3

An Internet Portal based on 'Twenty Questions'

B.W. van de Fliert, G. Meinsma, T.P.P. Visser, M.A. Peletier

Abstract

An efficient Internet portal should contain a search engine or maybe even a decision support system to supply the user with the information (s)he may be looking for. In this report an intelligent agent is suggested that relates different sites to each other, based on the answers supplied by the users looking for certain information. For this purpose a self-learning system has been made, based on the neural network of the game Twenty Questions [3], but with a strategy that relates different objects or sites by correlating the list of answers to the questions.

Keywords

Internet, portal, search engine.

3.1 Introduction

3.1.1 Internet search facilitators

The Internet presents us with an intriguing paradox: while seeking information is one of the principal reasons for using it, actually finding information can be extremely difficult. The current growth rate of the Internet suggests that this gap between information requests and information retrieval will widen further in the near future.

Observation of individual users who are efficient information retrievers shows a tendency towards creating a personalized structure that overlays the Internet. 'Bookmarks' or 'Favorites' are a common and effective aid in this process. The resulting personalized structure is often efficient in dealing with information requests that correspond to the user's personal interest, but offers little help on other topics. One can recognize here a self-enhancing effect: domains of personal interest generate frequent information requests, which have the by-effect of maintaining and expanding the personalized structure, thus rendering the structure even more effective in these domains. Topics outside of the personal areas of interest, or topics that by nature arise less frequently, lack this positive feedback loop. The structure thus optimizes itself on domains of personal interest, at the expense of others. For information requests that stray from the well-trodden path users tend to apply a catch-all or 'one site fits all' strategy: they strongly favor sites that are extremely broad in nature and offer effective search capabilities or an intuitive categorization.

Search facilitators exist in a variety of types and structures:

- The classical string search engines (Altavista, Hotbot, Ilse, Google, and many others) gather link information in an automated manner and provide access by a search interface.
- Systems based on categories (Yahoo) are maintained by humans, often assisted by robots and text analyzers.
- Search robots (e.g. Lexibot) are stand alone programs that screen the information that is available by a large set of search engines, and only offer to the user what is deemed appropriate. The user has control over how to filter information and how to configure searches.
- Find engines (e.g. Nano, Flyswat, Atomica also known as Gurunet) offer the user the possibility to directly retrieve information about words and phrases that appear in text documents such as Word documents. Upon request a list of relevant sites related to the phrase is supplied to the user; this may be real-time information or reference sites. It may select the most popular topic related to the keyword, depending on number of hyperlinks to this site or number of visits to this site.

3.1.2 The question presented by Skopos

The company Skopos aims to create an Internet portal that concentrates on information associated with the province of Overijssel; this will comprise foremost local government, but also commerce, local business, tourist information, and non-profit organizations (e.g. schools, societies). The portal is aimed towards the lay user, with relatively little knowledge of Internet in general and search systems in particular.

The main requirements come from two sides. First, the system should be easy to maintain; in particular, adding information from existing or new information sources should be straightforward and should not impact the running system. If possible, this process should even be automatic, with the information sources delivering their data in a pre-defined format that can automatically be processed.

Secondly, the interface presented to the Internet users should guide them through a decision procedure in an efficient manner. This decision procedure is an essential part of the design, both because of the lack of familiarity of the users and because of the often vague notion that users have of their own needs. A decision process helps users in defining their needs in an accurate manner.

3.1.3 Outline

In this report we mainly consider the decision procedure, and only now and then do we hint at the problem of maintenance. We begin with a brief summary of two examples and their connection with our problem. The bulk of this report is Section 3.3 where we consider our approach to the Skopos problem, based on the two examples. It results in an explicit algorithm that is ready for implementation.

3.2 Two examples

3.2.1 Twenty Questions

'Twenty questions' is a common game in Anglo-Saxon countries in which one player tries to guess an object that the other player is thinking of. In the original game, questions can only be answered by 'Yes', 'No', and (in some versions) 'Irrelevant'. At `www.20q.net` [3] an implementation of this game is presented in which the system asks the questions and the Web user answers them.

