1. Algorithmic Aspects of Web Intelligent Systems

Dimitrios Kalles[1 3], Athanasios Papagelis[1 3], and Christos Zaroliagis[2 3]

[1] AHEAD Relationship Mediators, 65 Oth.-Amalias St, 26221 Patras, Greece
[2] Computer Technology Institute, PO Box 1122 Patras, 26110 Greece
[3] Department of Computer Engineering and Informatics,
University of Patras, 26500 Patras, Greece
Email: kalles@aheadrm.com, {papagel,zaro}@ceid.upatras.gr

Summary.

We discuss the algorithmic aspects of a Web intelligent system which demonstrate several challenges encountered during the development of such systems on the Web. The particular system is an on-line fun portal that adopts several key ideas spanning from site design to user walk-through and extended user-input analysis, in order to *intelligently* adapt to user interests and, consequently, improve user experience. Our algorithmic techniques build upon concepts from machine learning and statistics (naïve Bayes analysis and user models), as well as on several ideas for efficient and timely data manipulation (incremental algorithms and extended caching), and allow to efficiently handle and analyze the collected data.

1.1 Introduction

World Wide Web (Web) has emerged, during the last decade, as one of the most prominent research fields. The Web has many commons with traditional research areas, while at the same time it differs in several aspects and opens up new challenges. As an example, one could argue that the Web is the rough equivalent of a huge distributed database. However, it differs substantially from traditional databases regarding the way it stores information (unstructured vs structured ways), its size (there is no database, no matter how big, that can be compared with the size of the Web) or the number of concurrent users (at any given time there are millions of users gathering data over the Web).

Web's size, heterogeneity and complexity challenge our ability to efficiently manipulate data using traditional techniques. In order to cope with those characteristics several Web applications require intelligent tools that help extracting, from this enormous amount of data, the proper information relevant to the users' request. The development of intelligent tools over the Web requires the innovative use of AI as well as of advanced IT and algorithmic techniques [1.24].

In this work we report on the algorithmic aspects of a Web intelligent system which demonstrate several challenges encountered during the development of such systems on the Web. The particular system is an on-line

fun portal, called myFreddy (`http://www.myfreddy.com`). MyFreddy adopts several key ideas spanning from site design to user walk-through and user-input analysis, in order to *intelligently* adapt to user interests and, consequently, improve user experience. MyFreddy combines into a demanding application concepts from several research topics. It builds on ideas from machine learning and statistics (naïve Bayes analysis and user models), agent theory (dynamic responses by examining the environment), friendly user interfaces (easy navigation, environment customization) as well as on several ideas for efficient and timely data manipulation (incremental version of algorithms and extended caching). Our system refers to a community of users that share common interests and consequently the algorithms used had to be adapted to the specific community needs as well as to Web limitations and characteristics.

Close but distinct relatives to our system exist in the form of various Web agents and recommendation systems. Those applications use text analysis (content based approach) and user matching techniques (collaborative approach) to assist users browsing the Web [1.16, 1.1, 1.8, 1.19, 1.20, 1.22, 1.21, 1.2, 1.6] filter usenet news [1.11, 1.13] or find specialized info on the internet [1.3, 1.9, 1.12, 1.17]. A well known recommendation system can be found at Amazon.com. There, users create their own profiles, which record the items they looked at or bought. Amazon recommends by looking at other users who bought the same items, and suggests items that the users have looked at or bought. Alternatively, the system can make recommendations based on the users' previous choices. Amazon's database could be extremely valuable if one could use it for research and analysis reasons. However, several commercial restrictions disallow public dissemination of those data.

MyFreddy was conceived as an advanced Web-based community based on voting subjects. The key features of the system are:

Support for multiple types of data. We internally support several types of data (text, image, sound, video), languages and voting subjects.

Easy navigation. The site is shallow, a few clicks are enough to reach any point inside it. This means that one needs a minimum amount of time to get familiar with its basic functionality.

Extended customization. The user can customize the environment according to his needs. The customization refers to both the site appearance and the nature of the material presented to users.

Content provided solely from users. The Web interface allows users to upload different kinds of content to the site. In return, the user can monitor how his content is doing through a personal page. Even though this can have serious consequences regarding the quality of the content, it helps maintaining a fresh, always updated site while at the same time it maximizes its value as a community.

Funny content. Although there are virtually unlimited types of voting subjects we decided to reduce the content space to those that produce fun.

Some of the current types of objects are jokes, quotations, funny comics and funny pictures.

Built around a visual Web agent. The site is built around a friendly visual intermediate called Freddy (after which the site was named myFreddy) that has been designed as a rendered 3D model. Its main purpose is to guide users through the different site functions, to reward them when they are voting well or to tease them when not. Freddy uses several types of feedback from users to dynamically synthesize funny responses.

Extended data analysis. The community incorporates several algorithms to suggest documents of interest to users, match users according to their voting patterns or predict their votes as an interactive game.

MyFreddy is a complete running system aggregating (as time progresses) big amounts of not necessarily structured data that require extensive data analysis. The amount and the characteristics of data make their effective analysis a non-trivial task.

In this work, we focus on the algorithmic techniques used in our system that allow to efficiently handle and analyze the collected data. We discuss the problems and challenges met in developing efficient algorithms and present various algorithmic techniques that select next documents to vote, recommend documents to users, predict users' votes, and match relative documents. In addition, we comment on advantages and disadvantages of the proposed algorithms on the given context. We note that MyFreddy allows for rapid testing of algorithms and serves as a working example of incorporating demanding algorithms into real-world systems.

The remainder of the chapter is organized as follows. Section 1.2 gives a brief description of our system and valuable insight on important architectural issues. Section 1.3 describes in detail most of the algorithms as well as Web limitations and problem characteristics that lead to current adjustments. The final section summarizes our contribution and gives an overview of how well the community did in the real world.

1.2 An Overview of the System

MyFreddy has been built around a small set of database driven Web pages. These pages communicate with the main algorithm's library as well as with the visualization library to produce the end user functionality. In this section we will give a brief description of the user-interface. The interested reader is strongly encouraged to visit the Web site (`http://www.myfreddy.com`) to obtain a more detailed view. We will also outline some key architectural issues that we have been dealt with throughout its development. These include performance issues, transparent user authentication techniques as well as the agent's inference engine.

