows that this is very different from optimizing the accuracy of each individual predictor. Quite frequently we have found that the more accurate predictors are less useful within the full blend." (Bell et al., 2007b, p6) In other words, the final result is improved because of the disjoint reasoning performed by the different predictors.

Like RA, PA is an extensive topic. The take-away stays the same: by combining different modeling methods, more patterns in the data can be mined, and the resulting combination can outperform the best performing method. This is key to the model we shall build in the next chapter.

This chapter has introduced the basic theory we will need to develop our take on user modeling. The next chapter will build this model, While Chapter 4 presents our evaluation experiments.

— 3 —
Methods & Implementation

In this chapter we will build our approach to user modeling: blending multiple predictors on per-user basis. We will first explain the reasoning behind our hypotheses, before we present a method for personalized aggregation.

These personalized *meta models* are then used to perform prediction aggregation in a recommendation scenario, and rank aggregation in an information retrieval scenario.

3.1 Latent Subjectivity

As seen in Chapter 2, there are lots of ways of predicting unknown relevance scores. In fact, judging by the number of different approaches, the only limiting factor seems to be the different patterns and relations researchers discover in the available data. Different methods leverage the data in different ways to achieve their goals of accurate predictions.

As seen in Section 2.5, aggregate modeling is used to combine different, complimenting methods into one coherent system. By leveraging what we shall call *disjoint patterns* in the data, several less than optimal predictors can be combined in a way that outperforms the best of the combined predictors.

However, there exists an underlying subjectivity to relevance prediction that is seldom discussed. When a researcher or developer elects to use some modeling method in order to represent users of their system, a concious selection of applicable measures is made. Consider the following relevance judgements:

- PageRank (Sergey and Lawrence, 1998) assumes that the relevance of a web page is represented by its authority, as computed from inbound links from other sites.
- When providing personalized search results, one ranking signal may be the social connections of the current user. Items deemed relevant by the user's peers will then recieve a boosted ranking (e.g. Carmel et al. (2009)).
- When determining the relevance of an email, one predictor might be based on how often the sender sends emails to the current user.
- When recommending movies, one predictor may be based on the ratings of users with similar profile details. Another predictor might be dependent on some feature, e.g. production year of well liked movies.

- Recommendations based on the Pearson Coefficient (Segaran, 2007, p11) assumes that the statistical correlation between user ratings defines their similarity.

Are these metrics subjective? While the methods themselves do not discriminate, their selection reflects how whoever created the system assumes how each user can and *should* be modeled. This *latent subjectivity* is not desirable. For example, While one user might appreciate a social influence in their search results, another user might not. While one user might find frequency of communication maps well to relevance, another might not. One user might feel the similarity of movie titles are a good predictor, while another might be more influenced by production year. The exact differences are not important — what is important is that they exist.

Another way of explaining latent subjectivity is that *user modeling methods are dependent on the subjective assumptions of their creators*. In other words, each modeling method use some aspect of available data to make predictions, and the importance of each of these methods is determined by whoever creates the system, or by the on average best error-minimizing combination, not by each individual user. Aggregate modeling methods face the same problem of misplaced subjectivity: Aggregation is done on a generalized, global level, where each user is expected to place the same importance on each modeling method. While the aggregation is of course selected to minimize some error over a testing set, the subjective nature remains: The compiled aggregation is a generalization, treating all users the same — hardly a goal of user modeling.

We propose a method where these descisions are left to each user, providing an extra level of abstraction and personalization. This leaves the subjective nature of modeling method selection where it should be: In the hands of each individual user. If each method is *only used* based on how well it performs for each user, any possibly applicable user modeling method suddenly becomes a worthy addition. Consider the following two questions:

1. What combination of which methods will accurately predict unknown scores?
2. Which methods could possibly help predict a score for a user?

The first question is what has to be considered in traditional modeling aggregation: First a set of applicable methods leveraging disjoint patterns must be selected. Then, an optimal and generalized combination of these must be found, most often through minimizing the average error across all users.

