

ALKINDI MOVIE RECOMMENDATION ENGINE

ALGORITHM SPECIFICATION III:

RATING MANAGER

Eugene Stern
Alkindi, Inc.

May, 2001

CONFIDENTIAL

1 Introduction

Alkindi's recommendation engine is based on an approach to recommending known as *collaborative filtering*. Collaborative filtering simulates system *users* collaborating to recommend *products* to each other, using each other's opinions to filter out all but the most relevant content. In the case of the movie recommendation engine, the products are movies, or videos, which are identified using a unique correspondence between product identifications and titles.

More specifically, Alkindi's engine works by:

1. **Input Step:** Building up an individual taste profile, consisting of *evaluations of products*, for each user;
2. **Clustering Step:** For each user, identifying other users with similar taste profiles;
3. **Output Step:** Recommending to each user products that users similar to the user have tended to enjoy.

1.1 Role of This Document

This document specifies the Input Step — the routines Alkindi's Rating Manager uses in selecting products to ask users to rate. This may seem very broad, since any time a product is shown to user, in any context — say, as a recommended movie, or as a search result — the user can rate it or indicate that they haven't seen it. What we explain here is how to select the products displayed to the user *specifically so that the user can rate them*. In the current implementation of the movie recommendation engine, this covers the two screens of movies that the user rates before becoming a member and getting recommendations, as well as subsequent instances when the user asks to rate more movies ("Improve my Alkindex").

Since one of the key purposes of the Input Step is to collect data for assigning the user to appropriate clusters, this document assumes the machinery developed in “Algorithm Specification I: Clustering Manager.”

This is an Algorithm Specification document; it specifies *what* quantities need to be computed by a component of Alkindi’s recommendation engine (in this case, by the Rating Manager), but does not specify *how* those quantities are to be computed from a software design standpoint. For the sake of concreteness and specificity, this document describes a number of computational steps as subroutines, with particular inputs and outputs. The intention is only to specify underlying algorithmic ideas in an understandable way, and in common language, and not to insist on a particular software design, submodules, or interfaces between the modules. The actual design of the software is described in a separate document (the Software Requirements Specification).

1.2 Tunable Parameters

There are many fixed parameters that play a role in Alkindi’s recommendation engine, which can be tuned in attempt to improve performance. (For example, the number of days that pass from the time a product was recommended to the time that product can be recommended again could be one such parameter.) We call these *tunable parameters*.

Tunable parameters are pointed out in this document whenever they appear. All the tunable parameters are compiled in a separate document, which also lists current values for the parameters. Since this document contains parameters for all the modules of the engine (clustering, recommending, etc.), the notation identifies both the module and the name of the parameter in the specification of that module. For example, a recommending-related parameter named *A* in this document would be named **SEL_A** in the Tunable Parameters document. (The Rating Manager is alternately known as the Selection Manager.)

Some parameters associated with the engine as a whole, but not specifically with the rating manager, also appear in this document. These parameters are referred to here by the names assigned to them in the Tunable Parameters document. (For example, a general, engine-related parameter called **ENG_SELmin** in the list of tunable parameters is referred to by the same name in this document.)

1.3 Assumptions and Data Used

We assume that the products that may be recommended are grouped into *product clusters*. For each product cluster, the Clustering Engine specified in “Algorithm Specification I: Clustering Engine” partitions the users who rated core products in that product cluster into user clusters. (The user clusters are constructed so that the members of a single user cluster rate products in the corresponding product cluster similarly.) A single product can belong to more than one product cluster, and a user belongs to a user cluster corresponding to each product cluster that contains at least one core product that the user has rated. (Definitions, and more details, can be found in the algorithm specification of the Clustering Manager.)

The clustering of the user base occurs offline, and is followed by a process of calculating and storing in the database a broad range of statistics associated with the clusters. This takes place offline as well, and allows the Rating Manager to simply look up in the database many

statistics that would be difficult to calculate in real time. The calculation and storage of these statistics are documented separately. The database tables and their columns are described in an Excel spreadsheet called `pkgs_deps.xls`. The relationships between the tables are documented in an Entity Relationship Diagram stored in the file `Alkindi_ERD.jpg`.