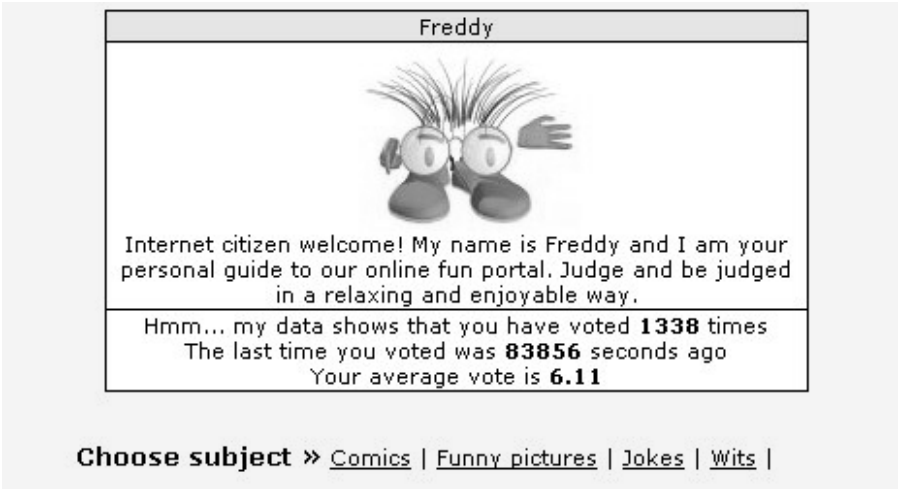

Fig. 1.1. The first page of www.myFreddy.com

1.2.1 User Interface

In myFreddy every user has to vote for each shown object in order to see a *randomly chosen*[1] new one. Here, contrary to the majority of sites that use voting schemes, it is obligatory to vote in order to browse through the content. This ensures that we collect a large amount of data that characterize the user and can be analyzed and combined in several ways at a later step. Moreover, this characteristic reduces the complexity of the site from the end-user perspective and the overhead of having to choose content and *then* vote for it[2] (assuming that we always want to get feedback from users).

When a user accesses the site he receives a personalized greeting from Freddy; see Figure 1.1. Assuming that the user selects the *Jokes* subject, he sees a page like the one shown in Figure 1.2. A document has already been fetched and presented to him. On the top of the page the user can vote for the specific joke on a range from 1 to 10, where 1 means a very bad joke and 10 means a very good one. On the left of the page there is a search form, which he can use to search for documents containing specific words. There is also a list of the community members that have contributed most of the jokes and have received the highest average ranking for their contributions. On the bottom of the page there are several links to the best and worst voted documents of the specific subject, to the user's personal page where he can

[1] The selection of the next document is not actually random even though the difference is not conceivable from the end user point of view. Later we shall see algorithms for *next-document* selection.

[2] The overhead of having to select and then vote a document is well described in [1.4], where the authors consider implicit ways to get users' feedback, like the amount of time a user spends on a specific page.

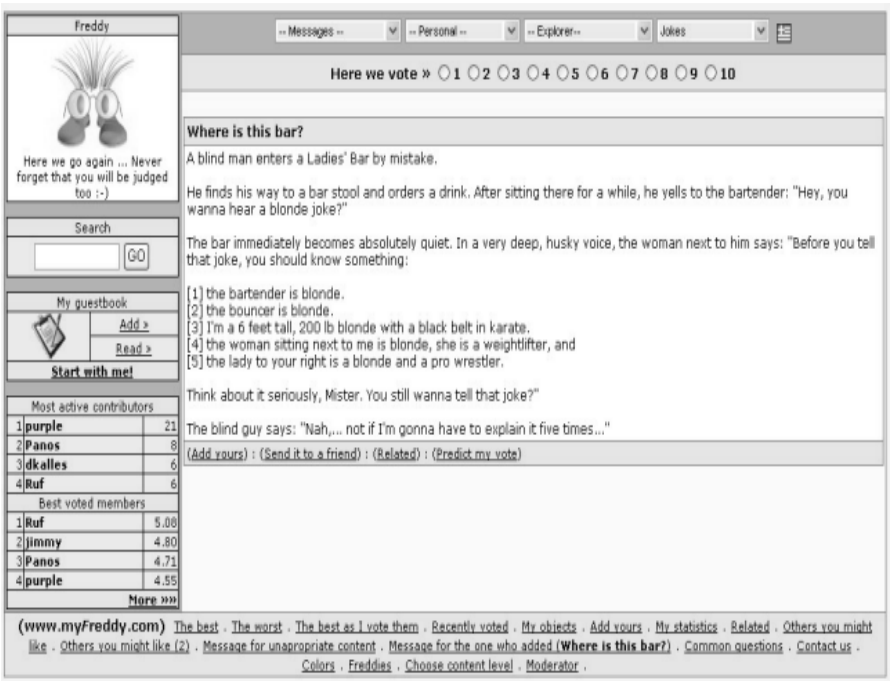

Fig. 1.2. The main voting page of Jokes

monitor his own contributions, to send a specific object to a friend or make comments about it, as well as to get document recommendations.

Following a user's vote, a new document is automatically fetched and presented to him. On the left panel the user can now see his vote and the average vote of the previous document. Freddy also reacts to the user input with a specific comment. For example, in case of a bad vote Freddy will not miss the opportunity to tease the user. Figure 1.3 presents two possible comments made by Freddy after a good and a bad vote. Currently, the community maintains a pool of more than 150 comments and more than 50 Freddy images. Those comments can be activated under certain combinations of input conditions, to be detailed later.

Figure 1.4 presents the user personal page from where the user can monitor his submissions. The vertical bars indicate the amount of votes that a specific object has received for every voting level (from 1 to 10). Using this page the user can also monitor for each object he has submitted the average vote and the total votes the object has received as well as its type.

In the bottom part of a voting document (see Figure 1.2) there are links to add a new object, send it to a friend, find relative documents or predict the user vote.

