# The Value of Bad Recommendations – Injecting Orthogonality in Product Recommendations and Learning Over Time

by


## Luis Blando

BS, MS in Systems Engineering, Catholic University of Córdoba, 1992
MS in Computer Science, University of Nevada, 1994
MS in Computer Science, Northeastern University, 1998


Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

**Master of Science in Engineering and Business Management**

at the

**Massachusetts Institute of Technology**

**May 2001**



**Signature of Author** _____

System Design and Management Program
May, 2001

**Certified by** _____

Dan Ariely – Asst. Professor
Sloan School of Management
Thesis Supervisor

**Accepted by** _____

Paul A. Lagace
LFM/SDM Co-Director
Professor of Aeronautics & Astronautics and Engineering Systems

**Accepted by** _____

Stephen C. Graves
LFM/SDM Co-Director
Abraham Siegel Professor of Management

# THE VALUE OF BAD RECOMMENDATIONS – INJECTING ORTHOGONALITY IN PRODUCT RECOMMENDATIONS AND LEARNING OVER TIME

by Luis Blando

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

**Master of Science in Engineering and Management**

ABSTRACT

Perhaps the most exciting aspect of electronic commerce is the potential ability to learn the preferences of individual customers and make tailored recommendations to them. When making such recommendations the recommendation system is facing a fundamental dilemma. Should it provide the best recommendation based on its current state of understanding of the customer or should it try to learn more about the customer in order to provide higher potential payoffs in the future? This dilemma is at the heart of the current work.

The dilemma facing a recommendation system is presented conceptually, and an approach for ideal learning is proposed and tested. In order to test our hypothesis, we modified one commercially available recommendation engine to consider measures of novelty in an initial learning phase. We analyzed results from the normal and modified engine for different datasets and characteristics of customers.

*to my late mom, as promised*

Table of Contents

# LIST OF FIGURES

# ACKNOWLEDGMENTS

The author wishes to thank the management of GTE Laboratories/Verizon, particularly Tony Confrey, for their financial backing in this project; the people at Saffron Technology, particularly Roger Barney, for their continued technical support; Andre Turner for his help in running the experiments; and the several current and former students that provided their rankings as experimental data.

The author wants to express his gratitude to his advisors, Dan Ariely and Manny Aparicio, whose constant encouragement and sometimes-crazy ideas kept this project moving. Working with them has been a pleasure, as well as a mental roller-coaster.

Finally, and most importantly, I want to thank Laura, my best friend of many years. Even though having endured this work-and-study neglect several times in the past, she still stood strong and unconditionally supported me for one more run.

*Chapter 1*

# INFORMATION OVERLOAD AND PERSONALIZATION

With the advent of the Internet, and the explosion of popularity of the world wide web, new ways of interfacing with the customer are possible. Whereas before the communication was usually one way and feedback from the user was delayed or indirect (e.g. television or print advertisement) nowadays the Internet allows for a much more interactive experience.

In addition, web sites have been growing at a tremendous pace in the past years. Available information, therefore, has exploded and has quickly become unmanageable for the average person. Personalization technology is an enabler to attempt to cope with this information overload problem.

Not only are we now able to immediately receive feedback from the user by, for example, measuring the time spent on our website; but also we can pro-actively personalize what we present on the screen. This ability to target the individual as opposed to a market segment is called “one-to-one web marketing”.

The underlying hypothesis under one-to-one web marketing is that users will benefit more from the web experience, thus leading to greater attachment to the website owner and possibly to increased receptiveness to up-sell and cross-sell opportunities and thus in increased revenue.

## The Technology Behind Personalization

There are a number of mechanisms to decide what to present to the user at each of the pages in the website. These technologies span the range from deciding

which pages to present next to the user, what colors to show the text in, to which products or advertisements to present on the page based on the system's knowledge of the user. Generically speaking, this "tailoring" of a website to a user is termed "personalization".

As it will be presented in the next chapter, personalizing a website implies a number of activities. Underlying most of these activities, however, lies the question: "How do we select among X, Y, or Z given what we know about the user?" One obvious example of this is Amazon.com, where users are recommended products. However, the answer to the above question can (and should) drive personalization in a much wider range, such as which banner ads to present, what colors to use, and other non explicitly-requested items.

At the heart of non-trivial personalization there is thus the problem of "product recommendation", where product can range from actual products, to screen configurations and other recommendations. This thesis work concerns itself with the technology behind product recommendation, and suggests an alternative way to improve its performance.

*Chapter 2*

# THE MANY MEANINGS OF WEB PERSONALIZATION

The concept of personalizing content has become very popular nowadays. However, there's substantial confusion as to what personalization means. Some web sites claim to have personalized content because they print the name or nickname of the user when the user logs in. Some web sites "personalize" by making recommendations based on the purchases of other users. Other web sites actually recommend products based on the individual's preferences.

Ironically, even the most popular "personalized" sites are not really tailored to each individual, but rather to a market segment. This section will explore the different approaches to personalization from a high-level perspective, and present the pros and cons of each.

## Customization

The first approach that we will consider is one we have termed "customization". Solutions employing this approach usually ask the user for specific information (i.e. name, age, zip code, preferences in movies, etc) which is then used to tailor the site's content to the user.

These systems, although very useful, cannot really be considered intelligent, as they do not attempt to infer what "else" the user might like, given what they know the user already likes. In a nutshell, these systems do not "learn" from their interactions with the user.

This approach to personalization is a first-step and has become very popular. For example, Figure 1 shows a snapshot of a customized page for my.netscape.com. Note the circled sections and the information they contain. Since the user told the system that he lives in Boston and has relatives in Argentina, the site lists the weather in both Boston and a couple of **user-selected** cities in Argentina. In addition, the user is also provided with movie listings and stock quotes that he has requested.

It is important to realize that this type of personalization is "static". In other words, even though the actual content will change (i.e. updated weather, new movie releases, and updated stock prices), the categories that are shown to the user (city weather, stock quotes, movie reviews) remain fixed. That is, there is no learning involved. The system will not present the user with content beyond the specific content that the user pre-selected. For example, the system will not present the user with:

- Information about a recent earthquake in Argentina of devastating consequences. In this case, even though this information is clearly relevant to the user, it does not fit the "weather-from-Argentina" category.

- The release of the "Evita" soundtrack CD. In this case, even though the user probably liked the movie, and it deals with Argentina, this CD is not in the "movies" category and thus will not be shown.

This type of system is useful when the content can be separated into clear and crisp categories that the user can configure. Configuration is performed when the system presents content categories to the user (weather, movies, stocks) and the user provides information that uniquely (and deterministically) identifies the content to be presented. For instance, the user would enter 02144 for the ZIP

code, the AMC-Boston movie theater for the movies section, and {VZ, PMIX} for the stock quotes. Once this has been configured, the content will "refresh" the elements, but will not change (add/remove) categories.

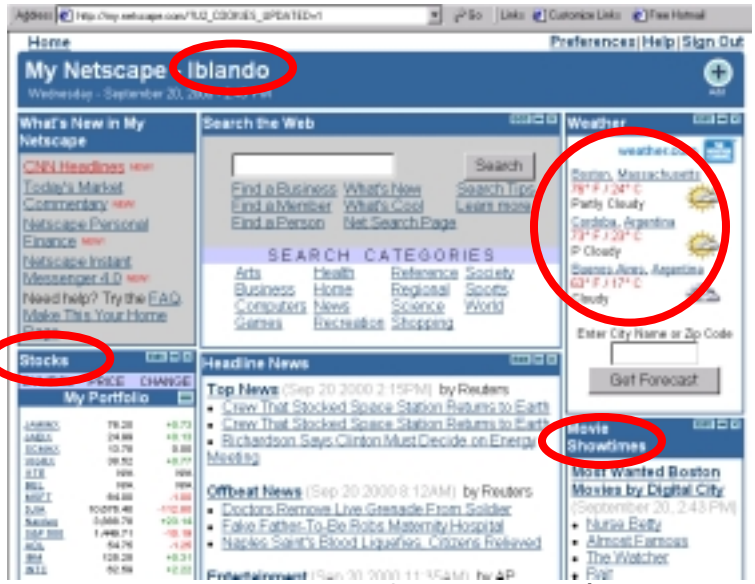

Figure 1: My Netscape - Customization example

**Rule-Based Systems**

Whenever personalization can be fully expressed as a function of known data in the form of *if-then* rules, a rule-based approach is advisable. For a rule-based system to be a viable solution, the "knowledge engineer" needs to have a sufficiently complete understanding of the domain and all the possibilities that may arise. It is the job of the "knowledge engineer" to document this "knowledge" into the form of hard-coded rules.

It can be very difficult to use a rule-based solution for recommending products because the domain can be very large with extensive possibilities. However, rule based engines can be very effective for both narrowing a recommendation search

and for fine-tuning a result set. For example, many shopping sites use rules, similar to the rules below, to enhance the customer experience:

```
IF {PROD1,PROD2} in recommendation set
  IF deliveryTime(PROD1) < deliveryTime(PROD2) THEN
    RECOMMEND PROD1
  ELSE
    RECOMMEND PROD2
```

Rule based systems are not very adaptable (with the exception of the parameters that the rules might be provided with). Rule-based systems are applicable when we know the domain and are not interested in learning.

Many toolkits are available to create inference systems from rule-based technology. Figure 2 shows a snapshot from the promotional materials of one such toolkit. Note how the engine recommends a second on-sale item based on the fact that the user has selected an on-sale item. Behind the scenes, the engine is using a rule such as

```
IF SELECT-ONSALE THEN RECOMMEND ONSALE
```

One factor that should be of concern is that, as Figure 2 boasts, the engine would recommend a second on-sale item even though the user has never actually purchased an on-sale item (the user only browsed an on-sale item). What if the user was an individual that really does not care for on-sale items, but simply found an item of interest that just happened to be on sale? He would be shown other on-sale items! This type of inflexibility is characteristic of rule-based systems.

At the core of this problem is the fact that rule-based systems usually do not allow us to really get at the reason why customers make the decisions they make.

In a sense, rule-based systems are dependent on very few product attributes and rules are built on top of those. These relatively few attributes might not be enough to represent the user's complex decision functions. Hence rule-based system's lack of predictive accuracy. For example, the "onsale" attribute used in the rule above might not accurately represent the reason why the user decided to buy the product in the first place. As a matter of fact, maybe the user bought the product because of its color without paying any importance to whether the product was on sale or not. However, since this rule only looks at one coarse-level attribute ("onsale"), it will make the incorrect inference of showing the user other on-sale items.

In essence, rule-based systems are useful for specifying universal truths or facts, as they make little (or no) attempt at customizing their responses for a given user. In addition, rule-based systems are inherently error-prone; the above "on-sale" rule might recommend a lower-priced item than an item currently in a user's shopping cart!

Of course, one can always start adding rules to try to generate different outcomes for different types of users. Those attempting this approach would quickly find themselves with an unwieldy number of rules to maintain.

Rule-based systems are very good for universal facts that are to be enforced, or applied, to the entire universe of possibilities. They can easily express filter criteria or can be used for the initial "knowledge" in an inference engine when there's little collected behavior to draw upon.

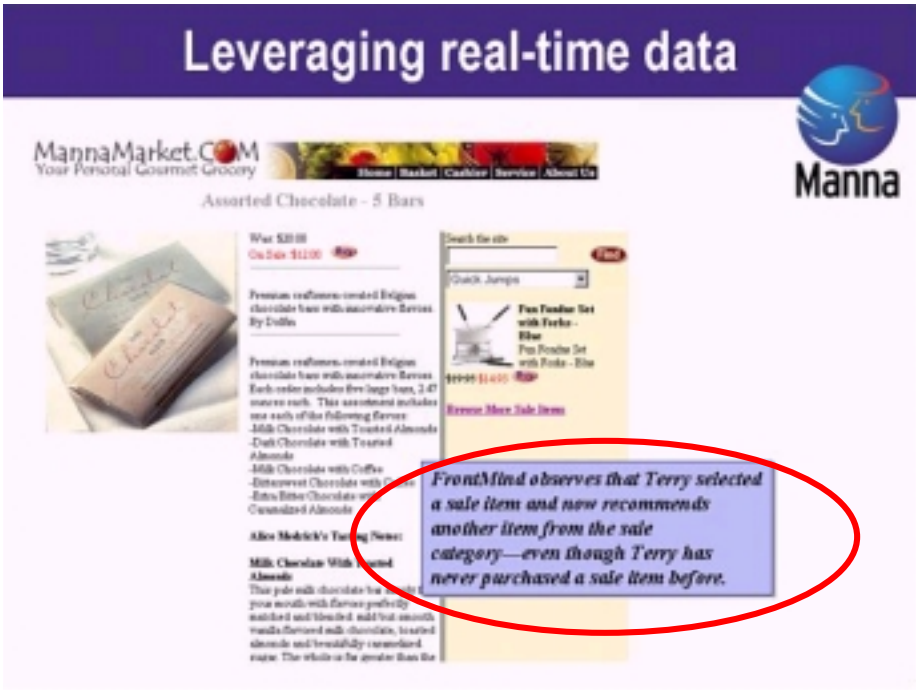


Figure 2: Rule-based system

One drawback of rule-based filtering is the well-established fact that consumers do not always know what they like. Since the rules are written on the assumption that the knowledge engineer knows what the users will like, they are inherently error-prone. Rules can also be reverse-engineered from the study of past interaction data, though this is useful for possibly fine-tuning certain rule parameters and not necessarily for discovering new attributes previously unknown.

Furthermore, since rule-based algorithms are usually at a loss when trying to explain why they have reached a particular recommendation, a consumer will always remain dependent on the agent to make "magical" recommendations for him (and potentially perpetuate a bad recommendation pattern). This shortcoming, generally speaking, is related to any method that uses self explicated preferences (explicit statements of preferences such as "safety is the most important attribute").