The second question is quite different. Instead of looking for an optimal set of methods and an optimal combination, we look for the set of *any applicable method* that *some users* might find helpful. We believe this is a much simpler problem: instead of trying to generalize individuality, it should be embraced, by allowing users to implicitly and automatically select which methods they prefer, from a large set of possible predictors.

3.2 Hypotheses

So far we have taken a look at approaches to recommendations and personalized search: Traditionally, a single method that considers some pattern in the data is used. Modern techniques blend different methods to achieve a combined accuracy that outranks the performance of each individual method.

However, based on the notion of latent subjectivity in method selection and aggregation, we desire an even more personalized technique. To this end, this paper will test three hypotheses (H1-H3), in an effort to establish whether per-user aggregation is a viable technique. First:

H1*: The accuracy of user-item relevance predictions can be improved by blending multiple modeling methods on a per-user basis:* $\forall m \in M : \mathrm{error}(am_u) < \mathrm{error}(m)$.

It stands to reason that if a recommender system is indeed impaired by the subjective selections of modeling methods, a per-user blend of these methods should outperform each of the individual approaches. Second:

H2*: A per-user aggregation method can outperform global and generalized blending methods:* $\forall a \in A : \mathrm{errorr}(am_u) < \mathrm{error}(a)$.

If our assumption that model aggregation inherits the subjective nature of its chosen parts, a per-user aggregation without such misplaced subjectivity should outperform a generalized blending. Third:

H3*: The result set from an information retrieval query can be personalized by blending multiple modeling methods on a per-user basis.*

As described in Section 2.4.2, every measure that influence the final ranking of results from a search engine can be seen as individual signals. Each signal, be it an IR score, something like PageRank or the result of a user modeling method, contributes to the final ranking. As the aggregation of these signals are the same problem faced when

aggregating recommender systems, a per-user approach to blending signals should be able to produce a set of personalized search results. In other words, techniques from information retrieval and user modeling should be combined on a per-user basis to provide individually adapted results.

3.3 User Meta Modeling

So, how do we perform personalized method aggregation? Let us first define a few terms. We define *Meta Modeling* (MM) as using one modeling method to adapt other modeling methods. More specifically, *User Meta Modeling* (UMM) means adapting user modeling methods with another user modeling method. This leaves us with two distinct levels of user modeling methods: the *methods level* and the *aggregation level*. Formally, a system for UMM can be described as a 6-tuple:

$$\begin{aligned} \text{UMM} &= (Items, Users, Ratings, Framework, Methods, Aggregation) \\ &= (I, U, R, F, M, A). \end{aligned}$$

We have a set of $Items$ and a set of $Users$. There is also a set of $Ratings$: each user $u \in U$ can *rate* an item $i \in I$. As before, we use the term "rating" loosely — other applicable and equivalent terms include *relevance*, *utility*, *connection strength* or *ranking*. In other words, this is a measure of what a user thinks of an item in the current domain language. However, since "rating" will match the case study we present later in this chapter, that is what we shall use.

The $Framework$ variable specifies how this data is represented. The two canonical ways of representing users, items and ratings are graphs and matrices (see Section 2.3). We shall use a matrix, where the first dimension corresponds to users, the second to items, and each populated cell is an explicit rating:

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

As we are dealing with multiple approaches to user modeling, we have a set of $Methods$ that each create their own user models. This corresponds to the *methods layer*. Each

model $m \in M$ are used to compute predictions, i.e. estimations of unknown ratings. As demonstrated in Chapter 2, there are many different forms of user modeling, that each consider differents aspects of the available data: the users, items and ratings, as well as other sources such as intra-user connections in social networks or intra-item connections in information retrieval systems. Examples include Slope One, SVD and Nearest Neighbor weighted predictions (see Section 2.3.2). These methods predict unkown connections between users and items based on some pattern in the data, for example user correlations or social connections.

The $Aggregation$ part of our 6-tuple refers to how the predictions from the different methods are blended into one prediction. This corresponds to the *aggregation level*. To achieve the best possible compounded result, we wish to use methods that look at disjoint patterns, i.e. complementary predictive parts of the data (see Section 2.5). As found by Bell et al. (2007b, p6) the accuracy of the combined predictor is more dependent on the ability of the various predictors to expose different aspects of the data, than on the individual accuracy of each predictor. As described in Section 2.5, multiple prediction results are normally combined into a final singular result, based on a generalized combination found by minimizing some error across all users.