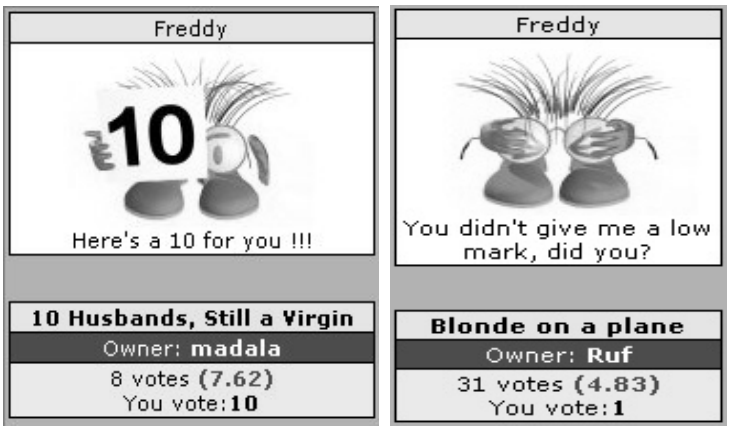

Fig. 1.3. Two Freddy comments for a good and a bad vote

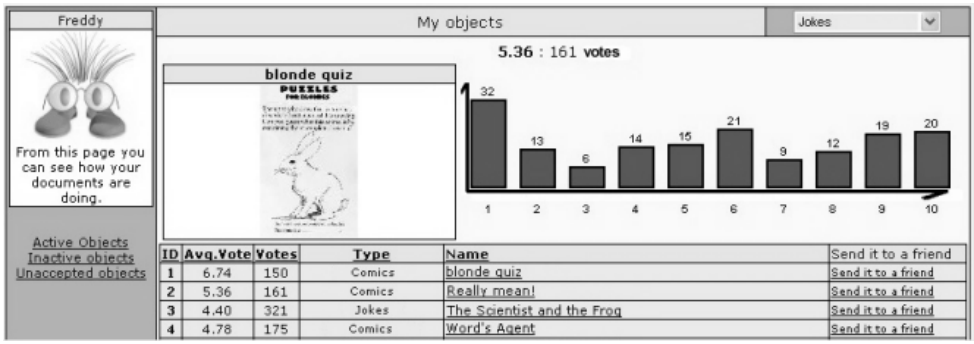

Fig. 1.4. The user personal page and details about the submitted objects

In order to be able to find related documents, suggest interesting document collections or predict user votes, myFreddy incorporates several algorithms that will be described in Section 1.3. An example screenshot of the vote prediction game is given in Figure 1.5. The prediction is based on continually updated user models trained using user votes for previous documents. For the specific joke at Figure 1.5 the system predicts that the user will give a low vote. If the user actually gives a high vote, then the user-model is incrementally updated to reflect this change (see Figure 1.6).

If the user chooses to see related objects, then a page similar to that in Figure 1.7 is presented. The user can choose among three algorithms to retrieve different types of relative documents.

If the user wants to see other documents he might like under a specific subject, then he can choose the relative option (*Others you might like*) at the bottom of the main voting page (see Figure 1.2). An example collection of recommended documents for the specific user is presented in Figure 1.8. These

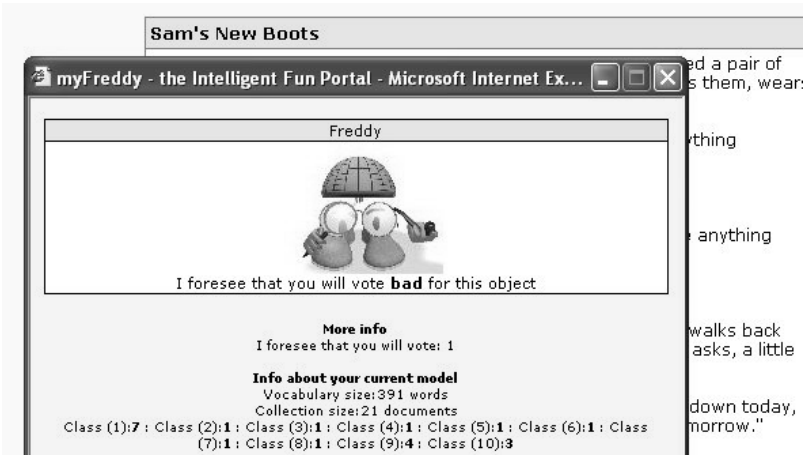

Fig. 1.5. Vote prediction and user-model statistics

documents are the result of an extended analysis based on voted documents and matching techniques between users.

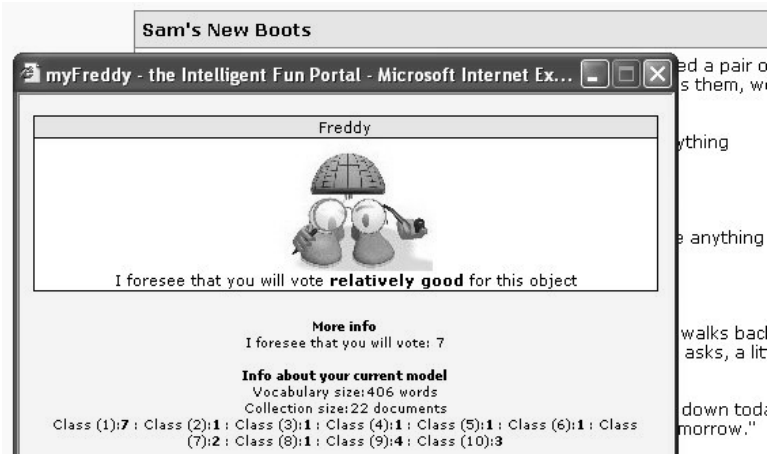

Fig. 1.6. Updated vote prediction

1.2.2 Performance

Performance considerations were a common topic throughout the development of our system. The nature of client/server computing poses heavy computational overheads on the server side. Moreover, the basic forms of the

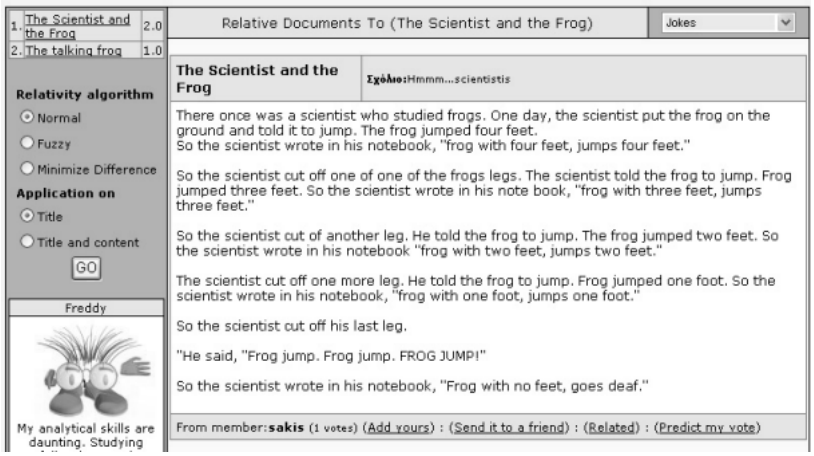

Fig. 1.7. Suggesting relative documents using various techniques

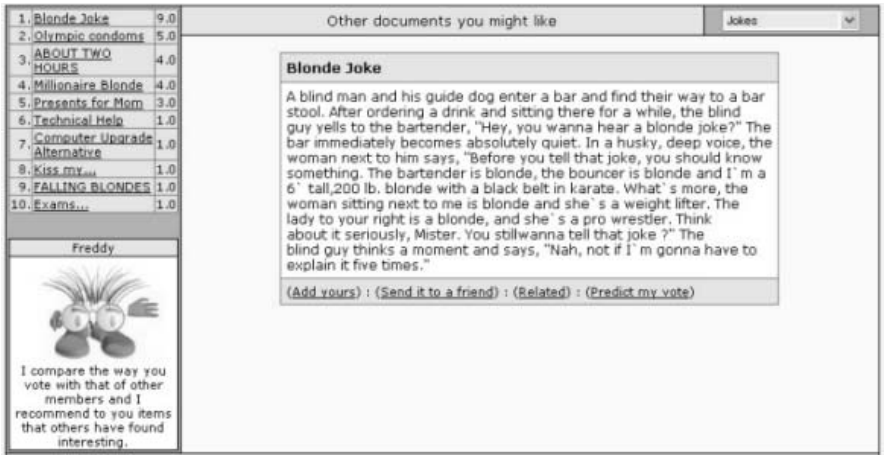

Fig. 1.8. Suggesting interesting document collections

algorithms used are very demanding regarding processing power. The above combined with the imperative need for immediate responses from the Web server (a Web surfer should not witness long inactivity periods) suggest that we should deal with performance in a holistic way.