**Self-Learning Systems - Collaborative Filtering**

Collaborative filtering uses a community-based approach. In a nutshell, collaborative filtering is about discovering what people "like yourself" have liked before, and recommending those items. Instead of trying to determine the intrinsic characteristics of the product that would be desirable for a particular user, collaborative filtering tries to place the user into a "segment" and recommends the products that, historically, were liked by other members of the segment. This has been likened to computerizing the "word-of-mouth" effect.

Operationally, collaborative filtering predicts a person's preferences as a linear, weighted combination of other people's preferences. The canonical collaborative filtering approach is shown in Figure 3.

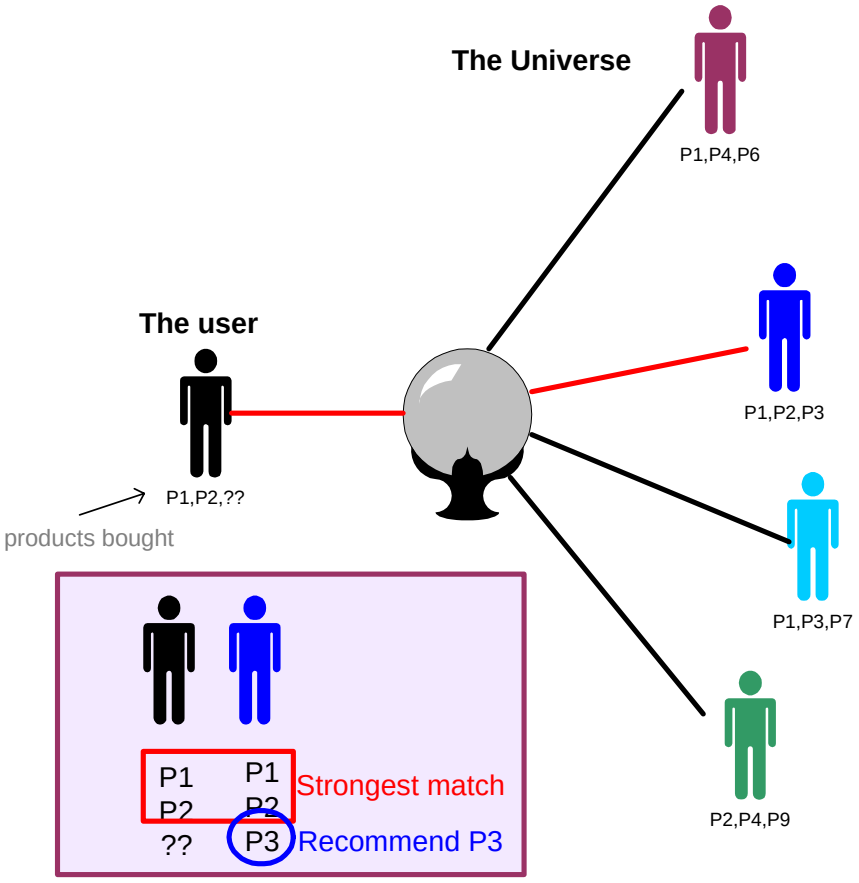


Figure 3: Canonical collaborative filtering

"The user", let's call him Bill, has purchased products P1 and P2 and wants the inference engine to recommend other products. The collaborative filtering engine tries to match the user with one of the entries in its universe of users. After exhaustively checking prior purchases of other available users, the engine determines that the blue user is the one that most closely resembles Bill. Since the blue user has also purchased P3, it recommends this product to Bill. Note that although we used "products-purchased" as the specific characteristic to classify users in this example, other characteristics, such as demographic profiles, can and have been used.

One of the problems of canonical collaborative filtering is that as the universe of users becomes large, the computational time required to allocate a user to a similar kind grows. In a nutshell, it doesn't scale.

To alleviate this performance problem, collaborative filtering toolkit vendors have taken the approach of creating like-minded segments in order to reduce the universe space. Therefore, when a new user is requesting a recommendation he or she is allocated to a "segment" and provided with the recommendations for that segment.

There's a range of approaches used by collaborative filtering solutions that differ in how the user is "allocated" to a segment. These different implementations have implications on two fronts:

(1) Efficiency – some methods aim at comparing every user with every other user to determine proximity (i.e. nearest neighbor approaches). Since this approach is very intensive, optimization approaches have been attempted. These optimizations create clusters of users and then allocate a target user based on key statistics for the cluster. The goal is to reduce the order of magnitude of comparisons from O(N) to O(C) where N is the number of users, C is the number of clusters, and C<<<N.

(2) Accuracy – by segmenting users in clusters, the intra-cluster variance affects the performance of the results. For borderline target users, (those that barely make it into a cluster), the potential error in assigning the cluster's recommendations to the target user might be nontrivial.

This “segmented” approach has its drawbacks. First, and foremost, a user might be on the “boundaries” of a segment and therefore anything we recommend will not be exact. Second, since we are generating segments using an averaging process, individual user information is being lost. Third, the segmentation optimization for the collaborative filtering approach is implicitly built on the assumption that the population is pseudo homogenous and that such segments can be reasonably defined. If the population is very heterogeneous, and crisp segments cannot be identified, the within-segment error in recommendations will be large. Finally, another concern with the segmentation approaches is that it usually works best when the subjects are relatively static over time. For users that change their preference patterns frequently, re-segmentation is necessary, which increases the resources required for the system.

Another drawback of collaborative filtering (in all of its forms) is that it is not completely self-sufficient. Since it relies on historical data to make recommendations, how can it ever recommend a brand-new item? Artificial mechanisms need to be employed so that systems built around collaborative filtering can be jump-started with new items. This exercise, however, is error-prone.

A very popular site that uses collaborative filtering for its recommendations is Amazon.com.

As shown in Figure 4, upon entering the site and selecting the recommendations option the user is presented with a set of recommended items for purchase. In the particular example of Figure 4, a book about fly-fishing in Vermont is recommended.

Figure 4: Amazon.com collaborative filtering-based recommendations

Note the prominent **Get Better Recommendations** message, with the "rate the selections" link in the bottom left corner of Figure 4. Rating past purchases or recommended items is a method of obtaining feedback from the customer about past recommendations. By rating the selection, the user may move himself out of his current target segment and into a new one, with potentially different recommendations.

Note that Figure 4 does not tell us why the Vermont fly-fishing book has been recommended. However, with a little bit of work we can find out a possible reason. Figure 5 shows the complete entry for the recommended book.

Circled in red is a book the user had bought in the past. Note that there's a list of books that "similar customers" bought in conjunction with the recommended book. This is why Amazon is recommending the Vermont book. However,

notice that in doing so the engine has neglected the fact that this particular user lives in Massachusetts, does not own any other book about Vermont, and likes some variety in the topics he reads.

Figure 5: Understanding why in Amazon.com...

**Attribute-Based Models (a.k.a. Individual Models)**

An alternative to the collaborative filtering approach is to construct attribute-based inferences. In these models, both products and consumers are given different attributes that are meant to represent the inherent qualities of the products (or preferences). Products are selected based on the user's specific preferences.

Figure 6 shows how selection (inference) works. Given a universe of attributes, which are commonly defined and shared in the system, each user has what we call an "attribute signature", which is basically her particular preference for each attribute. In the example, this particular user has a loading of 0.3, 0.6, and 0.2 for attributes one, two, and three, respectively. There are only three attributes in this system.

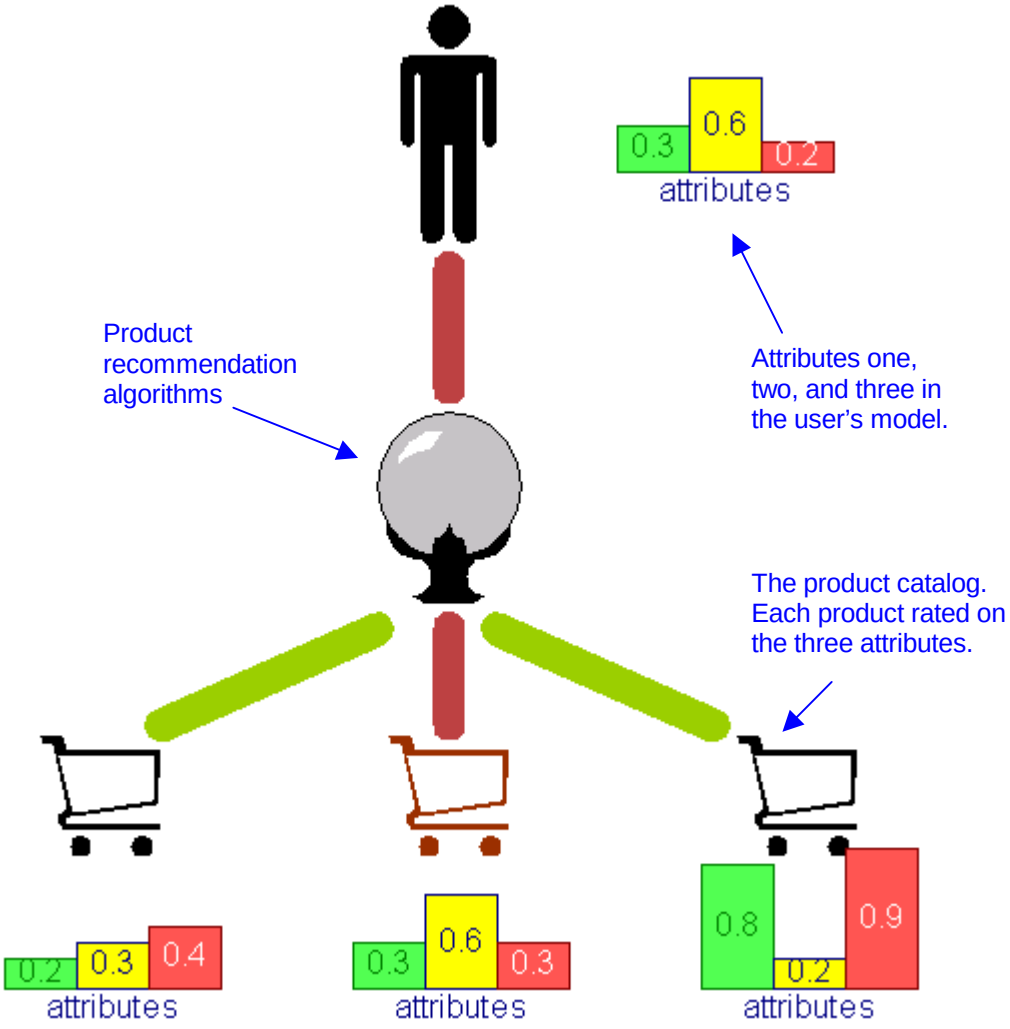


Figure 6: Attribute-based selection example

Each product is also rated along these different attribute "dimensions". Selecting that product whose attribute signature is "closest" to the user's attribute signature is the job of the recommendation engine. Note that the center product (in red) has an attribute signature that is almost identical to the user's.

Our example is an admittedly simplified scenario, as there can be many more than three attributes and it's not necessary that all products be rated on all attributes or that an user have a signature[1] that contains all attributes. Still, it is representative of the general idea.

In essence, individual-based models delve deeper in the "constituents" of a product and are more concerned with the "reason" why a user likes or dislikes a particular product. This makes individual-based models more flexible and, usually, more accurate than other prediction mechanisms. In addition, individual-based models can immediately recommend newly added products, as long as these products have been appropriately "attributed". This means that they do not suffer from the "cold start" problem that plagues collaborative filtering solutions.

Both collaborative-filtering and individual-based approaches need information about the individual user. In the case of collaborative-filtering, the information needed takes the form of a series of population ratings on existing products. In the case of individual-based approaches, the information takes the form of a set of attribute weights. For collaborative-filtering, however, it is also necessary to know (and consider) the information for the rest of the users. Pure individual-based models need not consider this. Of course, hybrid mechanisms are usually the most powerful as they can tailor products on an individual basis and generate leads on a community basis.

[1] A collection of ratings on each of the attributes known to the system is called a "signature".

An important factor to consider is that new products and changing user preferences are relatively simple problems for individual-based models but complex for collaborative-filtering approaches.

To date, individual-model solutions have been less common than collaborative filtering approaches, possibly due to the fact that coming up with the right attributes and diligently tagging content is an arduous and expensive task. Selecting the right attributes to use to correctly categorize a product catalog is a difficult undertaking, which runs the risk of misspecification. For example, a product catalog for computers might never add "case color" as one of the attributes, for traditionally computers have been only about power, speed, and capacity. Yet, as the artistic design of the newer models of personal computers have proven, the exterior's color is important for a segment of the PC-buying population (which now includes people that might have not cared for a computer in the past).

In addition, individual-model solutions are harder, computationally, and more difficult to represent because the matrix of attributes can become very large and thus can be difficult to maintain and process. This computational complexity has prevented truly individual-based approaches from reaching the mainstream. It is only recently that advances have been made to optimize these computations and make these approaches more feasible.

**Which approach to take?**

Given the different possibilities for performing customization, a sensible question to ask is which approach is applicable for which situation. This section will explore this issue.