The Rating Manager frequently uses stochastic selection, in the same way as the Recommendation Manager does. The details of how such a choice is made are given in the algorithm specification of the Recommendation Manager (see section 2.2: “Making a Stochastic Choice”).

1.4 Rating Data and Presenting Products for Rating

1.4.1 Kinds of Ratings

Users have an option to rate any product presented to them. If they’ve seen or purchased the product (for example, seen a movie), they can rate it on a rating scale, for example with an integer from 1 to 6. (Details on rating scales can be found in the algorithm specification of the Clustering Manager.) For products not purchased, users have an option to indicate their interest in the product, by clicking one of the options “Interested,” “Not Interested,” and “Don’t Know.”

1.4.2 Presenting Products for Rating

Throughout this document, there are numerous references to “products that have been shown to the user,” or “products that the user has been asked to rate.” Of the products the user has been asked to rate, we distinguish between those that the user has rated, and those the user has not rated. “Rating” a product means evaluating it on a rating scale (e.g., 1-6) rather than indicating interest. This allows us to interpret rating, or failing to rate, a product as data telling us whether or not the user has purchased or seen that product.

In more detail, anytime a product is presented specifically for rating (in the Alkindi movie recommendation engine, this happens when a user joins the service, or clicks on “Rate More Movies, a subcategory of “Improve My Alkindex”), and the user has submitted a page which includes that product, it counts as a product that the user has been asked to rate. If a user submits an evaluation on our (1-6) rating scale, the user has rated the product. If the user submits an indication of interest, or nothing at all, the user has failed to rate the product.

The user can also view product titles in other contexts; for example, as the results of a search or a call to “Get Recommendations.” In this case, the products are not being shown specifically for the purpose of being rated, although the user can rate them or give interest feedback. In this case, we count the product as having been presented for rating if the user has evaluated it on our rating scale or given interest feedback, but not if the user has submitted no data at all. Ratings and interest feedback are interpreted as above (rated/not rated), and in the case of no feedback, since we don’t count the product as having been presented for rating, we don’t keep explicit data on whether or not the user has rated it. (Thus, for example, if a user fails to rate a product that has been recommended, we do not take this as an indication that the user has necessarily not seen/purchased that product.)

2 Selecting a Product to Present for Rating: Outline

2.1 The Basic Idea: Getting to Know Users and Products

Consider a single *rating event* — a single user rating a single product. Each rating event gives us additional data on the user and additional data on the product. If the user has rated few products but the product has been rated by many users, the rating event should be viewed primarily as giving us data on the user. If the product has been rated by few users, but the user has rated (comparatively) many products, the rating event should be viewed primarily as giving us data on the movie.

The first idea of the selection algorithm is to distinguish between the times when we are trying to get data on a user and the times when we are trying to get data on a product. Evidently, with new users, we will always be focusing on getting data on the user. As we collect more data on a user, we are more likely to want that user’s ratings to give us more data on our products. This means that any time we select a product to present for rating, we should begin by stochastically choosing between two different selection strategies: getting to know the user, or getting to know the product.

In detail, this choice is made as follows. Let A be the user’s Alkindex (specified in “Algorithm Specification IV: The Alkindex”). Let S_1 and K_1 be two tunable parameters. If the user has been shown fewer than $S_1 - K_1 \cdot A$ products (“shown” means “had presented for rating,” as explained in section 1.4.2), we automatically use the “get to know the user” selection strategy. Otherwise, we use the “get to know a product” strategy with probability $w_m = a + (b - a) \cdot A$, and the “get to know the user” strategy with probability $w_u = 1 - w_m$. (What these strategies consist of will be explained below.) Here:

- a and b are tunable parameters, with $0 \leq a < b \leq 1$.
- A is (again) the Alkindex.

Thus, w_m ranges between a and b , and increases toward b as the user’s Alkindex increases.

2.2 Getting to Know New vs. Existing Products

The second idea of the selection algorithm is that there are two reasons why a product could have insufficient data (causing us to want to collect more data on it). One reason is that a product might be new to our database (for example, a movie that has just been released on video). Another reason is that a product is fairly obscure; i.e., relatively few people have seen or purchased it and are thus capable of rating it.