Caching can help considerably towards this direction. In our case, caching does not refer to the information retained in main memory, because a complete program-instance exists only for the production of a single page. When the page construction is completed any allocated memory is released to the system. With caching we refer to the database which logically connects the pages inside a user browsing session.

For the needs of the community we have developed a database caching scheme. This scheme uses a small set of functions to control the cache which can be populated using various types of data; from parts of HTML code to serialized arrays or lists. Even though the database is less efficient than memory, it is still valuable when the already created data is computationally expensive and have a high probability to be used again in a short period of time. An example of such a case is the production of sets of suggesting documents (see Figure 1.8). These sets are computed the first time a user requests them and remain in the cache for a given period of time allowing the user to traverse through them without wasting computational resources.

Our application-specific caching is in accordance with recent studies regarding the negative impacts of long Web page delays on the produced users' satisfaction [1.23]. Those studies indicate that the biggest delaying factor is the page size. However, intelligent Web applications usually require extensive inner data manipulation and thus the page production can also be a significant delaying factor.

1.2.3 Users and authentication techniques

In order to be able to analyze the preferences of users the system should somehow distinguish between different users. A basic and common scheme to achieve this goal is a login/password system. Such systems are secure enough and they have the big advantage of universal applicability: a user can use them from every place and with every personal computer. However, such a solution is an unnecessary burden for the majority of users who just want to browse through the content.

For this type of users we adopt a transparent recognition procedure through cookies [1.5]. When a user first enters the site we search for a specific identifier in his computer. In the case such an identifier is found we pass it to the open browser session and we use it to mark the user votes and actions. Otherwise, we create a random identifier and we pass it to his computer for later use. This procedure does not run the risk of unveiling any user personal data (the identifier is just a random number) and it is totally transparent for the end user. A disadvantage is that the user can be identified only when he accesses the site from his well-specified environment.

For the advanced type of users that actively contribute material to the community and want superior security and personal data accessibility, myFreddy offers a personal page that can be accessed only through a login/password system. Thus, we make use of both types of authentication in order to match diverse requirements.

1.2.4 Agent's inference engine

Freddy is the visual Web agent and also the persona of the site. It reacts to implicit or explicit user input by providing a comment or a helpful tip.

Currently, it uses the vote of the user, his relative position inside the site and the characteristics of the voted document to produce a dynamic response. We are planning to extend its input to include the intermediate time between user votes, the time that passed before the user last entered the site, characteristics of the user voting sequence, the time of day and some forms of aggregated derived input. All perceived inputs are interpreted to a small set of *conditions* that are, on their behalf, matched with several possible agent images and comments. A specific set of inputs can activate several conditions which mean several prospective comments. To solve possible conflicts we have created an internal ranking of the conditions according to their comparative significance.

1.3 Algorithms

The Web community of MyFreddy collects big amounts of semi-structured or unstructured data which can be combined and analyzed in several ways. Effective data analysis is usually accomplished by algorithms which take into account the characteristics of the data. In this section we will first review the goals of data analysis, then describe the characteristics of the collected data and how they affect our problem solving approach, and finally we will present algorithms that select next documents, recommend documents to users, predict users' votes and match relative documents.

Potential goals of data analysis are to:

- Pull up objects relative to the currently presented one.
- Choose next objects that the user would like or avoid others he would not like.
- Create object collections that the user would find interesting.
- Predict the user vote as an interactive game.
- Suggest other community members with similar interests.

The algorithmic techniques we will describe are targeting at least one of the above goals.

Our problem solving approach is determined and to some degree limited by certain preconditions. First of all, we are looking for completely automated yet quality solutions. Otherwise, the maintenance requirements would be prohibitive and the value of the proposed enhancements would be negligible. In addition, we are looking for computationally feasible solutions putting into the equation the imperative need to satisfy multiple users in real time.

Data-mining techniques can satisfy the need for automated data analysis. However, they do not fully satisfy the other two requirements: solution quality and speed. Most of existing data-mining algorithms can be used over huge amounts of data and the maintenance costs are low. However, the results may be misleading or incomplete, and also the whole process is rarely optimized for real time analysis.

1.3.1 Data characteristics and generic handling techniques

It is important to distinguish between the different logical-types of the collected data. These types include the title of objects, the content of objects and the users' votes over objects.

The distinction between the title and content of objects stems from two reasons. Firstly, every object no matter of its type has a title, which allows us to use text-oriented techniques over all titles. Secondly, the size of the title is usually at least an order of magnitude less than that of object's content which allows for computationally efficient manipulations. Throughout this work we refer to objects that contain only text as documents.

Except for their different logical types, the collected data have additional characteristics which pose several problems for their efficient manipulation. These characteristics involve the amount of data, their heterogeneity, and the noise they may include.

Huge data amount. The collection of data expands either when a community member submits new objects or when a user votes for some object.

To deal with the amount of data we have used several techniques. We have already seen a caching scheme, where all computationally expensive derivations are cached to a database. Two other valuable techniques are:

- *Incremental algorithms.* Incremental algorithms incorporate new training data without having to re-compute the output on the entire training set. These techniques are highly used in machine learning and statistics.
- *Data filtering.* Data filtering produces re-sampled subsets of the original data. The idea is to produce a much smaller set of *representative* data which can be efficiently handled. Several re-sampling techniques can be used from extremely simple ones (pick a random data portion) to complex ones (pick a representative data portion). Those techniques pre-process data for later manipulation and their quality can significantly affect the output of the algorithm.

Heterogeneous data. Data includes votes, unstructured text, pictures and possibly video and sound. To efficiently deal with diverse types of data we need to make use of miscellaneous methods and techniques. Text mining techniques can be used over all object titles as well as with text based subjects. Image mining and video mining can be used accordingly with image and video based subjects. The complexity of those techniques can be daunting even for the 'simple' text case. The research community is currently focused on unstructured text documents due to their important practical applications. However, a range of techniques will probably emerge for others types of unstructured data.

Noise. The collected data suffer from increased levels of noise. A user can upload an object with several grammatical flaws or existing pictures with alternative titles and sizes. More seriously a user voting pattern is usually

the result of a non deterministic process that can be influenced from several external parameters like the time of the day, his mood, the time an object took to download or whether the user has seen it again. Most of the presented algorithms were selected to efficiently deal with noisy data.