We need to consider the following parameters when selecting a personalization approach:

- The size of the decision space, i.e. the number of decisions that need to be made; and
- The expert's knowledge of the decision domain.
- The characteristics of the user population

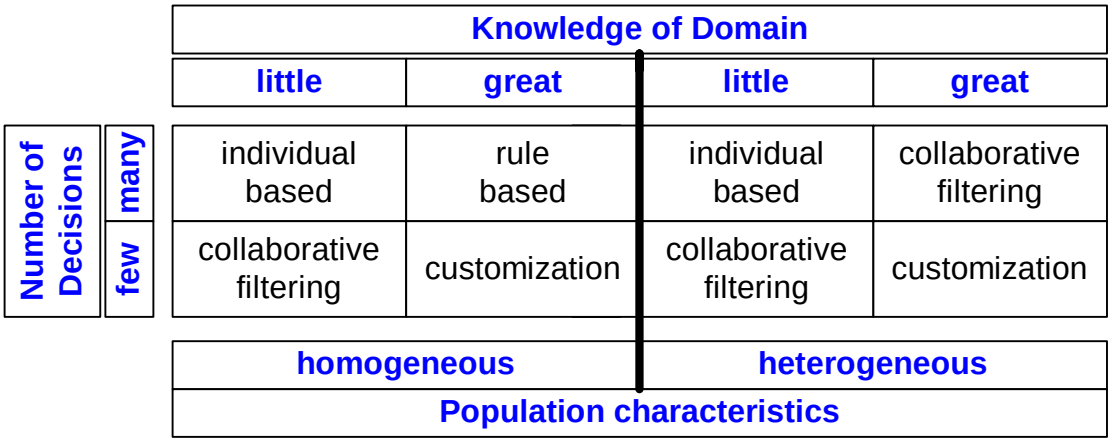


Figure 7: How to select personalization methods

Figure 7 shows which customization to apply depending on these three factors[2].

*Simple Customization*

Simple customization is appropriate when the decision space is small and we have a great deal of knowledge about the domain. .

[2] Note that this heuristic is approximate.

*Rule-Based Customization*

Rule-based customization is appropriate when the decision space is large and we have a great deal of knowledge about the domain. If we are extremely proficient in the domain of personalization, we can completely specify all the possible patterns in personalization that might appear and can define the exact rules that will yield the best possible content for the user based on past experience.

This approach is usually used by "Expert Systems". These systems, popular in years past, usually have a large mass of highly developed rules regarding a particular domain. These rules usually take years to build by a team of true world experts on the subject. It is important to note that the domains for which these systems are built usually have no (or few) surprises and therefore most of the answers can be predicted given a set of preconditions. Expert systems in the area of medical diagnosis, petroleum exploration, and computer system configuration have all been built with reasonable success.

The domain of one-to-one web marketing personalization efforts, however, is far from being known. Trying to estimate the tastes and desires of users is an incredible difficult task, and one where there are hardly any universally valid rules (or for that matter, completely deterministic rules).

*Learning System*

Learning systems are appropriate when the decision space is large and we have little knowledge about the domain.

The two learning systems that we consider are collaborative filtering and individual-based systems. To decide which type of learning system is best, we need to evaluate the target users and the content universe.

Individual-based models embrace the difference in people and make no (or little) assumptions about the similarity between users. They recognize the inherent characteristics of content (i.e. which attributes it contains and which ones it doesn't) and of users (which attributes they like and which ones they don't). Using this information they provide the best product for each user by computing the distances between the content's characteristics and the user's preferences.

People, and children in particular, have changing and varying tastes. This means that each user's preference profile will change frequently. In such a setting, collaborative-based filtering has been shown to consistently under-perform individual-based models.

While people have changing preferences, they can also behave in herd-manner. Trends become in-vogue, styles become fashionable, and the latest action figure is all you see on television (and all you **want** to see on television!). In this case, many users are "joined" in their likes/dislikes and thus the collaborative filtering approach can shine.

**Focus of this work**

Within the different alternatives for personalization, individual-based models are the most promising and potentially most accurate. This work will focus on the recommendation engine component of these methods.

*Chapter 3*

# SIMILARITY AND NOVELTY

As explained before, this work will concentrate on the recommendation engine part of personalization systems. This chapter will describe the general idea behind individual-based model recommendation systems and its shortcomings.

## Similarity Ratings

Given a set of products, each of them rated on a collection of attributes, plus a profile of attribute-ratings preferences for a user, the problem of recommendation can be reduced to finding a subset of products that best match the user's preference profile. In essence, the problem is reduced to searching the product space for all those products that are "similar" to the ideal product.

For example, the user's ideal product in Figure 6 has ratings of 0.3 for the green attribute, 0.6 for yellow, and 0.2 for pink. Subsequently, out of the universe of three products, the recommendation engine finds the one that is most similar to this ideal product. Measuring this similarity can be done via a number of mathematical measures, and recommendation can follow from these measurements. For example, recommend the product whose geometric distance to the ideal product is shortest.

## Shortcomings – The "Myopic Agent" effect

Using similarity alone for recommendation agents has some drawbacks. Let's take the case of a brand-new user. In this case, the agent knows absolutely nothing

about this user. It is therefore hard-pressed to make any valuable recommendations, and it has roughly a 50/50 chance of recommending a product that makes sense. If the user didn't like the product, a somewhat different product will be presented until the user likes the selection. Once a successful selection has taken place, however, the "similarity policy" will dominate and the agent will tend to recommend products that are similar to the one that it knows the user likes.

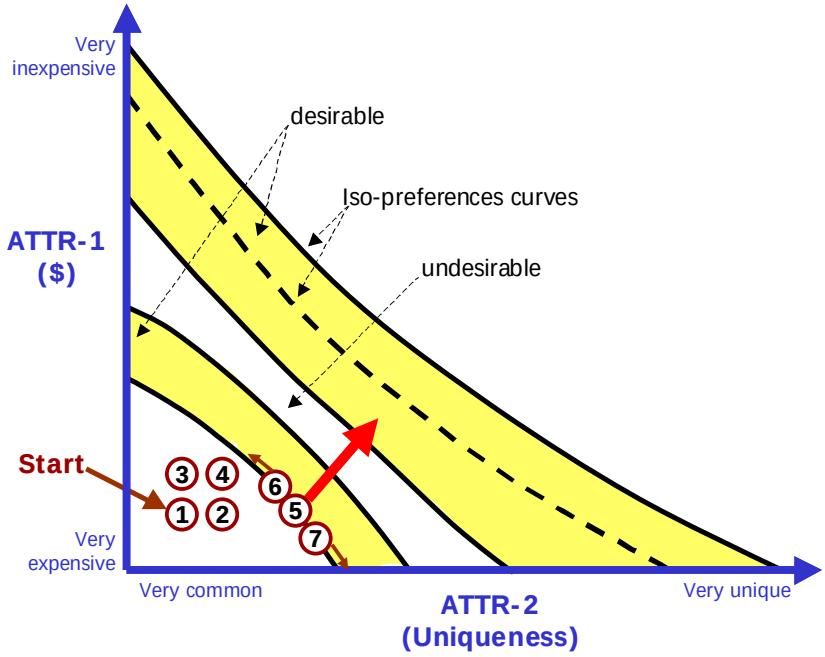


Figure 8: Myopic agent effect

Figure 8 shows a diagram to illustrate this problem. It shows an admittedly oversimplified product universe space that has been rated on only two attributes (cost and uniqueness). Each product in the catalog from which the agent chooses can be placed in this plane. Furthermore, there are groups of products that share the same characteristics. Let's imagine for a moment that the products are rare coins and that our user is a coin collector. We can draw the product space and

define the iso-preferences[3] curves for this particular coin collector. In this case, this collector does not care for middle-of-the-road coins, but rather prefers either very unique (even if expensive) or very common (only if inexpensive) coins. After drawing the isopreferences curves, as shown in Figure 8, we see that there exists a "band" of undesirable space in between two bands of desirable products.

If we were using a similarity-based agent to recommend products from this catalog to this collector, we might experience what we have called the "myopic agent effect". The first product recommended happened to lay, out of random luck, in one of the undesirable groups (1). The agent then attempts to "get away" from those products and lands on yet another undesirable group (2). This sequence continues until finally the agent finds a product that the user actually likes (5). At that moment, however, that agent is prone to continue to recommend products from that group (iso-preference curve) until it exhausts all of them, at which time it will be search time again.

Unfortunately, during all this time the agent failed to recognize that there might be other areas of the feature space (such as the one shown with a red arrow in Figure 8) that the user might also enjoy.

The problem evidenced before is commonplace with locally optimum approaches such as the one shown. Product recommendation engines that narrow mindedly zero-in on the feature space region where some products have been liked and ignore the rest of the space are bound to suffer from this problem. If the user does not have any knowledge about the product universe and relies solely on the agent's advise, this approach also results in no learning on the agent's part, as the user cannot really "correct" any of the agent's recommendation.

[3] Any product on the curve is "as desired" as any other product on the curve.

Furthermore, the above discussion has always implicitly assumed that both the marketplace and the user's preferences remain static over time. This is hardly the case in today's Internet economy. Products are introduced constantly and they immediately enlarge the product universe that product recommendation engines need to consider. In addition, user's preferences do indeed change and that poses a new set of problems to the engine as new areas of the feature space might now become desirable.

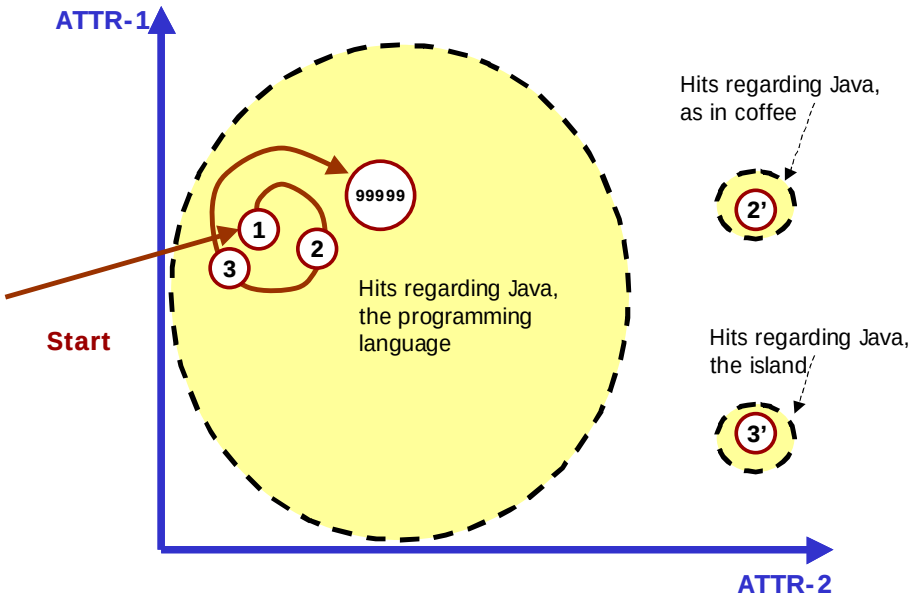


Figure 9: Biased product catalogs

There's yet another situation where similarity-only agents fail to deliver the right set of results to end-users. Consider the situation depicted in Figure 9. In this case, the product "catalog" is disproportionably large for one type of products and very small on other, equally "desirable" products. A typical example of this phenomenon can be found everyday in web-page searches. For instance, a search for the keyword "java" might return a large number of hits for the programming language interpretation of the word, and none (or few) about the beverage or the island interpretation.

Furthermore, if a user happens to "like" a product in the programming language category, further requests for recommendations will undoubtedly revolve within the same homogeneous category. When product catalogs are even and well segmented (i.e. a few products well discriminated by the selected attributes) this problem is important, but not critical, as the user would exhaust one (small) group and only then move to a different desirable group (possibly increasing his satisfaction level with the change). However, when product catalogs are very biased (i.e. a large number of products of one type and a small number of products of different-but-related categories) this problem can be acute, since exhausting the large product group is impractical and the user would in fact never be recommended products from other categories but the one the agent is "sure" the user likes.

This is exemplified in Figure 9, where the agent starts in the "programming language" group and continues to make its recommendations 2, 3, …, 99999 from that same group. Notice that these recommendations will most likely contain redundant information, and this agent will not be able to detect that. Contrast this return set with the case where the agent actively seeks products from different segments, as depicted by the sequence (1), (2'), and (3'). In this case, the user will be presented with a variety of products from distinct product regions, and the agent will in fact learn which region it needs to concentrate its next search on.

*Chapter 4*

# INJECTING ORTHOGONALITY - A HYPOTHESIS

A different approach to recommendation engines is presented, rooted on the concept of exploring the feature space while learning about the user to detect areas of interest and avoid the problems described in previous chapters.

## A hypothesis

Consider for a second what would happen if we were able to device a recommendation agent that would "explore" the entire feature space before committing to any given groups of products. In our example, for instance, this agent would be able to identify the two desirable areas and thus recommend products from both.

We contend that such an approach would, over the long run, outperform a typical similarity-based agent. We have called this special agent an "active learning agent". Such an agent would initially pro-actively explore the feature space by recommending products that are different from products it knows the user likes. In such an environment, the agent would learn about the entire space and later revert to similarity-based recommendations, once the entire product space has been "sampled".

Figure 10 shows a high-level representation of the logic behind the agent's learning process. In essence, when a new user enters the system, the agent will know very little about the environment and thus will on purpose recommend

dissimilar products to the ones it has detected the user likes. Once the agent determines that it knows enough about the user's preferences and the product space, a standard similarity approach can be used.