If we have chosen the “get to know a product” strategy, our next step is to decide whether that product should be a *new* product or an *existing*, meaning non-new, product. (This document does not specify what causes a product to be classified as new.) One idea is that casual users are more likely to be able to rate new products. Another is that users who tend to rate atypically many of the products that they are shown are good candidates to be asked to rate our more obscure products — i.e., existing products.

In more detail, let S_2 and K_2 be two more tunable parameters, with $S_2 \geq S_1$ and $K_2 \geq K_1$. If the user has been shown fewer than $S_2 - K_2 \cdot A$ products, where A is again the Alkindex,

and there exist new products that have not yet been shown to the user, we automatically use the new product selection strategy. If the user has already been shown all new products, we automatically use the existing product selection strategy. (The intuition is that there are many fewer new products than existing products. Again, the strategies themselves will be explained below.) Otherwise, we use the existing product selection strategy with probability $w_e = J - (J - I) \exp(-ZE')$ and the new product selection strategy with probability $w_n = 1 - w_e$. Here:

- I and J are tunable parameters, with $0 \leq I < J \leq 1$.
- $Z > 0$ is a tunable parameter.
- $E' = \epsilon_{\text{user}} - \epsilon_{\text{all users}}$. Here ϵ_{user} is the user's fraction of products rated to products shown to rate, and $\epsilon_{\text{all users}}$ is the same ratio computed over all users.

Thus, w_e ranges between I and J , and increases toward J as the user's fraction of products rated increases.

2.3 Boundary Conditions

It is possible that some of the selection strategies will fail to come up with a product to rate. If either the “get to know the user” or “get to know a new product” strategy fails to come up with a product, we continue the attempt to select a product to rate by returning to the beginning and eliminating the strategy that failed from further consideration. (Thus, in the former case, we continue by automatically using the “get to know a product” strategy. In the latter case, if we pass again to the “get to know a product” strategy, it will automatically be an existing product.)

The underlying motivation is that getting to know existing products is the most robust strategy, because it chooses from the most underlying products. (The reasons why this is so will become clear below.) Still, even this strategy restricts the selection candidates to a subset of all the possible products. (This is called *restricted mode*.) If the “get to know an existing product” strategy ever fails to return a product, we remove any product restrictions, and let the strategy run in *unrestricted mode*, in which more products are candidates for selection. If a selection still cannot be found, we terminate the search for a product to rate, returning an indication that none could be found.

2.4 Extension: Compiling a List of Products to Rate

Typically, we select products to rate several at a time; the Alkindi movie recommendation engine returns eight at once. This is done by repeatedly calling the function that gets a single selection, until a list containing the desired number of selections is compiled. If one of the strategy restrictions described above goes into effect at any time during the compilation of the list, it stays in effect for all subsequent selections, until the entire list is completed. If the search for a product to rate fails to return a product, the entire list compilation process terminates as well, returning the products that are on the list so far and an indication that the list is incomplete.

2.5 Structure of the Rest of this Document

The rest of this document serves to give details on the three selection strategies mentioned above: getting to know the user, getting to know a new product, and getting to know an existing product.

3 Getting to Know the User

The structure of this strategy is very similar to that of the process used by the recommendation manager to pick a recommended product. Very broadly, we follow the following steps:

- Pick product cluster from which to select product.
- Identify products that can be selected in the chosen product cluster.
- Select product whose rating varies most among user clusters.

We consider each of these steps in turn.

3.1 Selecting a Product Cluster

This is done by assigning weights to each possible product cluster and choosing a particular cluster according to the weights. It is very close what the recommendation manager does to select a product cluster to recommend from. The only difference is in the way we assign the weights on the basis of which a product cluster is chosen.

Let S_i be the i -th product cluster. The weight associated to S_i is

$$w_i = n_i \cdot \exp(A \cdot R_i + B \cdot E_i - C \cdot F_i - K \cdot D_i). \quad (1)$$

Let us explain the terms in this formula.