1.3.2 Choosing the next document

We have already seen that when a user votes, a new document is immediately fetched and presented to him. From his point of view the selection of the next document seems the result of a random process. However, to choose randomly from the document pool is just one of the possibilities.

Actually, the basic form of random selection has some disadvantages. A user can find himself voting for the same object two or three times in a row. This risk is minimized when the pool of objects is substantially big. Still, during a voting session there is a non-small probability of seeing some of the same objects. An important feature of our system is that it offers opportunities to achieve the same functionality in more interesting ways. Some of the techniques proposed for this problem include:

(i) *Random selection among the objects that the user has not voted for.* This model in variations is the one currently used. Its basic benefits are that it produces a better user experience and it can be computed effectively using standard database indexing techniques. The basic idea is to exclude all objects that the user has already seen and choose randomly from the remaining set of objects. A special condition occurs when a user has seen all objects of a specific subject. In such a case the algorithm returns the object he voted least recently.

(ii) *Random selection among the objects that the user would probably like.* This technique filters the objects based not only on whether the user has voted them or not, but also on a metric of how interesting he would find them. The algorithm is an extension of the previous one and uses a vote prediction metric that will be quantified later in this section. The idea is to filter out all the random selected objects for which the predicted vote is below a voting limit. This limit controls how expensive the algorithm can be. When it is set to a high value the algorithm filters out many objects and needs much more processing power -on the average- to find a suitable return candidate. This effect can be reduced by introducing an upper limit on the number of objects that can be checked. When this limit is exceeded we return the best candidate so far. Again, when the user has voted all documents we return the object he voted least recently.

The pseudocode of the algorithm that returns objects that the user would probably like is presented below:

Input. User U, a collection of documents S, vote limit V_l,
documents limit C_l.

Output. A document that user U has not voted and probable will like or if he has voted all documents the one he saw earlier in time.

Algorithm.

Get the set S_v of objects that the user has voted.
if $|S| > |S_v|$ **then**
{ Select a random object X that belongs to $S - S_v$.
Predict the vote V of U over object X.
if $V < V_l$ **then**
{ Save X and its value V.
Increase counter C.
if $C = C_l$ **then** return the object X that produced the biggest V among the C_l saved objects
else return to choose another random object.
else return object X.
}
if $|S| = |S_v|$ **then**
{ Select the $|S| - 1$ last objects, S_x, that the user has voted.
Return the single object that belongs to $S - S_x$.
}

1.3.3 Finding interesting object collections and predicting votes by matching users

The basic idea is to build a model for user A based on his voting pattern and then compare this model with all other user-models in order to find user B who is *closest* to A. Closeness is measured in terms of the *distance* $D(A, B)$ between two users which is calculated using a function based on the vote difference over the set of documents they have both voted. Let A_d be the set of documents that user A has voted and B_d be the documents that user B has voted. Let also $C(A, B) = A_d \bigcap B_d = \{Z_1, Z_2, \ldots\}$ be the set of documents Z_i that both A and B have voted, and let $V_{X,Y}$ denote the vote of user X over document Y. Then, the distance between two users is calculated as

$$D(A, B) = \left(\sum_{i=0}^{|C(A,B)|} (V_{A,Z_i} - V_{B,Z_i})^2 \right)^{1/2}$$

If users A and B have the minimum difference for the commonly voted documents, then they will most probably continue to have a small difference over the set of documents that B has voted, but A has not voted yet. Thus, if we return a collection of documents that A has not voted and B has voted and find interesting, then it is likely that A will find them interesting too.

Although this idea is simple and powerful there are some limitations. For example, the closest to A user B, might have voted fewer documents than

A, or the remaining collection of documents might be of low relative quality. In such a case, finding user B is of no value since we cannot produce a set of documents that A will find interesting. An extension over the basic idea, that builds on k-nearest-neighbors classifying methods [1.18], maintains an order of the relative distance between A and all other users and builds the set of returning documents based on several "neighbors" of A.

Another problem is that of the required computational power. We would like to have a small average response time. Since the community can have an unlimited number of users, using the entire set of users to build the returning set of documents could result in significant (or even enormous) delays. To reduce the computational cost, we use a heuristic approach that allows us to keep the average response time to a minimum. This approach reduces the number of users by selecting those with the most votes and the maximum cardinality of the documents set by selecting a portion of the documents that have been voted from each user.

The pseudocode of the algorithm that returns documents by aggregating results from several close-to-A users is given below.

Input. User A, maximum number of users x to compare, maximum size of document set Δ, maximum number of documents t to return, maximum number of users z from whom to select the returning objects.

Output. A set of documents that A has not voted and probably would find interesting.

Algorithm.

1. Create a set U_x by choosing the first x users among those with the most votes.
2. For every user $B \in U_x$ find the subset $C_\Delta(A,B)$ ($|C_\Delta(A,B)| \leq \Delta$) of objects that have been voted from both A and B.
3. Compute $D(A,B)$ over the subset $C_\Delta(A,B)$.
4. Insert user B to a list of users ordered by increasing distance from user A.
5. Create a user-set K by taking the z first users from the ordered user-list.
6. From every user inside K, get the top t documents that this user has voted, but A has not voted.
7. Return a merged set of top t documents regarding their combined average vote from every member of K.

A collection of documents that has been created using this technique is presented in Figure 1.8. The referenced user is the one currently browsing. This user is compared with other community members to produce a set of documents that he has not voted and could find interesting.

User pattern analysis can also be used to predict the user vote for a specific object. The basic idea is to find the closest to A, user B, who has voted the object of interest. Since the users are close enough we can – with

some confidence – assume that the user A will give the same vote that B gave for the specific object.

This technique is expensive. For making a good prediction we must compare user A with as many users as possible. The process cannot be accelerated by caching already computed user-models because they continously change due to new votes. In order to get a prompt reply we have to drastically reduce the amount of users and objects. In such a case, the accuracy could be seriously affected. However, this technique requires further investigation as it has the big advantage that it can be used over all objects and not only over text documents.

1.3.4 Finding interesting documents collection and predicting votes using naïve Bayes analysis

In order to build data structures that efficiently predict a user vote for presented documents or create collections of documents that the user would probably like we use the naïve Bayes algorithm to analyze past user votes in conjunction with the content of each voted document. We will give a brief overview of the Naïve Bayes algorithm and how it can be used for text classification. The interested reader can find in [1.18] a detailed description of the Bayes and naïve Bayes algorithms.