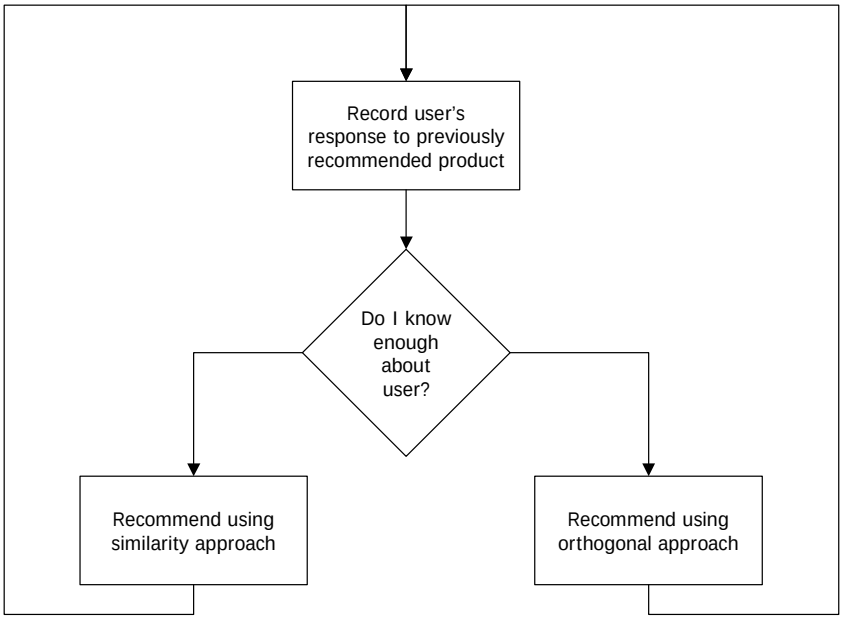


Figure 10: Learning flow with both similarity and orthogonal approaches

We contend that once we get to use the similarity approach, we have learned more about the feature space in relation to the user's preferences and therefore the agent would be able to make better recommendations.

**Some obvious challenges**

The approach presented above has a couple of obvious challenges. The first problem has to do when deciding when to switch from "exploring the space" to "using it". In other words, when do we make the change from recommending dissimilar products to simply using the information that we collect to recommend products that follow that profile?

This work will not attempt to address this issue, except to state that generic statistical metrics can be used to estimate the "confidence" of the agent about the entire product space and use that as well as an appropriately tuned threshold to automatically switch from orthogonality to similarity. We believe that determining how to optimally switch from one mode to another would only make sense once the goodness of the approach is proven, not before. This work attempts to prove the general approach leaves this detail of selecting the appropriate threshold for further research.

In addition, instead of selecting this threshold as a hard and fast rule systems can be designed to provide a continuous function that will measure both the degree of similarity as well as the expected learning from any given product to be recommended. Then, the agent could make complex decisions like deciding to recommend a product that is "only 10% less good than the best one" but will "yield 40% more expected learning than the best one". Furthermore, for agents that can recommend multiple products at a time, a mixed approach can be used which yields the best solution to this dilemma. Namely, some of the recommended products can follow the "similarity" approach while the others can follow the active-learning approach.

On account of the long-term nature of the benefits of this approach, it should be obvious that for short-lived interactions (i.e. less than 5-10 recommendation feedback pairs) are not suitable for this approach. The rationale is simple: if you only have a couple of chances of recommending something to a user, try to do your best as soon as possible, as you won't have time to explore the space and then recommend.

Last, but certainly not least, we need to define what "orthogonal" means. So far we have described this new approach recommendations as "dissimilar",

“different”, or “orthogonal”. We have yet to explain how exactly we compute this orthogonality factor for a product. This issue concerns us next.

*Chapter 5*

# MEASURING SIMILARITY AND NOVELTY

This chapter describes the computation of our version of orthogonality based on the measures that traditional recommendation engines provide.

## A Simple Recommendation Engine

In order to be able to prove or disprove our hypothesis with the fewer number of unknown variables, we decided to build a very simple, straightforward recommendation engine. Commercial-grade recommendation engine products are substantially more sophisticated. Suggestions for future work with one such product are given in a later chapter.

### *Linear distance between two profiles*

Our recommendation engine works on products that the user has liked. For such products, the engine examines the attributes of the products and adjusts the user's "histogram" of values for each attribute, as shown in Figure 6. Figure 11 shows the details of the distance calculation for a given user profile and a particular hypothetical product, on a four-attribute space.

Notice that we are not interested in the relative positions of the user's preferences in the attribute scale, but rather in the absolute difference. Thus, it is assumed that the model under which the user operates is an "ideal point model" where the preferences along an attribute are not monotonic but rather with a sigle peak (more sugar in my coffee is better up to some point at which it becomes worse).

In addition, a simplifying assumption is that deviations from the ideal point in either direction are equally unappealing. Because of the non-monotonicity assumption, this simple engine would not work well in an environment in which preferences are monotonically increasing (when higher levels are always better). In such an environment, this engine will not be able to favor a product whose attributes are higher than the user's (assuming higher is strictly better), yet at distance identical to that of another product, whose attributes are all lower than the user. In the example of Figure 11, the product shown, {**0.5**, 0.2, 0.0, 0.7} with a distance of 1.3, would be identical to a product {**0.9**, 0.2, 0.0, 0.7} also with a distance of 1.3. If the values of attribute **a1** imply higher-is-better, this engine cannot recognize that.

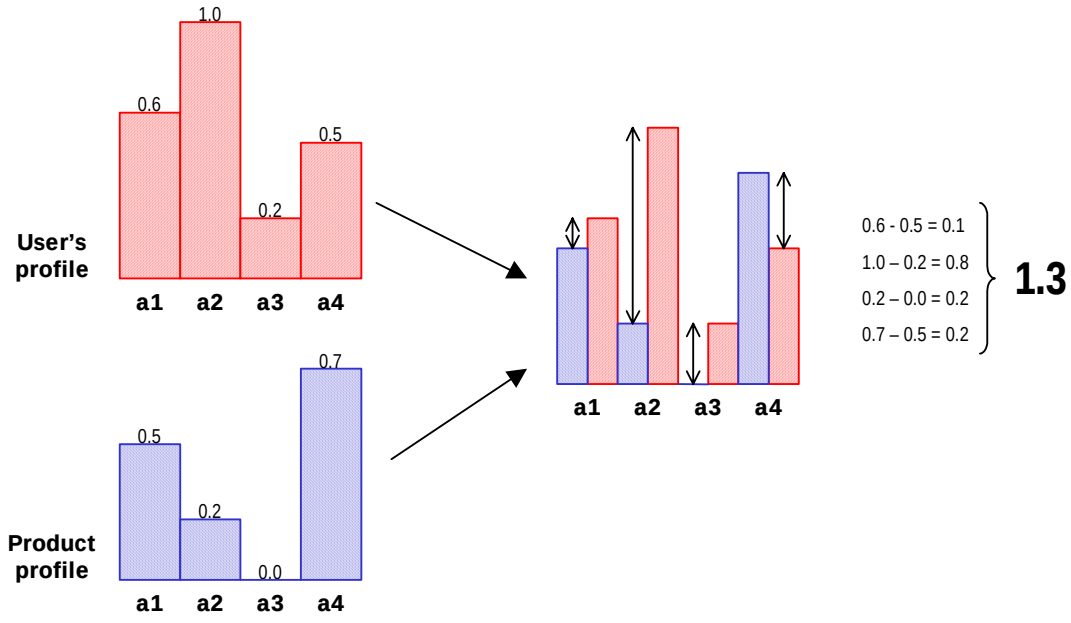


Figure 11: Computing the similarity by profile distance

*Distance-Based similarity (The Myopic Agent)*

At any given time, the user has a certain profile (or desirability) for each particular attribute. Each product contains a specific value for each attribute. Thus, in order to compute the "similarity" or "desirability" for this particular product and user combination, we simply compute the distance between the product's attribute profile and the user's attribute profile.

Once we obtain the desirability for each product, this engine simply selects the product with the highest desirability (that is, the one with the shortest distance, or closest to the user's profile). We have termed this behavior the "myopic" agent.

*Distance-Based Novelty (The Active Agent)*

Computing novelty in this simple engine is straightforward. The same identical computation takes place, yet at the time of recommending a product we select the one with the **least** desirability (that is, the one that is farthest from the user's profile). Note that a further improvement can be achieved if the engine would take into account not only the desirability but also its confidence / knowledge about the attributes in questions.

However, in order to form a recommendation engine we need to understand how to switch from exploring the space (i.e. recommending on novelty as explained above) and recommending on similarity using the information collected in the exploratory phase. We introduced this "switching" value as a parameter and ran our experiment with different values. This agent would therefore recommend the first N products based on novelty, and the rest of the products based on similarity. We have named this agent the "active" agent, or the "learning" agent.

*Chapter 6*

# EXPERIMENTAL SETUP

This chapter describes our experimental setup, and presents the metrics that we will use to measure the results.

## The design of the experiment

The purpose of the experiment was to validate or refute our active learning hypothesis. We therefore created two recommendation agents: the myopic agent, using only similarity as its guide; and the active agent, which first explores the space as described before. Comparing these two would allow us to validate/refute the hypothesis.

To recap, we defined and coded the following agents:

- Myopic agent: this agent uses the standard similarity-based mechanism, as explained in the previous chapter.

- Active agent: embodies our hypothesis, and is calculated as explained in the previous chapter.

In order to test the performance of these engines, we assembled different datasets representing products along with their attributes, as well as users and their ratings for each product. To keep the experiment manageable, we only allowed two possible responses, LIKE and DISLIKE.

The overall flow of the experiment would be, therefore, to loop user by user, recommending products in the order mandated by the agent, until all products had been recommended and all users had been exhausted. We can then measure the hit profile, namely whether an agent recommended a liked or disliked product at each iteration. In the following sections we will describe all these elements in detail.

**The Dataset**

A separate program to exhibit clustering as required by this work generated the dataset. The purpose of this thesis was not to prove/disprove the existence of such clustering in real-people responses and therefore we fixed that variable by generating a dataset that contained six different population classes each with six clusters in them. The dataset also contained 512 products.

Products are defined by 9 binary attributes (i.e., Xi =0,1 for I=1,…,9). There are, therefore, $2^9$=512 possible products. Customers are defined by a vector of betas. Betas are bounded by [-1,1]. "Hybrid customers" are those that can behave according to two different personalities. These customers have two sets of betas and their utility for a given product is the maximum utility from the two sets.

The utilities are defined by the inner product of the beta vector and the product vector. We introduced error terms. The customer will purchase a product if its utility for the product plus an error term is greater than zero. Error terms are normally distributed with mean zero.

*Generating the data:*

The nine attributes are divided in three sets:

A = [1,2,3]

B = [4,5,6]

C = [7,8,9]

We then defined six different clusters of customers, each with its own preference for the attribute sets mentioned before, as follows:

**Cluster N: preferred set > intermediary set > low set**

Cluster 1: A > B > C

Cluster 2: A > C > B

Cluster 3: B > A > C

Cluster 4: B > C > A

Cluster 5: C > A > B

Cluster 6: C > B > A

In each cluster, the Betas for the preferred set are drawn from an uniform distribution, U[1/3, 1], Betas for the intermediary set are drawn from U[-1/3,1/3] and Betas for the low set are drawn from [-1,-1/3].

Hybrid customers have two utility functions, and their actual utility is the maximum of the two functions. The six clusters used in this experiment are the following:

Hybrid Cluster1 = Cluster 1 and Cluster 4

Hybrid Cluster2 = Cluster 2 and Cluster 6

Hybrid Cluster3 = Cluster 3 and Cluster 6

Hybrid Cluster4 = Cluster 4 and Cluster 5

Hybrid Cluster5 = Cluster 5 and Cluster 1

Hybrid Cluster6 = Cluster 6 and Cluster 1

There are twenty customers in each of the six clusters defined above. A randomly distributed error with mean zero and variance 0.4 was added to each of

the utilities before making the choice. Choice was = 1 if the net utility (real utility + noise) was greater than zero. The data with this error term was named "Small Error", or SE subjects.

We also injected a randomly distributed error with mean zero and variance 1.4 to each of the utilities before making the choice. Choice was = 1 if the net utility (real utility + noise) was greater than zero. The data with this error term was named "Large Error", or LE subjects.

Similarly, we introduced small and large errors for the hybrid subjects thus resulting in two more datasets, the Hybrid Small Error (HSE) and the Hybrid Large Error (HLE) subjects.

In summary, the entire product dataset consisted of 512 products, each with 9 attributes. The subject (rankings) data consisted of four different subject types: SE, LE, HSE, and HLE. Each subject type had six clusters and each cluster twenty subjects. In total, we have 480 subjects (4 x 6 x 20).

**The flow of the experiment**

Given an agent type, the experiment consisted in sequentially process each user in the rankings set. After the first product is selected at random, the rest of the products are presented in the order recommended by the agent, for each user. We present 20 products in total to each user. That is, we do not exhaust all the products (512 recommendations).

**Measuring the results**

After each user is completed, a hit profile is calculated for this user/agent combination. A hit profile is simply one row of the complete hit matrix for this agent, as shown in Figure 12.

| | Iteration 1 | Iteration 2 | ... | Iter. 161 |
|---|---|---|---|---|
| User 1 | 0 | 1 | ... | 1 |
| User 2 | 1 | 1 | ... | 0 |
| ... | ... | ... | ... | ... |
| User N | 0 | 0 | ... | 1 |

User 1 disliked our first recommendation, liked the second one,..., and liked the last one.
User 2 liked our first and second recommendations,..., and disliked the last one
User N disliked our first and second recommendations,..., and liked the last one

Figure 12: A hit matrix for one agent

When we are interested in evaluating the results for different types of users, we compute the hit matrix for each user cluster. Given a user cluster, we average all the columns (one per iteration) and we obtain a single vector, the results vector, which represents how well this agent did on this particular "type of user". The value of each element of this ordered vector represent how well the agent did in this particular iteration. Notice that this value is in the range [0…1].

Once we have results vectors for each of the agents/user types combinations, we can compare the results of each agent for a given user type by plotting them on a graph. Generally speaking, we expect the myopic agent to have better performance in the first recommendations but overall to under perform the active agent. We expect the active agent, on the other hand, to start slower than the myopic one, as it is exploring the space, but jump directly from cluster to cluster as required thus providing a better overall performance.

*Chapter 7*

# RESULTS

This chapter presents the results of the experiment. Graphically, results are included for all subject types and clusters.

## Running the Experiment

All the data was ran through a myopic agent (a.k.a "learning 0", or "L0"), an active learning agent with a switching parameter of 5 (a.k.a. "learning 5" or "L5"), and an active learning agent with a switching parameter of 10 (a.k.a. "learning 10" or "L10").