- n_i is the number of products in the i -th movie cluster. (Making the weights proportional to n_i ensures that clusters with more products to recommend from get more priority in data collecting.) This number is computed as in the algorithm specification for the Recommendation Manager. Thus, if a product belongs to more than one product cluster, we spread out its contribution among all the clusters it belongs to. In more detail, let the i -th product cluster contain N products P_1, P_2, \dots, P_N . Let $m_{ij} = 1/c_j$, where c_j represents the number of product clusters the product P_j belongs to. Then we set $n_i = m_{i1} + m_{i2} + \dots + m_{iN}$.
- R_i and E_i are measures of the user's tendency to like and rate products in the i -th product cluster. Let ε_i represent the fraction of products in the i -th cluster that the user has rated (out of those the user has been asked to rate), and let ε_{ave} represent the overall fraction of products the user has rated (out of those asked). Then we let $E_i = \varepsilon_i - \varepsilon_{\text{ave}}$. Similarly, we let r_i represent the user's average rating in the i -th product cluster, let r_{ave} represent the overall average rating, and set $R_i = r_i - r_{\text{ave}}$.

- F_i is the fraction of products in the i -th cluster that we've already asked the user about. (Here each product is counted once for each product cluster it appears in, regardless of how of those there might be.) Evidently, the more products in a cluster we've already asked about, the less likely we should be (all other factors being equal) to ask about more products in that cluster. Here "products asked about" and "total number of products" in the product cluster refer only to core products, those products which are used in clustering users. For details on these, see the algorithm specification for the Clustering Manager, section 2.2 ("Core Products").
- D_i is a measure of a user's well-clusteredness with respect to the i -th product cluster. We set $D_i = 0$ unless F_i has reached a certain threshold F_{\min} ; this threshold is a tunable parameter. If $F_i \geq F_{\min}$, we set $D_i = 1 - b_i$. Here, b_i is a quantity defined in the algorithm specification for the Alkindex. It measures the fraction of bad recommendations the user is expected to get from the i -th product cluster. Thus, $1 - b_i$ measures the fraction of good recommendations the user is expected to get; it increases as the user is clustered better.
- A , B , and C are positive tunable parameters. K is a negative tunable parameter (product clusters in which the user is already well clustered are weighed less).

(1) determines a weight for each product cluster. Once these weights are computed, we pick a product cluster stochastically according to the weights.

3.2 Identifying Candidate Products for Selection

Once the product cluster is chosen, in looking for a product to present for rating, we restrict our attention to that cluster. To begin with, *we only consider core products — products that are used in clustering users with respect to that product cluster.* Next, *we throw out the products the user has already been asked about.*

The next, more subtle step is to restrict to products that have been rated by a high fraction of the people asked about them. The idea is that for new users, we insist that the products we offer for rating have been seen by a high percentage of the users asked about them; as we get more data on the user, we relax this requirement.

In more detail, this requirement is defined in terms of thresholds $T_0, T_1, T_2, \dots, T_k$ and cutoffs $\epsilon_0, \epsilon_1, \epsilon_2, \dots, \epsilon_k$ associated with each. The T_i and ϵ_i are tunable parameters such that $0 = T_0 \leq T_1 \leq T_2 \leq \dots \leq 1$ and $1 \geq \epsilon_0 \geq \epsilon_1 \geq \epsilon_2 \geq \dots$. Then, when the user has been shown no core products in the product cluster, in selecting products to be offered for rating, we restrict to those that have been rated by at least ϵ_0 of the users asked about them. Once the user has been shown T_1 (interpreted as a percentage) of the core products in the product cluster, we expand to considering products that have been rated by at least ϵ_1 (again, interpreted as a percentage) of the users asked about them. Once the user has been shown T_2 of the core products in the product cluster, we allow ourselves to consider products that have been rated by at least ϵ_2 of the users asked about them, and so on.

If the user is at the i -th threshold (i.e., has been shown T_i , but not T_{i+1} , of all the core products in the product cluster), but there are no suitable products that have been rated by at least ϵ_i of the users asked about them, we relax to considering products rated by at least

ϵ_{i+1} of the users asked. If there are still no suitable products, we relax the condition to ϵ_{i+2} , and so on, until some suitable products are found. If there are no suitable products in the product cluster even when the ϵ -restrictions are removed, that product cluster is removed from further consideration in getting to know the user.

3.3 Selecting a Product to Present for Rating

Having chosen a product cluster and restricted our universe of products in that cluster to core products that have been rated by enough users, the last step is to select the product in that universe whose rating has the highest variance among the user clusters with respect to that product cluster.

Let C_1, C_2, \dots, C_n be the user clusters with respect to the product cluster of interest. Let P_1, P_2, \dots, P_N be the products in the product cluster that are candidates to be presented for rating to our particular user.