The voting procedure can be considered as a function of the problem *How interesting did you find the document?* that assigns values to documents on a range from 1 to 10:

$$Interesting? : Document \rightarrow \{1, 2...10\} \tag{1.1}$$

In general, we have a function $f : X \rightarrow V$, where X is the domain consisting of elements $x \in X$ represented by tuples of attributes $\langle a_1, a_2, \ldots, a_m \rangle$, and V is the set of all possible values of $f(x)$. For the *Interesting?* function we consider as attributes the collection of all words and other tokens inside a document. We are interested in computing the best prediction of $f(x)$ given by

$$Pred = \max_{u_j \in V} \Pr(u_j | a_1, a_2...a_m) = \max_{u_j \in V} \Pr(a_1, a_2...a_n | u_j) \Pr(u_j) \tag{1.2}$$

The naïve Bayes classifier makes the assumption that the attributes are conditionally independent:

$$\Pr(a_1, a_2...a_m | u_j) = \prod_i \Pr(a_i | u_j) \tag{1.3}$$

Substituting Eq. (1.3) into (1.2), we get that

$$Pred = \Pr(u_j) \prod_i \Pr(a_i | u_j) \tag{1.4}$$

The required probabilities $\Pr(u_j)$ and $\Pr(a_k|u_j)$ for our example function can be computed as:

$$\Pr(u_j) = \frac{|docs_j|}{|docs|} \tag{1.5}$$

$$\Pr(a_i|u_j) = \frac{n_i + 1}{n + |vocabulary|} \tag{1.6}$$

where $docs_j$ is the collection of documents for which the user assigned the value j, $docs$ is the collection of documents that the user has voted, n is the total number of words inside the collection $docs_j$, n_i is the number of times that the word a_i occurs in the collection $docs_j$, and $vocabulary$ is the collection of all distinct words inside $docs$.

We refer to the process of creating the required naïve Bayes structures by using the textual content of every object that a specific user has voted as the *training process*. The output of this process is a collection of structures that characterize the voting behavior of a specific user (the user model).

In a second step we use the produced structures to calculate the probability with which a new, possibly not voted, document belongs to each classification (in our case the possible classifications are the voting levels from 1 to 10). According to the naïve Bayes algorithm the classification with the highest probability is our voting prediction. With this technique we can create sets of documents that the user has not voted and have high predicted votes. These documents are likely to be of interest to the user.

A problem with the above technique is the required processing power. In order to create and use the user model, the system must analyze a big number of documents. A valuable idea is to incrementally update the user model immediately after each user's vote, instead of building it from scratch when needed. This way, we can split the training overhead into smaller processing portions.

In accordance with this idea, each intermediate user model is saved to the database. Figure 1.9 illustrates the process of updating the required structures. We assume that after a reasonable number of training documents the naïve Bayes structure should be robust enough to make accurate predictions. After this point we do not further update the user model to save processing power.This training process is transparent to the end user, since it is completely based on his voting feedback.

There are several alternatives to the proposed incremental updating model. An interesting idea stems from the fact that the user behavior is not static but it dynamically changes through time. According to this idea the user model should be continuously updated with new data while at the same time old data should be removed from the model. However, since the model does not preserve the states from which it was built, this technique is difficult to be achieved using the current data structures. Still, it could be an interesting research direction to analyze the dynamics of such an extended model.

Note that Figures 1.5 and 1.6 illustrate the effect of previous votes-and-content analysis on object categorization. These figures also give information about the votes distribution and the naïve Bayes structures of the specific user.

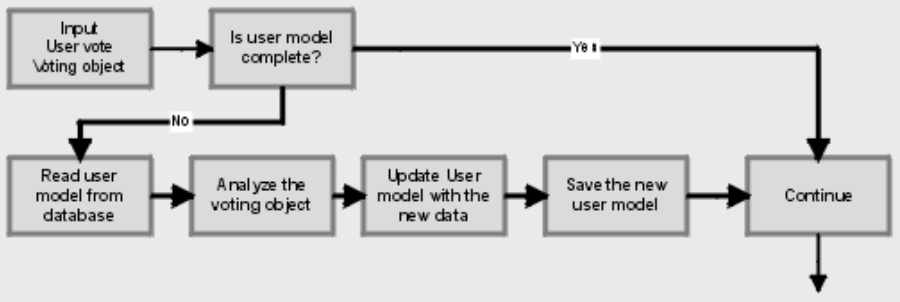

Fig. 1.9. Incremental updating of the naïve Bayes training structures

1.3.5 Matching relative documents

The process of matching documents produces sets of documents that share similar characteristics. The whole procedure is based on an initial referenced document and it can be thought of an extended type of search: we use every word inside the document as a search word and return the documents that maximize the aggregated appearance of those words. Basic differences between the classic and the proposed search are the increased levels of noise allowed and the amount of words that can be used to perform the search. A general problem is that there are no universally acceptable ways to interpret the similarity notion so our problem approach is by default biased.

Standard (basic) search has many disadvantages given the problem context. First of all, it cannot distinguish between the relative importance of words. For example, in the *Jokes* subject the world *blonde* is of high importance since it formulates an important matching direction between documents. Furthermore, there are many words with little information value (e.g., the conjunction *and*) that can misguide the output. Also, due to the dimensionality problem [1.18] the search is biased towards bigger documents. For certain languages, e.g., the Greek language which is also our concern, the problem is even more complicated because each word can appear in several different forms. However, this type of matching has a big advantage (considering the nature of the community): it can be implemented in a very efficient way using standard database indexing and searching techniques.

In order to deal with the increased noise level, we have normalized every word inside documents. The normalization procedure removes special characters, converts all letters to lowercase, removes spaces and, for the Greek

documents, replaces each letter with a similar sounding Latin character. We have also used specific word filters to remove less important words. A basic word filter removes every word with less than three characters. In a higher level, we populated a customized dictionary with words that usually have small information value and should be blocked (a future research direction dictates the evaluation of existing expertise on language processing).

To improve the basic search, we have tried several alternative matching techniques. Two of them are based on soundex [1.10] and metaphone [1.14] keys. The first uses a simple heuristic that transforms words so as similar sounding words to be considered a match. The second adds grammatical rules to the words conversion. Both algorithms introduce a fuzzy transformation allowing more words to match.

Another technique computes the distance (called Levenshtein distance) between two strings as the minimum number of insertions, deletions and modifications that are needed in order both strings to become identical [1.15]. Using this technique we can identify as relatives the documents that minimize their content distance. Again, all algorithms can be used either over the content of documents or over the title of objects; the latter preserves computational resources.

All these techniques produce a rich set of potential document matchers with different characteristics. The basic search is straightforward and fast but fails to discover certain obviously matching documents. Soundex and metaphone improve on that allowing more documents to match but with increased probability to return false matches. They are slower than the standard search but still quick enough. Levenshtein distance introduces ever more fuzzy characteristics to our search procedure. Now, objects are implicitly compared not only on their content but also on their size or their type (e.g., do they include dialogues?). However, Levenshtein distance has also the highest probability to return falsely matched documents and it is very expensive to be used over documents content.