Hit matrices were computed for all the 480 subjects lumped together, for each four subject types separately, and for each six clusters independently within each subject type. Results for L0, L5, and L10 were displayed graphically, and in order to better compare, a logarithmic trend line was created for each.

## All subjects

The comparison for all 480 subjects is shown in Figure 13. As expected, we see the L0 agent outperforming the active agents in the beginning, but both L5 and L10 eventually trend towards a higher accuracy value. It is interesting to note that in this graph, as in many others that follow, the L5 agent outperforms L10. We believe this phenomenon might be due to the fact that since we are only recommending a total of 20 products, the L10 agent spends half of these recommendations "learning" (that is, purposely recommending dissimilar products). The remaining 10 recommendations are not enough to "lift" the trend

high enough (to outperform L5). We will see, however, that there are cases where the extra learning still does prove better even in this short-run case. We believe that had we ran the recommendation experiments longer the L10 would in all cases outperform L5.

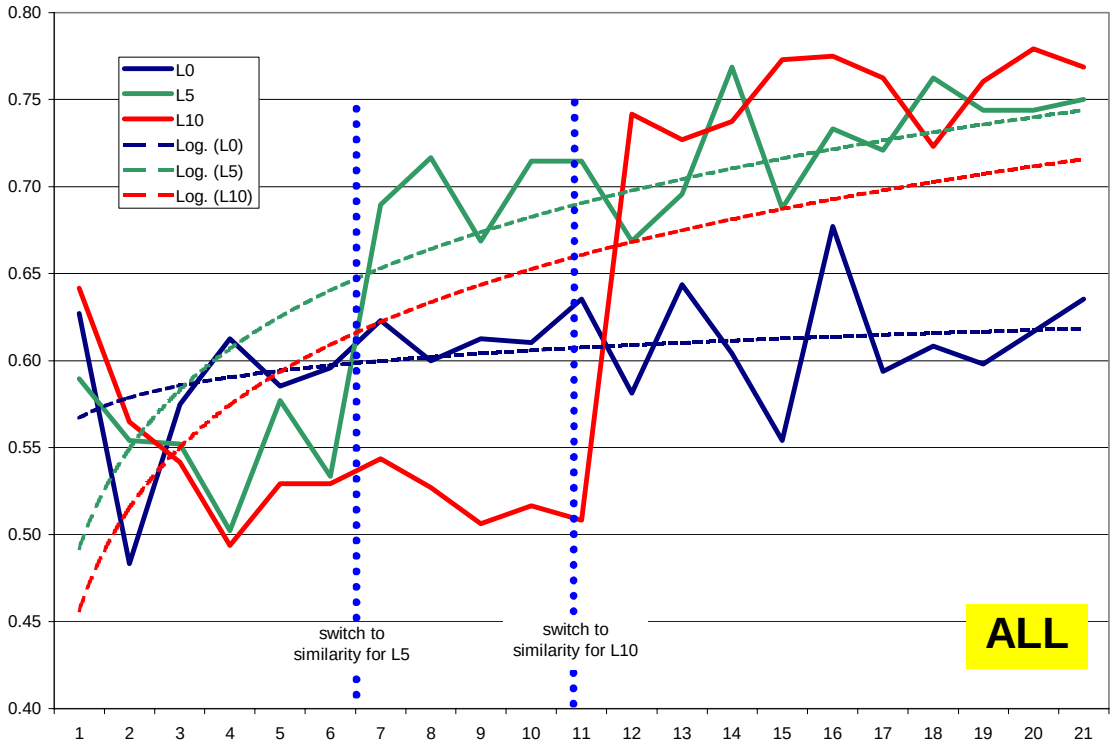


Figure 13: Results - all subjects

**LE subjects**

Figure 14 shows the results for all large error subjects. Notice the large swings in the raw data and the pronounced increase in accuracy when the learning agents change from "exploring" to "recommending", at iterations 5 and 10, respectively.

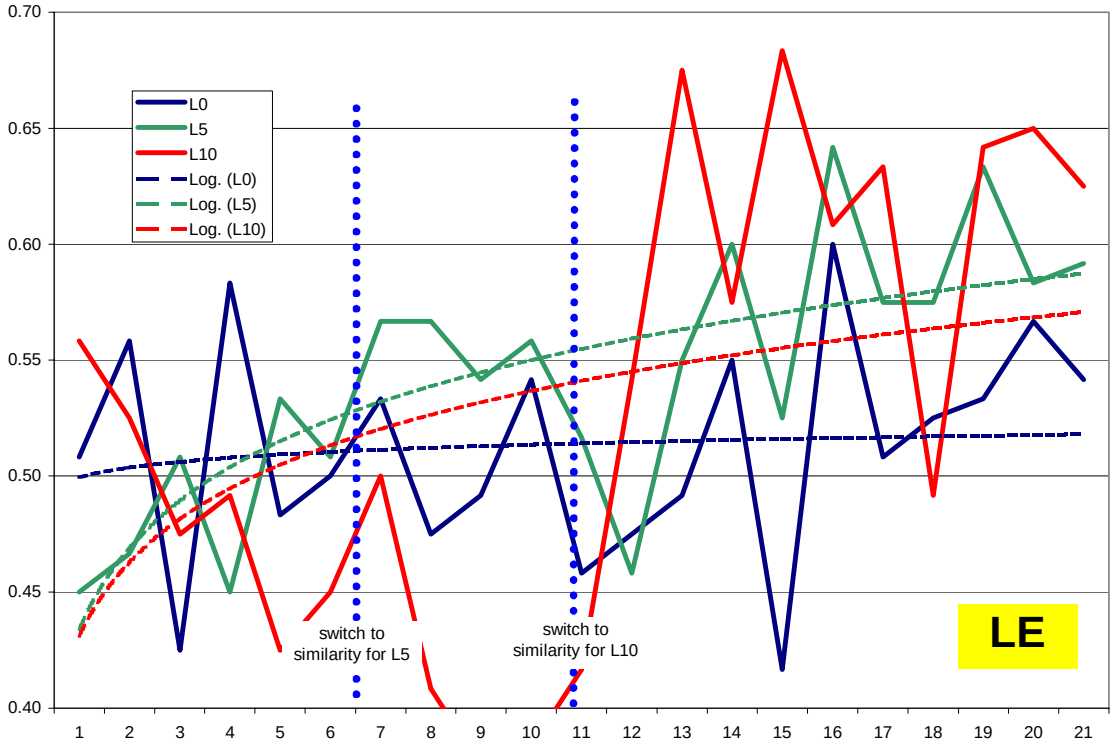


Figure 14: Results - LE subjects

*LE clusters*

Figure 15 shows the results for the six large error clusters. Notice how for cluster LE1 the myopic agent actually outperforms the others. This is the only case where the active agents are not the most accurate. In the case of LE1, it is possible that the first product selected (at random) falls exactly in the cluster where those subjects have a preference, and thus the myopic agent concentrates in recommending similar products, gaining high accuracy marks.

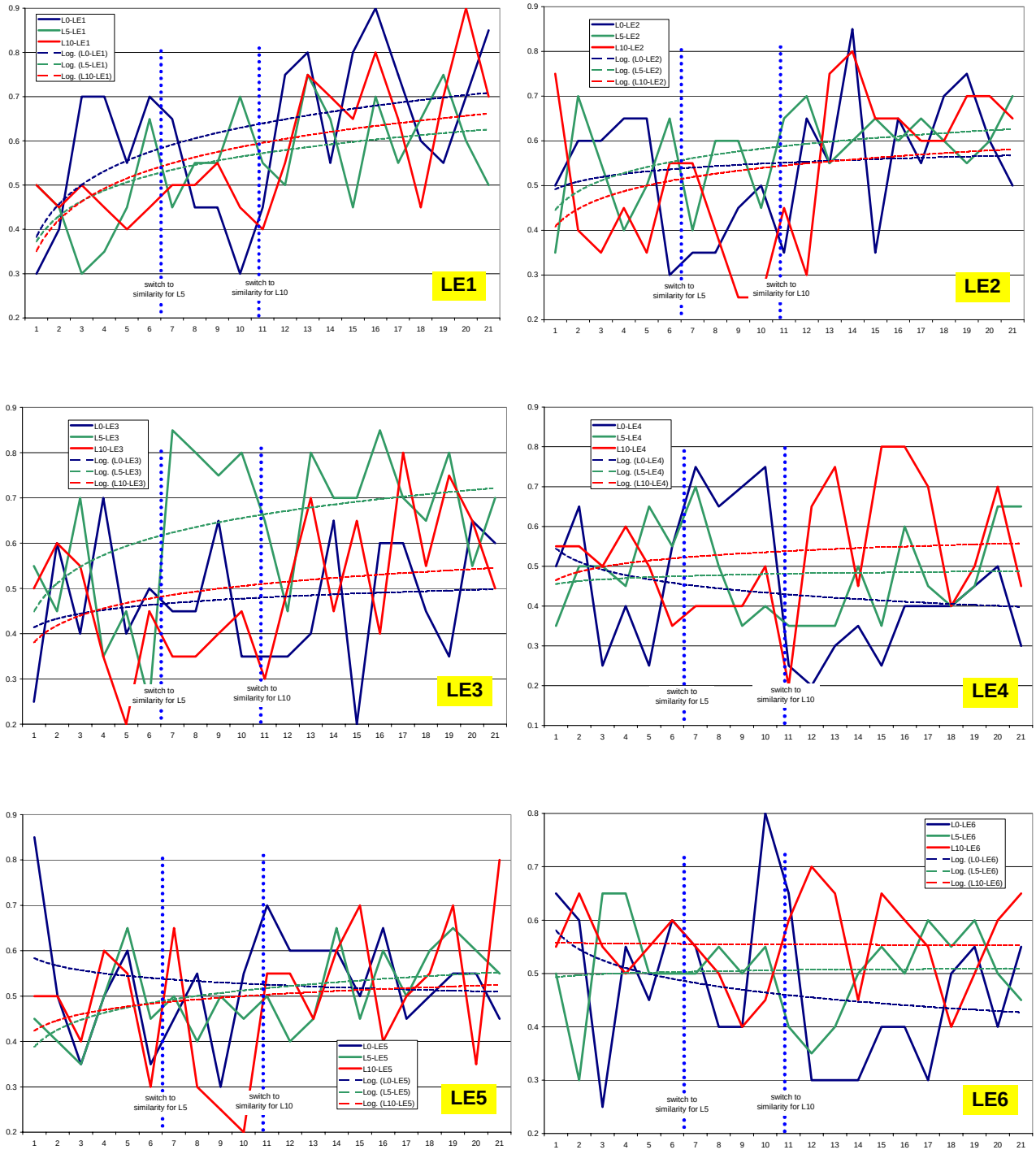


Figure 15: Results - LE clusters

**SE subjects**

As expected, the small error subjects, shown in Figure 16, display much smaller swings in the raw data. It is very apparent in the graph where the learning agents make the switch to recommending on the learned information.

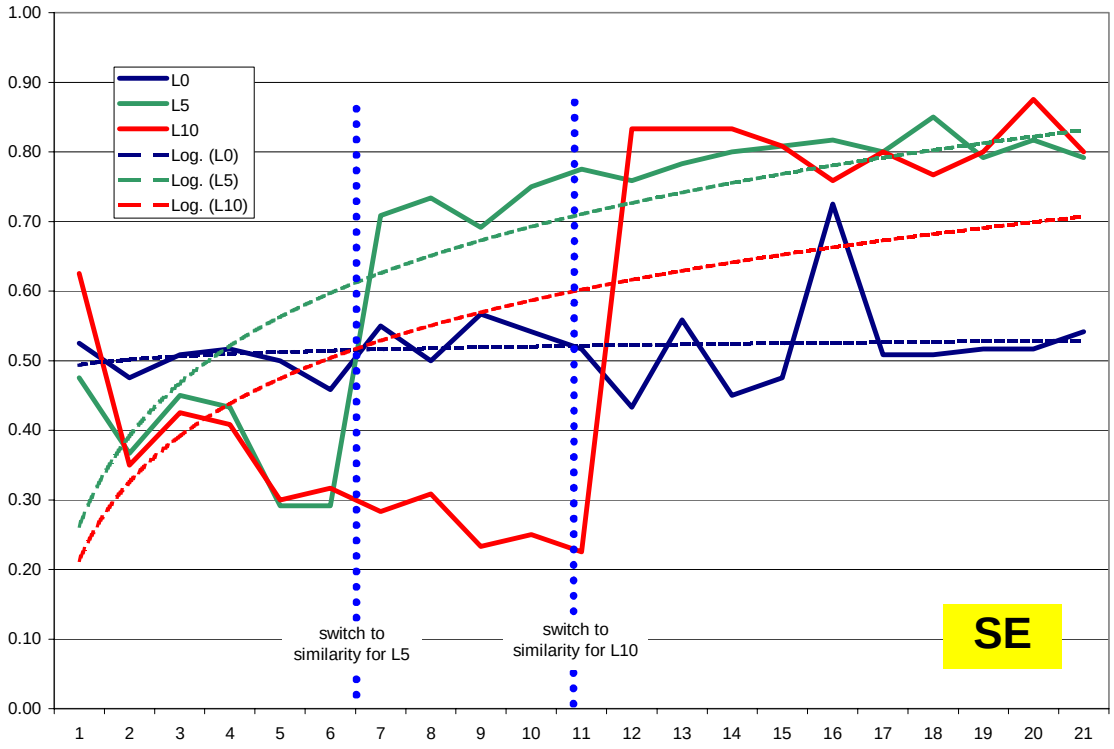


Figure 16: Results - SE subjects

*SE clusters*

Figure 17 shows the six graphs for the SE cluster subjects. The SE3 graph shows a case where the myopic agent probably landed in a part of the product space sufficiently different from the preferred sections and thus it took some time to "find" the right products to recommend. This behavior is one of the characteristic drawbacks of similarity-only recommendation engines.