The rules of the game at [3] are slightly different from the classical form: the system grants itself thirty questions rather than twenty, and the range of possible answers has been extended to 'Yes', 'No', 'Irrelevant', 'Unknown', 'Sometimes', 'Maybe', 'Probably', 'Doubtful', 'Usually', 'Depends', 'Rarely', and 'Partly'. After finishing a game, the system queries the user about new information and apparent inconsistencies, and updates its 'knowledge'.

The system implements a simple neural network that is created and maintained on the basis of the game play and the final feedback (more on the specific structure of this network below). It is based on a collection of 'objects' that the system 'knows about' at that moment (i.e. the collection of real-life objects in the case of [3], and the collection of Internet resources in the case of the Skopos portal) and a collection of 'questions' for which the answers relate the objects in the network. In this report we describe how the ideas of [3] can be adapted to create a portal of the type that Skopos is seeking. The reasons for this choice of a neural network are the following:

1. Both the collection of objects and the collection of questions might be very large. Human linking of such collections is costly, and automatic linking

(based on textual analysis, for instance) is prone to natural-language mistakes (imagine for instance the various meanings of the word 'bar').

2. The collection of objects is difficult to define; it is expected to change over time, possibly very rapidly, and certainly in unpredictable ways.

3. The relationship between questions and objects, in the case of the Skopos portal, may change over time, for example due to merging of local governments.

4. A neural network is similar in concept to the personalized structure that individual users develop (see the introduction). An individual user can only maintain a highly focused structure in an effective way; a system based on a neural net, however, has the potential to harness the experience and knowledge of a large group of users and create a structure that is not only detailed but also broad.

In such a dynamic setting a self-maintaining system such as a neural network can provide a cost-effective maintenance system.

3.2.2 Intelligent agents

The ranking methods employed by search engines are not unlike that of `www.20q.net` in that also search engines keep track of matrices of data that connect terms (questions) with links (objects). At present the main difference is that search engines are not self learning; they are static.

Search engines commonly order their hits by statistical relevance, either based on the number of appearances of a specific term in a file, or the number of hyperlinks to the specific site. The metasearches can make an ordering based on several criteria, since they screen the hits of other search engines that may use different search methods.

For such a statistical method the search engine typically has available the *term frequency* t^{f}_{ij} which is the number of appearances of term t_i in document d_j, and the *normalized term frequency* t^{nf}_{ij} which equals t^{f}_{ij}/T_j where T_j is the total number of terms in document d_j. In addition it uses the *document frequency* d^{f}_i which is the number of documents in which term t_i appears, and uses the *normalized document frequency* $d^{\mathrm{nf}}_i = d^{\mathrm{f}}_i/D$ where D is number of documents in the database.

Ranking may now be done as follows. For a query consisting of a set of terms $\{t_k\}$, calculate for each term t_k in this set the ranking contribution in document d_j, that is, $c_{kj} = t^{\mathrm{nf}}_{kj}/d^{\mathrm{nf}}_k$. Now $\sum_k c_{kj}$ gives per document d_j a ranking number for the query. Of these a top n are shown to the user and the user is asked to provide feedback. By taking a weighted sum of the c_{kj} values, the influence of the different terms in the query on the ranking may be changed; the weights may be adapted related to the feedback from the user. Yet another way of influencing the ranking is by changing the normalization. Many variations along this line exist.

3.3 A decision procedure for the Skopos problem

The idea is to develop a decision procedure that acts as an intelligent 'counter-clerk'. Imagine going to a bookshop. The most popular books are displayed on a separate shelf. If one would want one of these, one does not need any help of the clerk, but can immediately pick a book. If one is looking for a specific book, or maybe a certain type of book, without knowing already which one, one addresses the counter clerk. This clerk (we assume) knows all the books in the store and will be able to help the customer by listening to the needs of this person ('I want a book about horses') and by asking questions to further specify the book ('Do you want a book with stories about horses or a book on how to take care of a horse?'). Having defined more or less what the person is looking for, the clerk might give alternatives, books that are similar.