Another way of describing (and implementing) the two modeling levels is through application of the map and reduce functions of functional programming. When performing *prediction aggregation* (scores), this estimation can be expressed as

$$\hat{r}_{ui} = \mathrm{reduce}(u, \mathrm{map}(M, u, i)).$$

First, each modeling method is applied through the map function, with the current user and rating for which a rating should be estimated. This produces a set of scalar prediction values. These values are then combined through the reduce method, which takes the predictions and current user as input. In our case, this is the personalized aggregation method. If we wish to do rank aggregation (i.e. sorted lists), the equation is a bit different:

$$\tau_{u,n} = \mathrm{reduce}(u, \mathrm{map}(M, u, n)).$$

Here, τ_u is the list of recommended items for user u (following the notation in Dwork et al. (2001, p3)). Note that there is no input item in this formula as we wish to produce a ranking of the top n recommended items.

Expressing ourselves in terms of map and reduce now is helpful, as this will later guide our implementation of these operations in a proper MapReduce framework for parallell computation (as explained in Manning et al. (2008, p75)). The map function may apply each modeling method in parallell, as these are independent computations. The modified reduce function, which takes the resulting predictions map and the current user as inputs, serve as our personalized aggregator. Let us now describe how to make this function.

3.3.1 *Personalized Aggregation*

To perform UMM, we need the $Aggregation$ (or reduce) method to be an actual user modeling method, that adapts the blending with respect to each user. Until now we have talked about both prediction aggregation (scores) and rank aggregation (sorted lists). For now we shall stick to scalar predictions, but will return to rank aggregation in Section 3.5.

The simplest generalized way of prediction aggregation is a simple avereage over all predictions made by the different methods (e.g. Aslam and Montague (2001, p3)):

$$\hat{r}_{ui} = \frac{1}{N} \sum_{m \in M} p_m(u, i).$$

$\hat{r}_{ui}$ is the estimated rating from user u to item i, N is the number of methods in M, and $p_m(u, i)$ is the predicted rating from method m. However, in most cases we wish to weight each method differently (e.g. Claypool et al. (1999)):

$$\hat{r}_{ui} = \sum_{m \in M} w_m \cdot p_m(u, i) \quad \text{where} \quad 0 \leq w_m \leq 1, \quad \sum_{i \in M} (w_i) = 1.$$

Here, w_m is the weight applied to modeling method m. These weights fall in the range $[0, 1]$ and sum up to 1. As described in 2.5, these weights can be estimated using a host of machine learning methods. The known rating data is separated into a training- and testing set, which is used to estimate optimal weights by minimizing some error across the testing set (note that the training set is also split in some way into two sets to create the singular modeling methods). However, as discussed in Section 3.1, this is still a generalized, averaged result across every user. The system assumes that the best average result is the best result for each individual user.

So, in order to leverage as many data patterns as possible and to remove the latent subjectivity, we wish the weights to be user-specific. The simplest approach is to create secondary user models for aggregation. Intuitively, a user-specific weight vector could fill the role of this user model:

$$\hat{r}_{ui} = \sum_{m \in M} w_{um} \cdot p_m(u, i), \quad \text{where} \quad 0 \leq w_{um} \leq 1, \quad \forall u \in U : \sum_{i \in M} (w_{ui}) = 1.$$

where w_{um} is the user specific weight for method m. In other words, w_u is the user model vector. Each user then has a pesonal set of weights describing how much they prefer each modeling method. These weights can be estimated much in the same way as generalized weights, with the same training and testing set, just on a per-user basis.

However, a personalized linear combination of modeling methods is not enough. Because of the disjoint nature of the differing modeling methods, they essentially each find some reason that an item should receive a certain score. Consider the different patterns recommender systems and information retrieval measures might leverage:

- Ratings from other users weighted by user similarity.
- The similarity of items through the use of an IR method.
- Clustering items and users through global effects, finding general trends.
- Ratings based on local averages, generosity and other individual features.
- Comparing user preference terms with items.