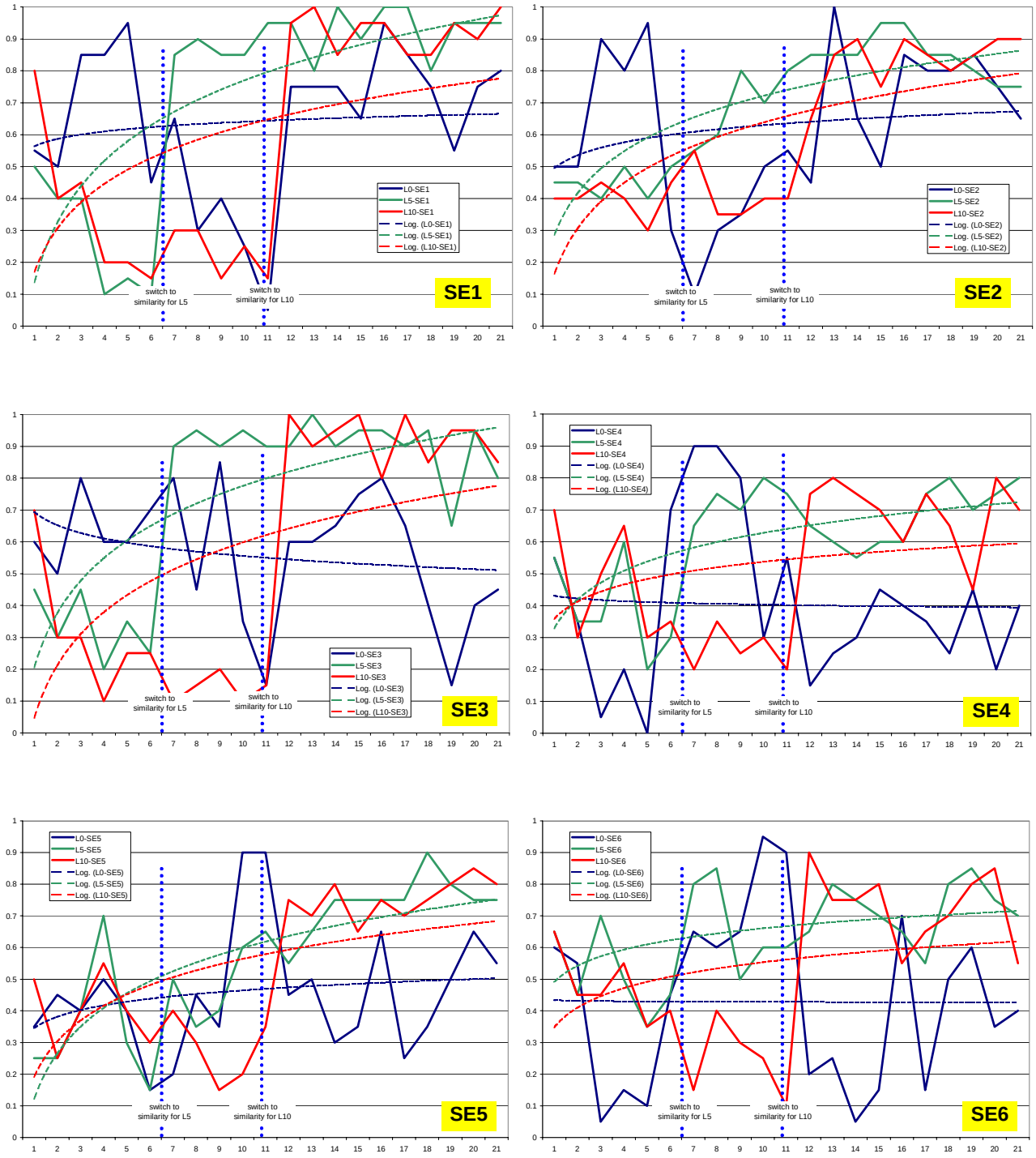


Figure 17: Results - SE clusters

**HLE subjects**

The hybrid large error data is probably the most complex of all the sets, as the subjects has large errors plus the "split personality". In this case we can see the value of larger learning phase, as L10 edges out L5 as shown in Figure 18.

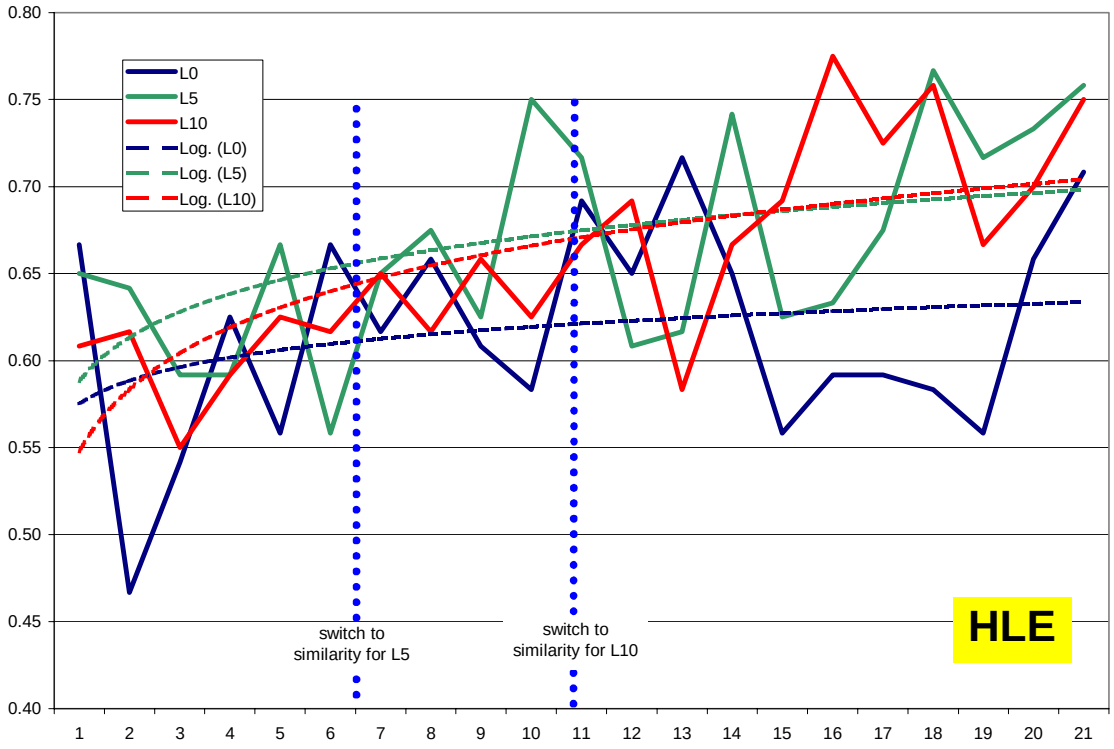


Figure 18: Results - HLE subjects

*HLE clusters*

The results for each of the HLE clusters are shown in Figure 19. In several occasions we find the L10 is valuable and the best predictor. HLE2 and HLE4, specifically, show the difference between L10 and the rest.

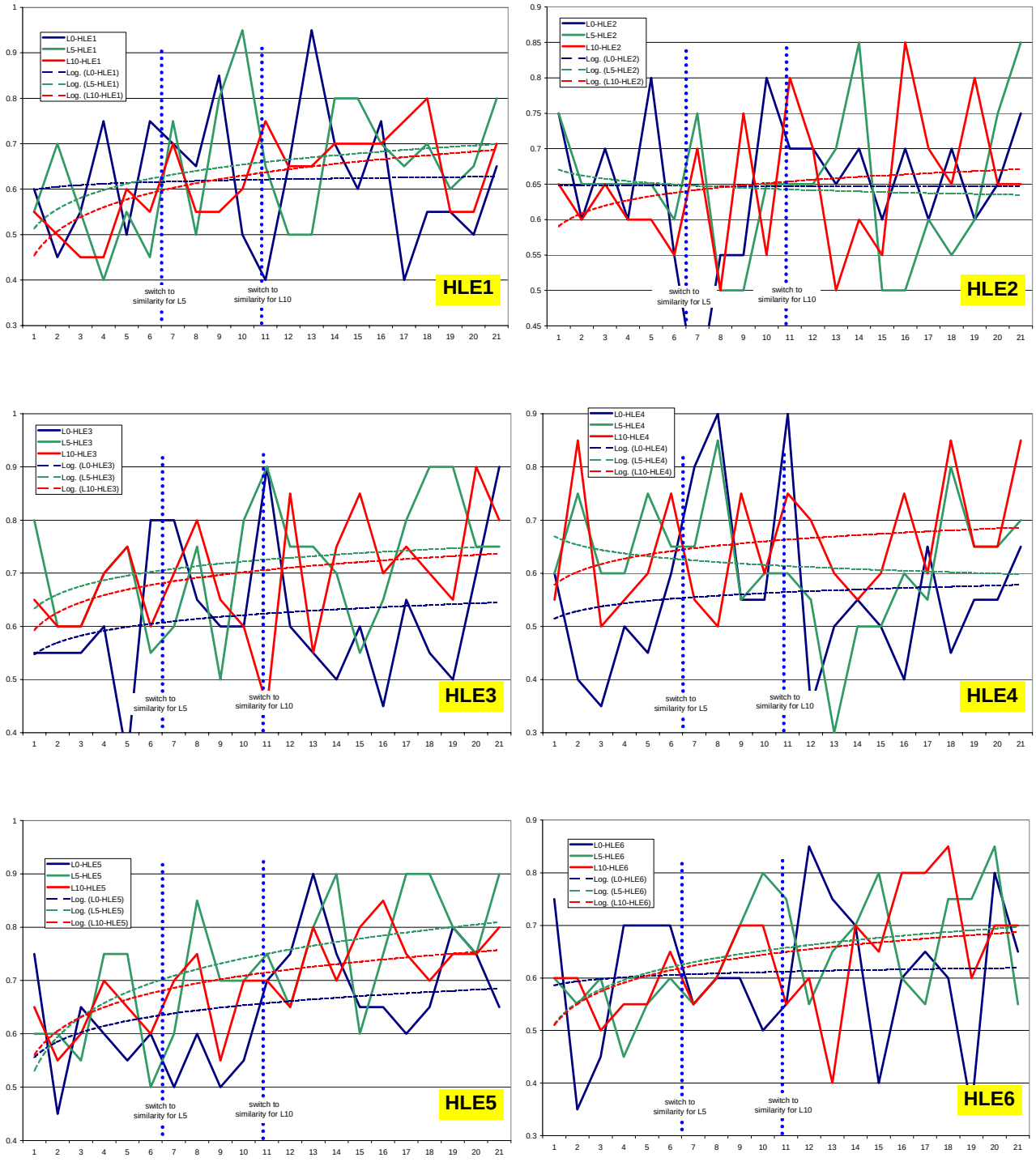


Figure 19: Results - HLE clusters

**HSE Subjects**

As before, this dataset also proves the higher accuracy of a longer exploratory phase, as shown in Figure 20. In addition, given the relative lower degree of noise (small error), our simplistic engine can better predict the users preferences, thus producing better results overall.

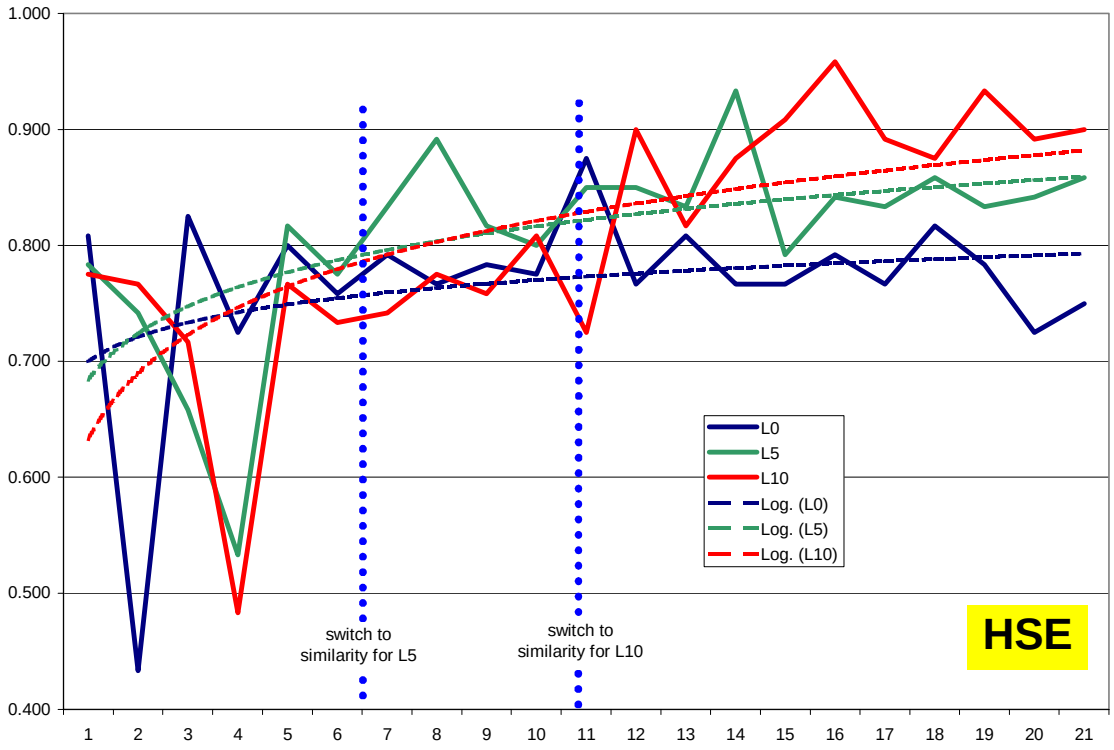


Figure 20: Results - HSE subjects

*HSE clusters*

Figure 21 shows the results for each of the six HSE clusters. Once again we can see that in several cases the longer learning phase proves useful to decipher these subject's split personality characteristics.