Table 1.1 summarizes all these algorithms. According to the nature of the community we have also variants based on the type of the source document and the type of the target document on which the algorithms apply. There are two different types, namely title –denoted by T– and the combination of title and content of objects –denoted by C. Each algorithm is referred to as XYmethod, where X,Y$\in$ {T,C}. For example, the TCSoundex algorithm refers to the soundex algorithm that matches the words of the title of the source document with the words that appear in the title and the content of every target document. Figure 1.10 illustrates different results produced by various matching algorithms. The scope of the algorithms was the title of documents and the initial referenced document was titled *the smart frogs*. The basic search (TTBasic) filtered out small words and returned the initial document and one relative. The soundex algorithm (TTSoundex) matched the words *frog* and *frogs* and returned more documents. The third algorithm

Table 1.1. Advantages and disadvantages of matching algorithms

Algorithm	Advantages	Disadvantages
TTBasic	Very Fast High probability of correct prediction	Overlooks many relative objects Small anagrams can misguide it
TTSoundex	Very Fast High probability of correct prediction Small anagrams do not misguide it	Overlooks many relative objects Noise can produce false matches
TTMetaphone	High probability of correct prediction Small anagrams do not misguide it Grammatical rules reduce noise	Overlooks many relative objects Noise can produce false matches
TTLevenshtein	Objects are implicitly compared on their size and their type	Can be misguided by continuous not related letters Slower than other methods
TCBasic	Fast More potential matchers	Big documents have higher probability to considered relatives (dimensionality)
TCSoundex	Fast More potential matchers than TCBasic	Dimensionality and increased noise produce many false matches
TCMetaphone	Fast More potential matchers than TCBasic Grammatical rules reduce noise	Dimensionality and increased noise produce many false matches
CCBasic	Many potential matchers Good when the title is not representative of the object	Dimensionality problem in a superlative degree
CCSoundex	Huge number of potential matchers (more than CCBasic)	The dimensionality problem combined with the noise often produces bad results
CCMetaphone	Huge number of potential matchers (more than CCBasic)	The dimensionality problem combined with the noise often produces bad results
CCLevenshtein	Content is implicitly compared on its size and type	Slow Vague advantages can also be disadvantages

(TTLevenshtein) matched and ordered documents according to their pattern distance from the initial document. The numbers reported by the first two algorithms represent the matched words between documents, while the numbers reported by the third algorithm represent the similarity between patterns. Increased levels of fuzziness usually produce more results, but less accurate ones (for example the last returned document of the third algorithm).

Figure 1.11 illustrates the results produced when we set the target scope to a combination of title and content. The initial referenced document was again the one titled *the smart frogs*. The basic search (TCBasic) now traversed also the content of documents and returned documents that contained the words *smart* or *frogs*. The soundex algorithm (TCSoundex) returned more documents because it also matched similar sounding words inside content. However, increased level of noise produced some clearly irrelevant documents. Furthermore, bigger documents were found to be more relevant to our search (the dimensionality problem). Take, for example, the first document returned that was titled *the Scientist and the Frog*. This is a rather big joke with many occurrences of word *frog*. This document was considered more relative to our search than even our initial referenced document which ranked third! The third algorithm (CCLevenshtein) returned documents that have similar characteristics. For example most returned documents contain some form of dialogue and/or were approximately of the same size. Should our intension be to find jokes of similar type rather than jokes with similar content, then this algorithm would produce the most accurate results.

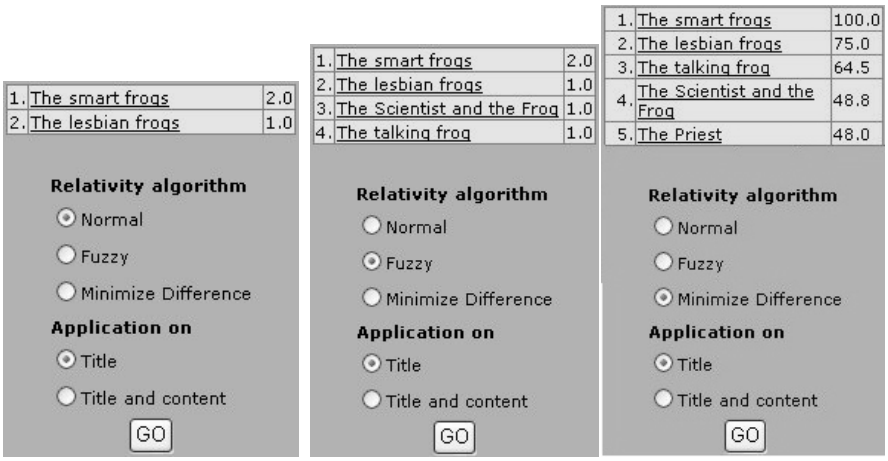

Fig. 1.10. Document matching techniques where source and target is set to title

More on matching relative documents. One of our main goals was to reduce the complexity of the community from the end user perspective. This

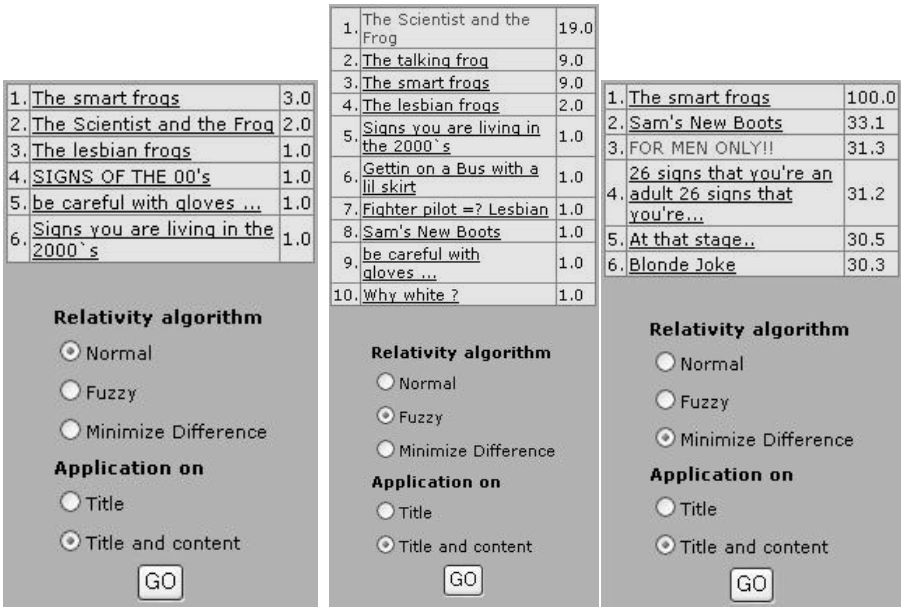

Fig. 1.11. Document matching techniques where target is set to title+content

directed that we should somehow prevent all the algorithmic details from being revealed to the end user. The ideal would be to produce a high quality set of representative documents with minimum user disturbance like most current search engines do.