Suppose first that the user has not yet rated any products in the product cluster. Let r_{ij} be the average rating of product P_i in user cluster C_j . We let V_i represent the variance of $r_{i1}, r_{i2}, \dots, r_{in}$. (If \bar{r}_i represents the mean of $r_{i1}, r_{i2}, \dots, r_{in}$, we define V_i to be $n/(n-1)$ times the mean of $(r_{i1} - \bar{r}_i)^2, (r_{i2} - \bar{r}_i)^2, \dots, (r_{in} - \bar{r}_i)^2$.) Then we return the product P_i whose variance is highest. This is what we mean by selecting the product whose average rating varies the most among the user clusters.

Finding the product in the product cluster whose rating varies the most among all user clusters makes sense if we think the user is equally likely to end up in any of these clusters. Once the user has rated some products in the product cluster, we are better informed about which user cluster the user may end up in, and we incorporate this information in the selection algorithm. That is, instead of computing V_i by including all user clusters, we consider only those clusters which are closest to the user. (Here, “closest” has the following meaning: once we have some rating data on the user, the user has some representation in “ratings space,” and we can measure distance from the user to the center of each user cluster. See the algorithm specification for the Clustering Manager for a detailed explanation of how users are represented in ratings space, and how distance there is measured.)

Naturally, the more rating data we have on the user, the better our idea of where in ratings space the user will ultimately turn out to lie. Thus, the more products a user has rated, the more we can restrict the clusters C_j over which we compute the i -th product’s variance V_i .

We implement this in a cascade manner, analogous to the preceding subsection. Namely, we introduce two sets of tunable parameters F_i and m_i , where $0 = F_0 \leq F_1 \leq F_2 \leq \dots \leq F_k \leq 1$ and $1 = m_0 \geq m_1 \geq m_2 \geq \dots \geq m_k > 0$. Once the user has rated F_i (viewed as a percentage) of the core products in the i -th product cluster, we compute the variance of each product over the m_i user clusters (out of all user clusters, so that the m_i are also interpreted as percentages) whose centers are closest to the user.

Once we have identified the user clusters over which to compute the variance, we compute it for the products which are candidates for selection, and return the product with the highest variance.

4 Getting to Know a New Product

4.1 Selecting a Product Cluster

The “get to know a new product” strategy also begins by choosing a product cluster stochastically. Since we are collecting product data, we are only interested in maximizing the chance that a user is capable of rating products in the product cluster we end up choosing. Thus the weight assigned to the i -th product cluster is

$$w'_i = n'_i \cdot \exp(B' \cdot E'_i). \quad (2)$$

Here:

- B' is a positive tunable parameter.
- n'_i is analogous to n_i in section 3.1, except that instead of counting core products in a given movie cluster, we count the core movies. We also exclude any products that the user has been shown already from the count.
- E'_i is like E_i in section 3.1, only it is adjusted per products rather than per user. Recall that ε_i is the fraction of products in the i -th product cluster that the user has rated (out of those the user has been shown). Similarly, we define $\varepsilon_{i,\text{all}}$ to be the fraction of products rated to products shown computed over all users. We set

$$E'_i = \varepsilon_i - \varepsilon_{i,\text{all}}. \quad (3)$$

We choose a product cluster stochastically according to the weights w'_i . Once a product cluster has been chosen, it remains to choose a new product from that cluster.

4.2 Choosing Between Popular and Unpopular New Products

We divide the products that are candidates for selection (new products in the chosen product cluster that have not yet been shown to the user) into two classes: *popular* and *unpopular* products. Recalling that for a given product, we write ϵ for the fraction of people who’ve rated it (out of those asked), we define a product to be popular if its ϵ value exceeds a threshold P , unpopular otherwise. (P is a tunable parameter.) A product that has been shown to fewer than Q users (Q is another tunable parameter) is an exception to this: it is automatically classified as popular. (Intuitively, products don’t get classified as unpopular until we have enough data on them to be able to tell.)

We now set a pair of weights u_1, u_2 which we’ll use to choose between selecting from popular and unpopular products.