While a linear combination of such disjoint patterns may achieve accurate scores, a nonlinear combination should be able to discover less apparent user preferences. For example, in any case where the agreement of two or more methods are more telling than their contribution to a linear combination, a nonlinear combination can catch such features of the user's preferences.

Consider the task of sorting incoming emails based on user preferences: a user might consider a new mail important if the sender is a social connection and the last communication from this person happened a long time ago. In this case, the agreement between two scoring methods is more important than any other prediction.

Another case is when one method is especially important for some range of its possible predictions. For example, if one method deals with catching possible spam, a confident capture from this method would probably be more important than what any of the other predictions.

In short, we want an aggregation method capable of capturing nonlinear preferences when it comes to combining user modeling methods. Let us express this as an equation:

$$\hat{r}_{ui} = f_u(M(u, i))$$

The user model is the nonlinear function f_u that takes the results of each modeling method $M(u, i)$, and combines them to one prediction. In other words, to achieve a nonlinear personalized aggregation, we need to train one function per user, that takes a set of scalar scores and produces one scalar output. One way of doing this is using a simple neural network, as we shall now explain.

3.3.2 Multilayer Perceptron

An *artificial neural network* (ANN) is a computational model in AI, inspired by aspects of biological neural networks. They are represented as graphs, consisting of a set of nodes (representing artificial neurons) and edges (representing synapeses). ANNs can be used as statistical data modeling tools, capable of learning complex relationships between input and output data (see Russell et al. (1995, p567) or Floreano and Mattiussi (2008, p163)).

A *multilayer perceptron* (MP) is a simple feedforward artificial neural network, that maps a set of scalar inputs to a set of outputs, and is capable of distinguishing data that is not linearly separable (Russell et al., 1995, p578). This fits our needs for a personalized aggregation function. By training a separate MP for each user, based on known ratings from this user, we can create individual nonlinear functions for mapping multiple predictions to one output.

An MP has multiple layers: an input layer, one or more hidden layers, and an output layer. Each layer has a set of nodes. Every node in every layer is connected to all nodes in the next layer with weighted edges (see Figure 3.1). During network activation, values propagate from the input layer, through intermediary nodes and edges, to the output layer.

The input layer nodes corresponds to each input we wish to process, and their value is set to the current raw input values. For example, if the inputs are recommender system methods, each input node corresponds to the prediction from one method. Nodes in the hidden and output layer use a nonlinear activation function to capture patterns in the input data. Depending on the input layer values, nodes in the first hidden layer are

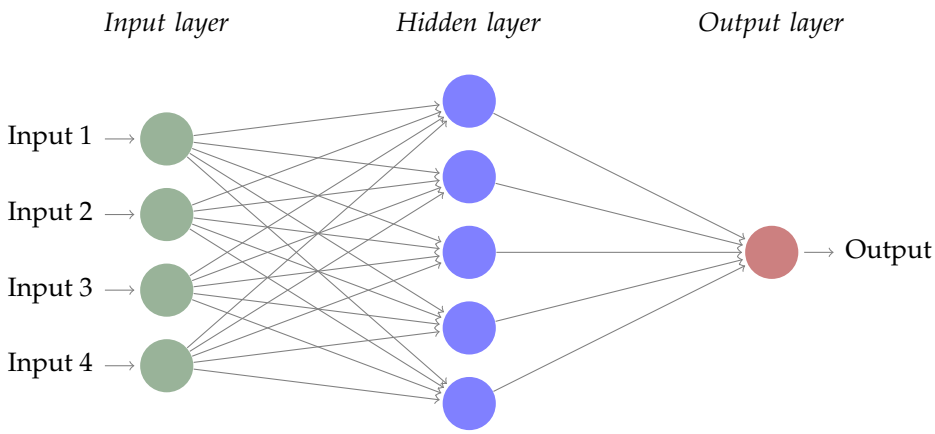


Figure 3.1: A multilayer perceptron with 4 input nodes, 1 hidden layer of 5 nodes and a single output node. Nodes in the hidden and output layers use a nonlinear activation function.

either activated or not, and propagates some value to the following layer. This continues until we reach the output layer, where the node(s) produce some final output value(s).