Take for example Google (`http://www.google.com`), which uses an efficient algorithm based on hyperlink structure analysis to produce quick and accurate results. The algorithm relies on the presence of hard-coded links between documents and does not require any feedback or tuning from the end-user in order to work. Those hard-coded links have been proved indicative of the quality of documents, so when good ranked pages point to some other page then this page gets a higher rank. This technique cannot be used in our community because we do not have hard-coded links between documents[3]. However, due to the fact that all documents have been ranked by end-users we already have an inner ranking of the importance of documents. Thus, a good first step would be to sort the returning set of matched documents based also on their average ranking.

Still, we have many potential matchers. In order to produce just one set of really relevant returning documents we have somehow to aggregate the results obtained from several heuristics. A good idea is to get all the, relative,

[3] Soft-links could be established by identifying high quality keywords inside documents that could be treated as hyperlinks. Several algorithms exist today that aim at identifying such keywords, for example see [1.7].

fast algorithms and sort them according to how strict they are. For example a document that is returned by TTBasic is almost certain that would be a relative document. Then, we have TTSoundex and TTMetaphone that incorporate more noise, but return more documents. TCBasic follows with application over a different scope, and finally TTLevenshtein that incorporates a faddish fuzziness.

For the time being we conduct experiments on the details of aggregated models. A prevailing idea is to get the aggregated set of documents that exist on the output of at least two of the above matchers. Then, we can sort them using the average vote they received from users. However, such models have yet to prove their value while at the same time their simultaneous usage is expensive.

1.4 Conclusions

MyFreddy is a new, fun community with rich content. Its philosophy encourages individuals to submit content and thus, the maintenance cost is limited. The community has been active since March 2001. Since then it has been visited by over 30000 users and has collected over 350000 thousand votes. Currently, it contains more than 3000 voting objects submitted by registered users in 16 categories (English and Greek versions).

We feel that myFreddy can be used as a test-bed for rapid testing of new personalization ideas and algorithms. In addition, the voting and object database is a very rich, real-world data-pool that can facilitate the research on the fields of data-mining and recommendation systems. We are planning to offer large parts of the collected data to interested researchers for free.

During the design phase of the community we have spotted several problems and developed several solutions for building working intelligent applications over the Web. We have also researched how advanced techniques can be used to pull-up personalized data with limited user disturbance under a very demanding environment.

References

1.1 M. Balabanovic and Y. Shoham. Learning Information Retrieval Agents: Experiments with Automated Web Browsing. In *AAAI 1995 Spring Symp. Information Gathering from Heterogeneous, Distributed Environments*, AAAI Press, 1995.

1.2 M. Balabanovic and Y. Shoham. Fab: Content-Based, Collaborative Recommendation. *Comm. of the ACM*, 40:3 (1997), pp. 66-70.

1.3 K. Bollacker, S. Lawrence, and L. Giles. CiteSeer: An Autonomous System for UMIACS Processing and Organizing Scientific Literature on the Web. Working Notes of Learning from Text and the Web, In *Conf. Automated Learning and Discovery* – CONALD'98, Carnegie Mellon Univ., Pittsburgh, 1998. `http://www.cs.cmu.edu/~conald/conald.shtml`.

1.4 M. Claypool, D. Brown, L. Phong, and M. Waseda. Inferring User Interests. *IEEE Internet Computing*, 5:6 (2001), pp. 32-39.
1.5 Cookies Specification: Netscape. Available at: `http://home.netscape.com/newsref/std/cookiespec.html`
1.6 O. de Vel and S.A. Nesbitt. Collaborative Filtering Agent System for Dynamic Virtual Communities on the Web. Working Notes of Learning from Text and the Web, In *Conf. Automated Learning and Discovery* – CONALD'98), Carnegie Mellon Univ., Pittsburgh, 1998. `http://www.cs.cmu.edu/~conald/conald.shtml`.
1.7 R. Feldman, M. Fresko, Y. Kinar, Y. Lindell, O. Liphstat, M. Rajman, Y. Schler, and O. Zamir. Text mining at the term level. In *Proc. 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, 1998, pp. 65-73.
1.8 C.V. Goldman, A. Langer, and J.S. Rosenschein. Musag: An Agent That Learns What You Mean. *Applied AI*, 11:5 (1997), pp. 413-435.
1.9 T. Kamba, H. Sakagami, and Y. Koseki. Anatagonomy: A Personalized Newspaper on the World Wide Web. *Int'l J. Human-Computer Studies*, 46:6 (1997), pp. 789-803.
1.10 D. Knuth. *The art of computer programming, Vol. 3: sorting and searching.* Addison-Wesley, 1973.
1.11 J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, and J. Riedl. GroupLense: Applying Filtering to Usenet News. *Comm. of the ACM*, 40:3 (1997), pp. 77-87.
1.12 B. LaMacchia. Internet Fish, A Revised Version of a Thesis Proposal. MIT, AI Lab and Dept. of Electrical Eng. and Computer Science, Cambridge, Mass., 1996.
1.13 K. Lang. News Weeder: Learning to Filter Netnews. In *Proc. 12th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1995, pp. 331-339.
1.14 P. Lawrence. Hanging on the Metaphone. *Computer Language*, 7:12 (1990), pp. 39-43.
1.15 I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics – Doklady* 10 (1966), pp. 707-710.
1.16 H. Lieberman. Letizia: An Agent that Assists Web Browsing. In *Proc. 14th Int'l Joint Conf. AI* – IJCAI'95, AAAI Press, 1995, pp. 924-929.
1.17 P. Maes. Agents that Reduce Work and Information Overload. *Comm. of the ACM*, 37:7 (1994), pp. 30-40.
1.18 T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
1.19 M. Pazzani, J. Muramatsu, and D. Billsus. Syskill and Webert: Identifying Interesting Web Sites. Inn *Proc. 13th Nat'l Conf. AI* – AAAI'96, AAAI Press, 1996, pp. 54-61.
1.20 J. Shavlik and T. Eliassi-Rad. Building Intelligent Agents for Web-based Tasks: A Theory-Refinement Approach. Working Notes of Learning from Text and the Web, In *Conf. on Automated Learning and Discovery* – CONALD'98, Carnegie Mellon Univ., Pittsburgh, 1998. `http://www.cs.cmu.edu/~conald/conald.shtml`.
1.21 J. Rucker and J.P. Marcos. Siteseer: Personalized Navigation for the Web. *Comm. of the ACM*, 40:3 (1997), pp. 73-75.
1.22 T. Terveen, W. Hill, D. McDonald, and J. Creter. PHOAKS: A System for Sharing Recommendations. *Comm. of the ACM*, 40:3 (1997), pp. 59-62.
1.23 M. Zari, H. Saiedian, and M. Naeem. Understanding and Reducing Web Delays. *IEEE Computer*, 34:12 (2001), pp. 30-37.