Let u_2 be the weight assigned to unpopular products. If there are no unpopular products, $u_2 = 0$. Otherwise, u_2 will range between two tunable parameters U and V with $0 \leq U \leq V \leq 1$. We introduce another tunable parameter $W > 0$ and set $u_2 = V - (V - U)e^{-W \cdot E'_i}$ if $E'_i > 0$, $u_2 = U$ otherwise. (Here, E'_i is as defined in section 4.1, and i is the index of the movie cluster under consideration. u_2 increases from U toward V as E'_i increases from 0.)

We set $u_1 = 1 - u_2$ if there are any popular new products, $u_1 = 0$ otherwise. We choose between the group of popular new products and the group of unpopular new products stochastically, using the weights u_1 and u_2 .

4.3 Selecting a Product to Present for Rating

Finally, whether we’ve chosen popular or unpopular products, we select a product at random from the chosen group, weighing equally all products in the group (and in the chosen product cluster) that the user has not been shown.

5 Getting to Know an Existing Product

5.1 Counting Ratings and Restricted Mode

The “get to know an existing product” strategy is intended to collect ratings for non-new products. One difference in the treatment of new and existing products is the following. With new products, we are interested in getting as many ratings for them as possible while they are new. (The idea is that they’re only new for a finite amount of time, so there’s an upper limit on how many new ratings we can get for them using the “get to know a new product” strategy.) However, existing products have an unbounded lifespan in the system. Clearly, there will be some existing products for which we’ll (eventually) have enough data, and no longer need to collect more.

To express this precisely, we introduce a tunable parameter $\Omega \leq 1$, and stipulate that the “get to know an existing product” strategy ordinarily operates in *restricted mode*, in which it selects only from products which:

- Are not new;
- Have been rated by fewer than Ω (interpreted as a percentage) of our users.

If we are unable to find a single eligible product in restricted mode, the strategy shifts into unrestricted mode, in which these two restrictions no longer apply.

5.2 Selecting a Product Cluster

In broad outline, getting to know an existing product works the same way as the other strategies. We begin by choosing a product cluster stochastically, weighing the i -th product cluster by

$$w_i'' = n_i'' \cdot \exp(B'' \cdot E_i'). \quad (4)$$

Here:

- E_i' has the same meaning as in equation (2).
- n_i'' is the number of non-new products in the i -th product cluster that have been rated by fewer than Ω of all users and not yet been shown to the user. (Again, products in a given cluster are counted so that a product’s membership is spread out among the product clusters it belongs to.)
- $B'' > 0$ is a tunable parameter.

5.3 Choosing Between Popular and Unpopular Products

Once a product cluster is chosen, we choose between popular and unpopular products in that product cluster in the same fashion as with new products. Again, we define popularity in terms of tunable parameters P' and Q' , which are analogs of P and Q in the previous section. (A product is popular if its ϵ -ratio exceeds P' , or else if fewer than Q' users have been asked to rate it.) The weight assigned to unpopular products is $u'_2 = V' - (V' - U')e^{-W' \cdot E'_i}$ if $E'_i > 0$, $u'_2 = U'$ otherwise. (E'_i is as defined by 2, and U', V', W' are tunable parameters. The weight assigned to popular products is $u'_1 = 1 - u'_2$. We choose to look within either the popular or the unpopular products stochastically, according to the weights u'_1, u'_2 .)

5.4 Products with Almost Enough Ratings

One extra step we take with existing products is to give some preference to products that have almost, but not quite, reached a certain threshold number of ratings. Let Γ and Δ be two tunable parameters, with $\Gamma \leq \Delta \leq \Omega$. Recalling that n_i is the total number of eligible products in our cluster, we let p_i be the number of eligible products in the cluster (in the restricted sense) that have been rated by between Γ and Δ (considered as percentages) of our user base. Let κ be one more tunable parameter. With probability $\kappa \cdot \frac{p_i}{n_i}$, we restrict our universe of eligible products — either popular or unpopular products in product cluster i that have been rated by fewer than Ω of all users and have not yet been shown to the user — to those that have been rated by between Γ and Δ of all users ratings. (With probability $1 - \kappa \cdot \frac{p_i}{n_i}$, we keep the universe of eligible products as is.)

5.5 Selecting a Product to Present for Rating

Having made this final restriction on the universe of eligible products, we now choose a product at random from this universe and return that product.