A node's *activation value* is its actual output value when the network is activated, determined by an *activation function*. The use of hidden layers and nonlinear activation functions is what allows an MP to separate nonlinear patterns. Every activation function relies on the weighted sum of the neuron's inputs a_i (Floreano and Mattiussi, 2008, p178):

$$a_i = \sum_{j=1}^{N} w_{ij} x_j,$$

where i and j are nodes, N is the number of input nodes to node i, w_{ij} is the weight of the connection between the two nodes, and x_j is the activation value of node j. There are many possible activation functions, each applicable to some domains and types of data. Two common choices are the following two sigmoid functions (Floreano and Mattiussi, 2008, p179-180):

$$\Phi(a_i) = tanh(a_i) \quad \text{and} \quad \Phi(a_i) = \frac{1}{1 + e^{-a_i}}.$$

The first function is the hyperbolic tangent which falls in the range $[-1, 1]$, while the second is an equivalent function in the range $[0, 1]$. Other possible activation func-

tions include gaussian distribution functions, ramp and step functions, and trapezoidal functions.

The *backpropagation algorithm* for supervised learning (Russell et al., 1995, p578) is used to train an MP. This iterative algorithm uses a set of training examples to adjust the weights in the network. Each training example is a set of input values with a corresponding correct output value. Backpropagation works in two phases:

(1) The first phase begins by activating the network with an example from the training set, which produces an actual output value based on the current network configuration. The error between this actual output and the desired output from our example is propagated back through the network, which generates deltas (differences) between the actual neuron outputs and desired neuron outputs.

(2) The second phase adjusts the weights of the network edges based on the computed error deltas. This output delta is multiplied with the input activation value to get the gradient of the weight, signifying how much the weight is wrong and in which direction. The weight is adjusted in the opposite direction of the gradient by subtracting some ratio of the gradient from the weight. This ratio is called the *learning rate*, and can be adjusted to change the speed and quality of learning in the network.

When the network is trained, and if it has found *stable patterns* in the training set, we can feed it new input values to produce an unknown output value. More specifically, this is a type of *learned generalization* (Floreano and Mattiussi, 2008, p177). The network learns the generalized connections between input and output, allowing us to identify how input predictions should be aggregated.

3.3.3 Modeling Phase

We shall now use a multilayer perceptron to create a personalized aggregation method. Before we can start blending predictions, we have to create the individualized neural networks. This requires training the user modeling methods, and the network itself. We also have to test the resulting models. Naturally, this makes partitioning the data a bit of a challenge, as we have two successive levels of user models to train. We basically need three types of datasets:

1. Training sets to create the standard modeling methods.
2. Training sets to create the personalized neural networks.
3. A testing set to test our final system.

Constructing these subsets of the available data is a common task in ensemble learning. We shall use an approach known as *bootstrap aggregating* (*bagging*), introduced by Breiman (1996). Originally, bagging is an ensemble learning classification methods, where multiple classifiers are trained by uniformly sampling a subset of the available training data. Each model is then trained on one of these subsets, and the models are aggregated by averaging their individual predictions.

Formally, given a training set D with n items, bagging creates m new training sets of size $n' \leq n$ by sampling items from D uniformly and with replacement. In statistics, these types of samples are called *bootstrap samples*. If n' is comparable in size to n, there will be some items that are repeated in the new training sets.

Bagging suits our needs perfectly, for a few reasons: First, the method helps create disjoint predictors, since each predictor is only trained (or specialized for) a subset of the available data. Second, it allows us to easily train the underlying modeling methods without any complex partitioning of the data. Our partitioning strategy is now clear:

1. Split the entire dataset into a training and testing set.
2. Train modeling methods through bootstrap aggregation of the training set.
3. Train personalized network from each user's ratings from the training set.
4. Test the resulting personalized aggregation model with the testing set.