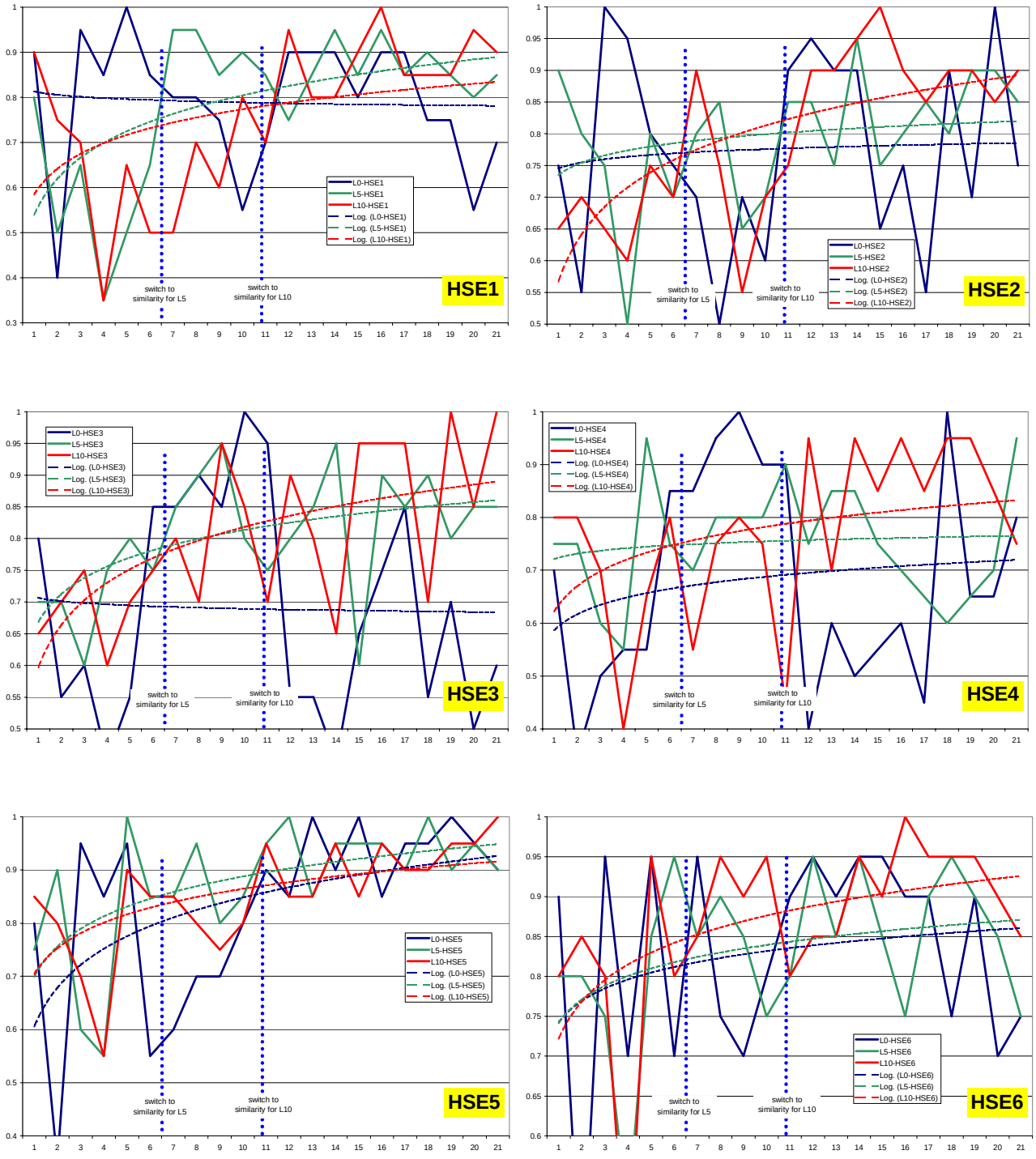


Figure 21: Results - HSE clusters

*Chapter 8*

# NEXT STEPS: USING A COMMERCIAL ENGINE

Although not implemented in this work, a natural follow-up would be to replace our simplistic recommendation engine for a full-fledged commercial engine, and re-test. This chapter presents a detailed analysis of how that could be accomplished with one of the leading recommendation engines in the market today.

## The Recommendation Engine

Even though current recommendation engines are similar in the semantics of their metrics, we will base this chapter on the specific information that derives from the recommendation engine we selected for this experiment. While we do so at some cost in terms of general applicability of the results, we hope that the increase in details that using a specific engine will allow for will more than outweigh the small incompatibilities for applying the results of this work universally.

We selected SaffronOne, from Saffron Technology in North Carolina, as the engine of choice for this work. SaffronOne was one of the few engines available that had a flexible enough programmatic API. Without such flexibility we would not have been able to reconfigure the selection algorithms to match our hypothesis. In addition, SaffronOne's choices of technology platform, design, and speed were compatible with our own experimental environment.

**SaffronOne**

The technology behind most recommendation engines are closely guarded secrets, since they embody the company's competitive advantage. SaffronOne is not an exception. Therefore, we have focused this chapter in explaining what are the metrics that we obtain from SaffronOne are rather than describing how exactly these metrics are computed.

SaffronOne has the concept of an "agent", which loosely equates to a personal advisor for each user. It also contains a concept of a "memory", or essentially a history of all the past interactions the user has had with different products. In addition, products are outfitted with "attributes" that characterize them.

An agent learns about a user's preference by observing an user's response (e.g. "LIKE") on a given product (more accurately, a vector of attributes). The `observe()` API call is used for that purpose, as shown in Figure 22. Internally, the SaffronOne agent is keeping track of the associations between the responses and the attributes, and in that way it learns about the user's preferences for a given response.

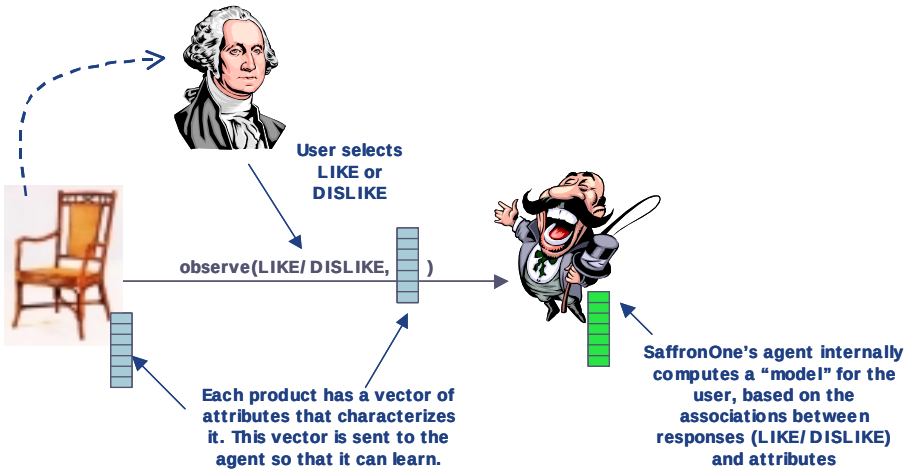


Figure 22: Learning in SaffronOne through the observe() call

In order to obtain the agent's rating for a given product, SaffronOne provides the imagine() call. As shown in Figure 23, the application is responsible for selecting first the product it wants ranked and second the type of response (e.g. LIKE, DISLIKE) it wants the ranking for. SaffronOne keeps track of associations by response types and thus an imagine() call with "LIKE" as the response target might yield a completely different result than a call with "DISLIKE".

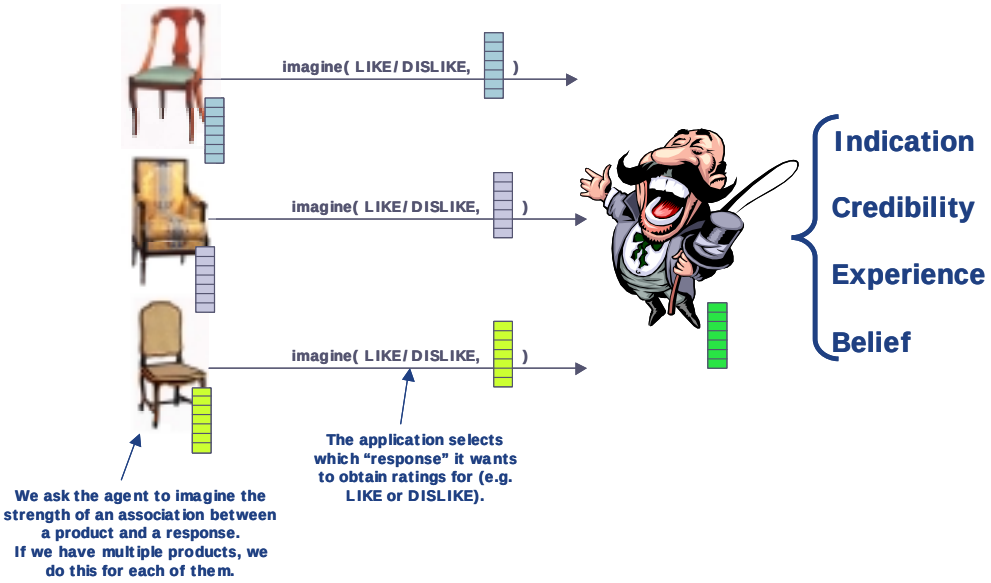


Figure 23: Rating different products in SaffronOne via imagine() call

Each imagine() call yields one primary measurement, Indication, and three secondary ones: Credibility, Experience, and Belief.

- **Indication** is a measure the similarity value, and the one most used in the recommendation engines.

- **Credibility** is a measure of both an individual's strength of "belief" along with that individual's level of "experience". It is equal to Experience times Belief.

- **Experience** measures the degree of familiarity between the attributes and specific associations. New features and new feature combinations will be calculated as lower levels of experience, whereas associations and features that have been seen many times before will translation into higher levels of experience.

- **Belief** measures the extremeness of the indications. One very strong probability is more credible than a number of weak ones. This is akin to "clarity". If one perspective looks like noise, it should be weighted less than another that more clearly perceives a strong indication.

**Combining different responses into one metric**

As described above, SaffronOne provides a set of metric for a given response type. Without getting ahead of us, our experiment (to be explain in subsequent chapters) requires the user to select one of two possible responses, LIKE or DISLIKE. The SaffronOne agent will therefore learn according to these two different response types. However, when we need to recommend a new product to the user we must find a way to combine the metrics from both response types into a single number that we will use to rank all the products.

**Indication**

The first measure we will define attempts to capture the overall "similarity" strength for a given product. This metric will be used to select products in the baseline version of our experiment, and resembles the selection criteria that traditional recommendation engines employ.

As Figure 24 shows, the indication value is computed via two consecutive imagine() calls for the same product, one with each possible value of the response type (LIKE and DISLIKE). These calls result in two sets of {indication, credibility, experience, belief} metrics, of which we only utilize the indication values. The overall product indication is thus defined as the difference between the indication that the product is LIKEd and the indication that the product is DISLIKEd. Since indication values lie in the [0…1] range, indication values lie in the [-1…1] range.

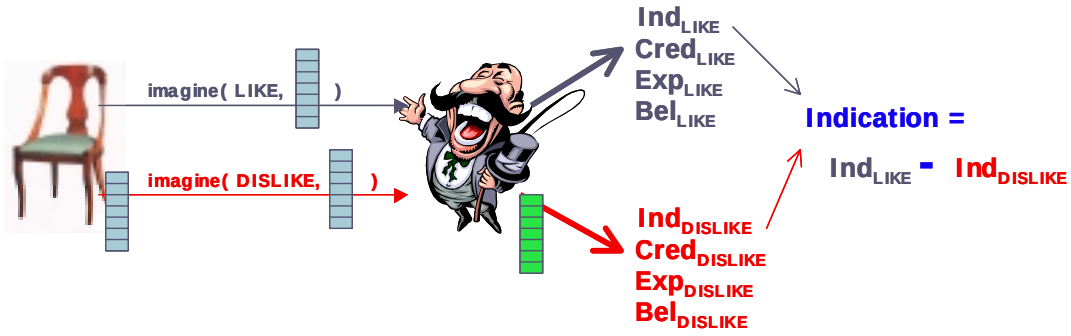


Figure 24: Computing Indication from Credibility
for LIKE and DISLIKE

**Novelty**

The second metric we need to define is needed to explore our hypothesis of an "exploratory" agent. We have used SaffronOne "experience" metric for this computation. The idea is to come up with a number that will represent this user's familiarity with the product we are ranking. In a sense, novelty is the opposite of experience, as defined in this work. With such a novelty measure we can embody our active learning hypothesis by sorting all the available (still not shown) products on their novelty values and selecting the product whose novelty is highest.

Figure 25 shows the computation of Novelty. Experience values start at zero and grow logarithmically towards one. In order to combine our learned values for both LIKE and DISLIKE responses, we first return the experience levels to their raw values via the exp() function. Notice that in this case we **add** both experience levels as we it is not relevant whether we LIKEd or DISLIKEd the product, but rather whether we "have seen" it before. Once we have added the raw experience levels we compute the logarithm to bring the value back into the [0…1]. Finally, since Novelty is a measure of "non-experienced" we invert the value subtracting it from one. Thus, Novelty values lie in the [0…1] range whereas low values mean well-known products and high values means unexplored territory.