Each modeling method is trained in ways specific to their implementation. Model-based approaches create pre-built strutures and provide offline training, while heuristic methods simply store the data for future computation. Either way, it is up to each modeling method what it does with the supplied training data.

Training the neural networks for each user is done through the backpropagation algorithm: Each user has a set of known ratings that serve as training examples. For each of these examples, the inputs are the predictions made by the modeling methods. The desired output is the actual rating given by the user. This requires that the underlying models are not all trained with the data point specifying this exact rating, or that the methods disregard this datapoint during the network training phase. Training our entire system then follows the algorithm outlined in Listing 3.1.

The neural networks are heuristic in that they are pre-computed before any predictions are made. After training, each of these user-specific network models must be stored in our system. Creating and saving an individual network for each user might seem like a daunting task, but two characteristics makes this a perfectly valid approach:

```
Algorithm: Training
  Input:
    - R: The ratings matrix
    - U: The set of users
    - M: The set of modeling methods
  Do
    1. For each method in M:
       1. Take new bootstrap sample from R
       2. Train method with this sample
    2. For each user in U:
       1. Select all known user ratings from R
       2. Collect predictions from trained methods for these items
       3. Create training examples from method outputs and known ratings
       4. Train a new MP with these examples
    3. Return
       1. The trained methods
       2. The trained networks
  End
```

Listing 3.1: The algorithm that performs training of a user meta modeling system. Returns the trained methods and networks. This is an offline, pre-prediction training approach.

- The training is performed offline, and can be done at any time. While the initial computational demand might be high, the training is performed before any predictions are made. Prediction performance is much more important than training performance in this scenario as it is during prediction we wish to quickly present results to the user.
- Storing a neural net is as simple as storing the trained connection weights. The overall network structure and activtion functions remain the same for each user. As demonstrated, an MP is a simple network with few nodes, resulting in a simple weight matrix that has to be stored for each user. The size of this matrix corresponds to the number of input methods and the number of hidden layers. This results in compact user meta models that can easily be stored.

There is also the question of when each network should be trained. After all, as a user continues to explicitly rate more items, or as more items arrive in the system, the output from underlying methods will change, and the network must be updated to reflect this new reality. This problem, called *concept drift* is a common occurence in machine learning methods. An optimal strategy would consider retraining the network after some preset number of new ratings or items have been added. As the network emply offline training, this strategy can work in the background, continously producing new or

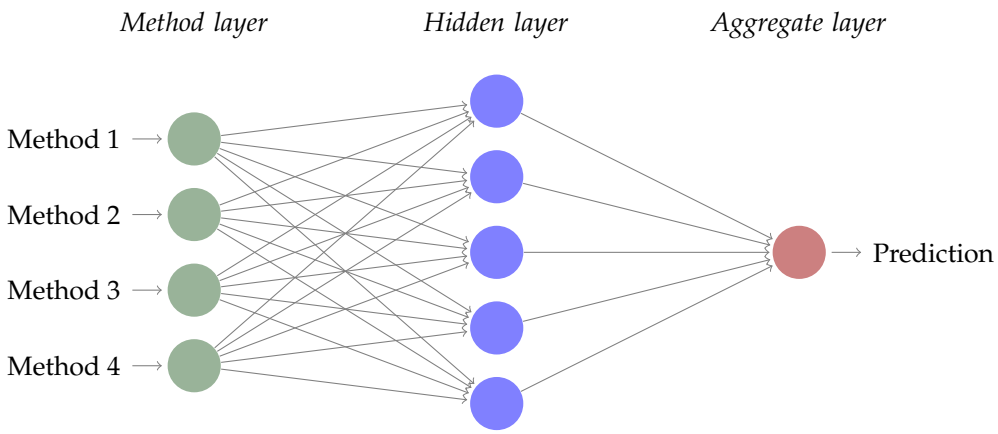


Figure 3.2: The use of a multilayer perceptron for personalized model aggregation. This example takes the results from 4 different modeling methods, feeds them into a pretrained personalized neural net, and creates a combined prediction as output.

updated networks while predictions rely on the previous generation of completed user models.

3.3.4 Prediction Phase

After having created personalized network for each user, it is time to put these networks to good use. This is the prediction phase, where the different predictions for an unknown rating is fed into the network. The output of the network is the personalized aggregate score of the item in question (see Figure 3.2).

The prediction algorithm is given in Listing 3.2.

```
Algorithm: Prediction
  Input:
    - u: A user
    - i: An item
    - n: The network for the current user
    - R: The ratings matrix
    - M: The set of trained modeling methods
  Do
    1. Collect predictions from each method in M for (u,i)
    2. Set the network inputs to the method outputs
    3. Run the network with these inputs
    4. Return the network output
  End
```

Listing 3.2: The algorithm that performs prediction, i.e. estimating the unknown rating from user u to item i.

3.4 Prediction Meta Modeling

3.4.1 Modeling Phase

3.4.2 Prediction Phase

3.4.3 Implementation

3.5 Rank Meta Modeling

3.5.1 Modeling Phase

3.5.2 Prediction Phase

3.5.3 Implementation

— 4 —

Experiments & Results

4.1 Evaluation Strategy

4.1.1 Datasets

4.1.2 Metrics

4.2 Singular Methods

Test H1

4.3 Generalized Aggregation

Test H2

4.4 Retrieval Aggregation

Test H3

— 5 —

Discussion & Conclusion

5.1 Discussion

5.1.1 Results Analysis

5.1.2 Conceptual Findings

5.2 Contributions

5.2.1 Findings

5.2.2 Answering Hypotheses

5.3 Future Work

5.3.1 Complex Individual Functions

5.3.2 Applicability of Different Methods

5.3.3 ...

5.4 Conclusion

— A —
Implementation Details

lorem

A.1 Hadoop

A.2 Mahout

A.3 Neuroph

A.4 JRuby

— B —

Expanded Results

lorem

— C —

Resources

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.

Albert, R., Jeong, H., and Barabási, A. (1999). The diameter of the world wide web. *Arxiv preprint cond-mat/9907038*, pages 1–5.

Alshamri, M. and Bharadwaj, K. (2008). Fuzzy-genetic approach to recommender systems based on a novel hybrid user model. *Expert Systems with Applications*, 35(3):1386–1399.

Aslam, J. a. and Montague, M. (2001). Models for metasearch. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '01*, pages 276–284.

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

Banko, M. and Brill, E. (2001). Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of the first international conference on Human language technology research*, pages 1–5. Association for Computational Linguistics.

Bao, S., Wu, X., Fei, B., Xue, G., Su, Z., and Yu, Y. (2007). Optimizing Web Search Using Social Annotations. *Distribution*, pages 501–510.

Barabási, A. (2003). Linked: The new science of networks. *American journal of Physics*.

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.

Bawden, D. and Robinson, L. (2009). The dark side of information: overload, anxiety and other paradoxes and pathologies. *Journal of Information Science*, 35(2):180–191.

Bell, R., Koren, Y., and Volinsky, C. (2007a). Modeling relationships at multiple scales to improve accuracy of large recommender systems. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, page 95.

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

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

Bell, R. M. and Koren, Y. (2007b). Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 43–52.

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.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

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

Carmel, D., Zwerdling, N., Guy, I., Ofek-Koifman, S., Har'el, N., Ronen, I., Uziel, E., Yogev, S., and Chernov, S. (2009). Personalized social search based on the user's social network. *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*, page 1227.

Cato, J. (2001). *User-centered web design*. Pearson Education.

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. (1999). Combining Content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, number June, pages 60–64. Citeseer.

Davenport, T. and Beck, J. (2001). *The attention economy: Understanding the new currency of business*. Harvard Business Press.

Dietterich, T. (2000). Ensemble methods in machine learning. *Multiple classifier systems*.

Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. (2001). Rank aggregation methods for the Web. *Proceedings of the tenth international conference on World Wide Web - WWW '01*, pages 613–622.

Edmunds, A. and Morris, A. (2000). The problem of information overload in business organisations: a review of the literature. *International Journal of Information Management*, 20(1):17–28.