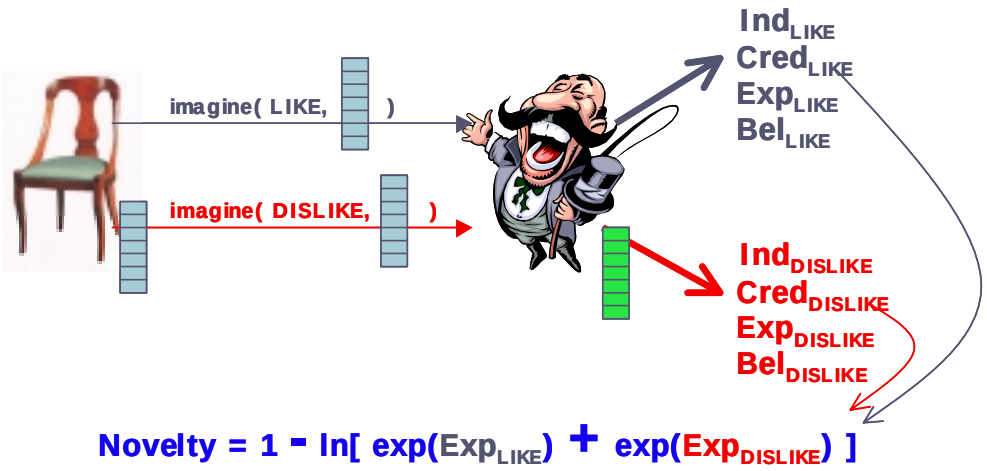


Figure 25: Computing Novelty from Experience of both LIKE and DISLIKE

### Implementing the Agents Using SaffronOne

In what follows, we provide specific implementation details for the two different agents: myopic and active, using the SaffronOne platform. All the details necessary to run the same experiment as presented before are provided.

*The Myopic Agent*

Figure 26 shows the myopic agent flow for each user. Vectors are denoted in bold font and "1" denotes LIKE while "0" denotes DISLIKE. Last, but certainly not least, we can build in all our agents the capability of dealing with LIKE-only results or LIKE and DISLIKE results, in order to be able to explore different active learning alternatives.

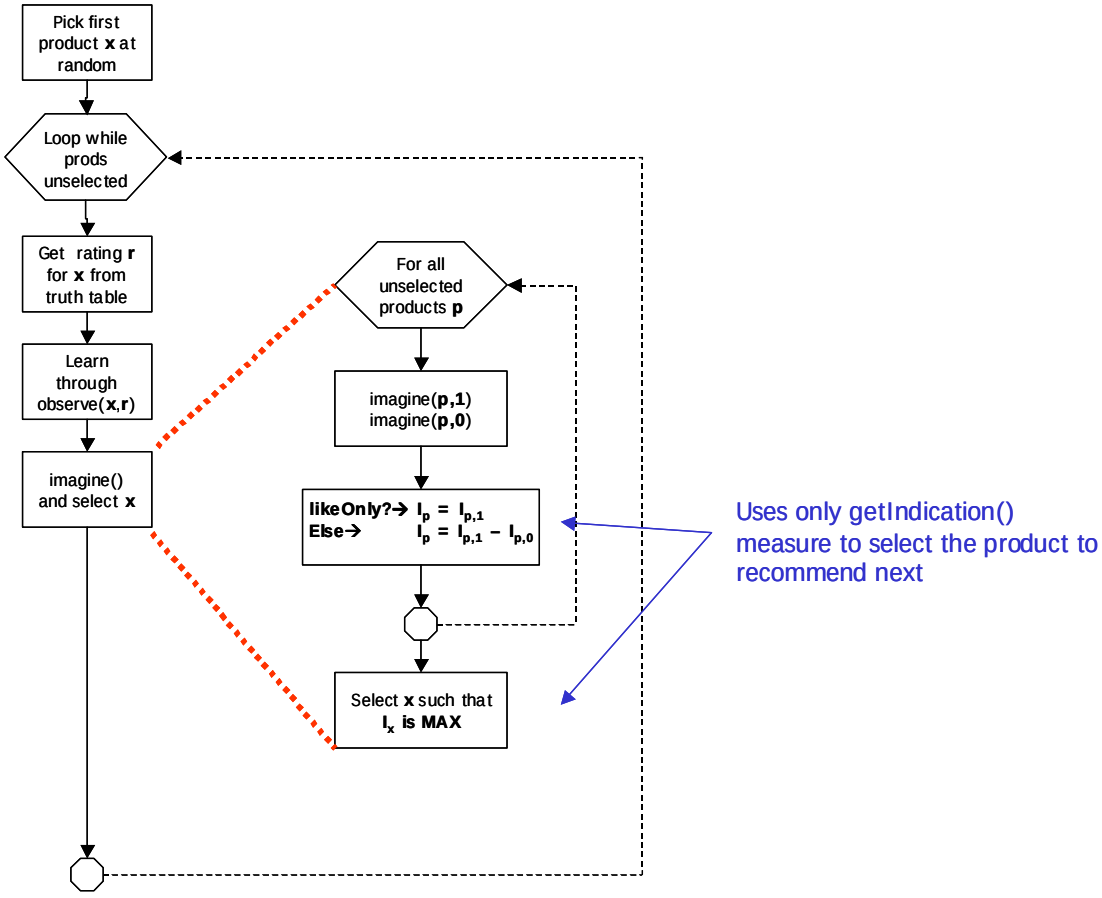


Figure 26: Myopic agent logic flow

The myopic agent is the simplest of them all. Products are "learned" by looking up in the rankings set to see if the recommended product was liked by this particular user or not. At recommendation time, we loop through all the as-yet-

unselected products and use the imagine() calls to obtain indication values for LIKE and DISLIKE responses. We compute an overall indication value based on these metrics. After we have all the indication values for the candidate product set we sort on it and recommend the product with the highest indication. The loop continues until there are no more products in the candidate set at which time we move to the next user.

*The Active Agent*

The active learning agent uses the "switch" parameter to control how many iterations it spends "exploring" the space before it switches to recommending on indication (as the myopic agent does). While exploring, this agent uses the experience values from Saffron to put together a measure of "novelty" for each product, ultimately selecting the product with the highest novelty value. When recommending on similarity, the logic is identical to the myopic agent. Figure 27 shows the active agent flow for each user.

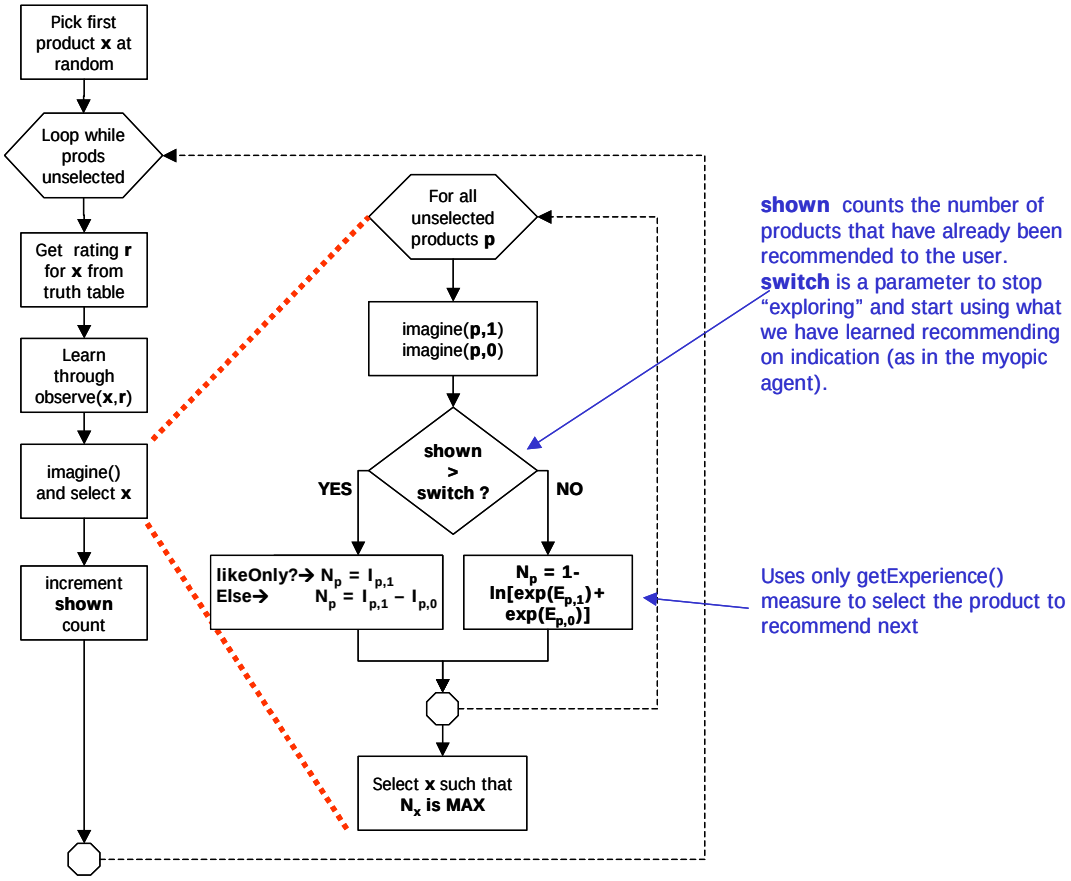


Figure 27: Active learning agent logic flow

As before, a provision has been made in the agent's logic to use LIKE only measurements, or LIKE and DISLIKE both. This provision is important for those cases where the response data is heavily biased towards one or the other (i.e. users dislike 80% of the products in the set).

### Technical details

The experimental setup can consist of several machines that run each agent separately but in parallel, to conserve time. Each of these machines must have the following environment:

- Windows 2000 operating system
- Java JDK 1.3.02
- SaffronOne V 2.01

All of the agents are written in Java, using the SaffronOne libraries. No other non-JDK libraries need be used. Input parameters can be read from plain text files and results can be output into text files as well.

Some of the SaffronOne calls used by the agents are not available to the public and thus it might be necessary to obtain a special “unlocked” version of the SaffronOne library to run these experiments.

*Chapter 9*

# CONCLUSIONS

The hypothesis of this work is centered on the need for recommendation engines to actively explore the space, at the expense of short-run success, to provide better overall recommendations. An overview of the field and the different recommendation engines opened this work, followed by a detailed description of our hypothesis and its associated metrics. In order to validate the hypothesis, an experiment was devised and executed. A detailed description of the data and experiment is provided. The results from the experiment and the associated analysis have been presented. Finally, detailed suggestions for future work in extending this work are given.

Our experimental results appear to validate our hypothesis. As shown in Chapter 7, active learning agents outperform myopic agents even in a short recommendation span of only 20 products, in spite of having to use some of those products up to explore the space. Some of our key findings include:

- Small active learning thresholds seem to be sufficient to learn about the space. In most cases our L5 agent outperformed out L10 agent. We believe that this might be due to the length of the recommendation span (only 20 products, 10 out of which were used for "exploring" by the L10 agent). However, the fact that L5 performs so well might be related to the number of real different preference clusters that might be available to the engine. This number might be, in all practical cases, small. Thus a well crafted engine that "touches once" each cluster and learns whether the user likes or not that "type of products" would be enough. If there are

less than 5 of these clusters then the other 5 exploratory recommendations of the L10 agent are essentially "wasted".

- Larger exploratory phases are indeed superior than shorter ones for the more complex cases. In agreement with our previous finding, we observed that for some of the most complex datasets (the split personality ones), the L10 agent outperformed L5. This is rational as the split personality feature of these subjects might have increased the number of "liked clusters" thus requiring more exploratory recommendations to learn about them all.

- Similarity-only engines can get lost in a sub-optimal state looking for a preferred cluster. In a couple of occasions we observed the myopic agent actually going down in accuracy as the iterations grew. Our belief is that this is symptomatic of the case when the myopic agent by sheer luck started in a very unfavorable region of the preference space and thus takes a long time to "get to" the closest preferred cluster, where its performance goes up.

Even though our results have been very encouraging, a reality check is in order. We used a very simplistic distance-based recommendation engine. Most commercial-grade engines should easily outperform this one and even include some of the ideas presented in this work. They do not, to the best of our knowledge, include the idea of orthogonality in the recommendation set however. In order to rubricate these results, the experiment should be re-run with a commercial engine, such as SaffronOne, as explained in Chapter 8. We expect the results of such a test to be less dramatic than the ones presented here, yet still significative.

# BIBLIOGRAPHY


Ansari, Asim; Essegajer, Skander; Kohli, Rajeev. *Internet Recommendation Systems*, Columbia University, December 1999.

Aparicio, Manuel; Strong, Paschal. *Propagation Controls for True Pavlovian Conditioning.* Book chapter, 2000.

Ariely, Dan; Lynch Jr, John; Aparicio, Manuel. *Which Intelligent Agents are Smarter? An Analysis of Relative Performance of Collaborative and Individual Based Recommendation Agents.* MIT, Fuqua, Saffron, 1999.

Gershoff, Andrew; West, Patricia. *Using a Community of Knowledge to Build Intelligent Agents.* University of Texas – Austin, 1999.

Glance, Natalie; Arregui, Damián; Dardenne, Manfred. *Knowledge Pump: Community-Centered Collaborative Filtering*, Xerox Research Centre Europe, 1997.

Macready, William G. *Tailoring Mutation to Landscape Properties*, Bios Group L.P., 1996.

Nichols, David M. *Implicit Rating and Filtering.* Lancaster University, UK, 1998.

Palme, Jacob. *Choices in the Implementation of Rating.* Working paper, July 1997.

Rabelo, Luis. *Which Intelligent Agent is Smarter? A Comparison.* SDM Thesis, MIT/Sloan, January 2001.

West, P.; Ariely, D.; Bellman, S.; Bradlow, E.; Johnson, E.; Kahn, B.; Little, J.;Schkade, D. *Agents to the Rescue?*, Working paper, MIT, February 1999.