Eppler, M. and Mengis, J. (2004). The concept of information overload: A review of literature from organization science, accounting, marketing, MIS, and related disciplines. *The Information Society*, 20(5):325–344.

Fischer, G. (2001). User modeling in human–computer interaction. *User modeling and user-adapted interaction*, 11(1):65–86.

Floreano, D. and Mattiussi, C. (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*.

Greenberger, M. (1971). *Computers, communications, and the public interest*. Johns Hopkins University Press.

Halevy, A. and Norvig, P. (2009). The unreasonable effectiveness of data. *Intelligent Systems, IEEE*, 24(2):8–12.

Haveliwala, T. (2003). Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796.

Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

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.

Huang, C., Sun, C., and Lin, H. (2005). Influence of local information on social simulations in small-world network models. *Journal of Artificial Societies and Social Simulation*, 8(4):8.

Huang, Z., Chung, W., Ong, T.-H., and Chen, H. (2002). A graph-based recommender system for digital library. *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries - JCDL '02*, page 65.

Jameson, A. (2009). Adaptive interfaces and agents. *Human-Computer Interaction: Design Issues, Solutions, and Applications*, page 105.

Kirsh, D. (2000). A few thoughts on cognitive overload. *Intellectica*, 1(30):19–51.

Klementiev, A., Roth, D., and Small, K. (2008). A Framework for Unsupervised Rank Aggregation. *Learning to Rank for Information Retrieval*, 51:32.

Kobsa, A. (2001). Generic User Modeling Systems.

Konstas, I., Stathopoulos, V., and Jose, J. (2009). On social networks and collaborative recommendation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 195–202. ACM.

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*.

Lieberman, H. (2009). User interface goals, AI opportunities. *AI Magazine*, 30(2).

Liu, F., Yu, C., and Meng, W. (2002). Personalized web search by mapping user queries to categories. *Proceedings of the eleventh international conference on Information and knowledge management - CIKM '02*, page 558.

Liu, Y., Liu, T., Qin, T., Ma, Z., and Li, H. (2007). Supervised rank aggregation. In *Proceedings of the 16th international conference on World Wide Web*, pages 481–490, New York, New York, USA. ACM.

Manning, C., Raghavan, P., and Schutze, H. (2008). *Introduction to information retrieval*. Camebridge University Press.

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

Newman, M. and Moore, C. (2000). Mean-field solution of the small-world network model. *Physical Review Letters*.

Noll, M. and Meinel, C. (2007). Web search personalization via social bookmarking and tagging. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, pages 367–380. Springer-Verlag.

Norman, D. (1988). The design of everyday things. *Basic Book Inc./New York*.

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

Polikar, R. (2006). Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45.

Qiu, F. and Cho, J. (2006). Automatic identification of user interest for personalized search. *Proceedings of the 15th international conference on World Wide Web - WWW '06*, page 727.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens : An Open Architecture for Collaborative Filtering of Netnews.

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

Robertson, S. (2010). The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Russell, S., Norvig, P., Canny, J., Malik, J., and Edwards, D. (1995). *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, NJ.

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

Segaran, T. (2007). Programming collective intelligence.

Sergey, B. and Lawrence, P. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

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.

Smyth, B. (2007). Case-Based Recommendation.

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.

Sugiyama, K., Hatano, K., and Yoshikawa, M. (2004). Adaptive web search based on user profile constructed without any effort from users. *Proceedings of the 13th conference on World Wide Web - WWW '04*, page 675.

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., Adar, E., Jones, R., and Potts, M. (2007). Information re-retrieval: repeat queries in Yahoo's logs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 151–158. ACM.

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.

Totterdell, P. and Rautenbach, P. (1990). *Adaption as a Problem of Design*, pages 59—-84.

Ujjin, S. and Bentley, P. J. (2002). Learning user preferences using evolution.

Walter, F., Battiston, S., and Schweitzer, F. (2008). A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems*, 16(1):57–74.

Xu, S., Bao, S., Fei, B., Su, Z., and Yu, Y. (2008). Exploring folksonomy for personalized search. *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '08*, page 155.