We aim to formalize the Skopos Portal problem in a similar fashion. It is important to help most users (the 'common citizen') as soon as possible. This means making the popular sites easily accessible, at the cost of hiding those objects that are hardly ever sought for.

At each moment there will be an ordering of objects, based on 'how probable' it is that the user is interested in that specific object. For this we will define a 'popularity index'. The most popular items will be immediately accessible (like the popular books). If the user does not make a selection of these, the system will address the user with a question or will wait for a keyword. Based on this question or keyword the user's popularity index will be adjusted.

3.3.1 Question versus keyword

In the following we will only discuss the handling of questions. We can handle an entered keyword as answering a very specific question. Entering the keyword 'taxes' can be thought of as answering the question 'Are you looking for information concerning taxes?' with 'yes'.

We will also deal only with questions that can be answered by 'yes' or 'no'. This does not mean that questions like 'Where do you live?' are not allowed, but just that the answer 'Enschede' will be interpreted as answering 'yes' to the question 'Do you live in Enschede' (and possibly also answering 'no' to the question 'Do you live in Hengelo').

3.3.2 Connecting objects to questions

We consider a single-layered database of files or sites, called *objects*. During operation the answers of the questions that are asked to the many users are stored in two matrices Y and N. The columns of these matrices correspond to the objects and the rows correspond to the questions, see Figure 3.1. The Y_{mk} denotes the number of times that a user has answered 'yes' to question m while he was looking for object k. Similarly the matrix N keeps track of the no-answers.

	Horse	Shrimp	Key
Is it alive?	10	10	0
Can you carry it?	2	8	13
Is it a horse?	11	0	1

Figure 3.1: An example of a 3×3 matrix Y

These matrices provide information on the previous user requests. For instance, the column sum of the matrix $Y + N$, is related to the number of users that were looking for the object associated to that column (but weighed with the effort that was needed to find this object). This gives a method to initialize the 'popularity'; the higher the sum, the more popular the object is. The row sum of the matrix $Y - N$ gives information on how selective a question is. The correlation of columns in matrices Y and N automatically categorizes the available objects.

3.3.3 User initialization

Every new user is supplied with a vector p, which we will call the *popularity index*. It is a vector with as many elements as there are objects, and a large value for p_k is meant to indicate that it is very likely that this user is interested in object k. The values of these elements will be updated every time the user has answered a question.

The popularity index may be initialized as

$$p_k = \frac{\sum_m Y_{mk} + N_{mk}}{\sum_{m,k} Y_{mk} + N_{mk}}, \qquad \forall k. \tag{3.1}$$

This indicates the fraction of users that have used the system before, looking for object k. Initialization can of course also be done using specific group- or personal-profiles. Relatively large values of p_k means that object k was sought for many times.

3.3.4 Selecting a question

In order to determine as fast as possible the object the user is interested in, it is beneficial to select a question that roughly bisects the group of most popular objects. If we order the objects in popularity and consider only the n most popular objects, then we try to find a question for which the answer, is 'no' in 50% of the cases and 'yes' in the other 50%. The result is that, whatever the answer to the question is, about 50% of the objects will drop in popularity.

A first concern is to characterize the subset of questions that are selective enough; a measure of selectiveness of a question m is

$$q_m = \frac{\sum_k^n |Y_{mk} - N_{mk}|}{\sum_k^n Y_{mk} + N_{mk}}.$$

The smaller it is the less information a 'yes' or 'no' answer to that questions provides. The reasoning is as follows. The number $Y_{mk} - N_{mk}$ has information about how selective the question is for object k. For example, if this number is large, it is likely that someone who answers 'no' to question m is not interested in object k. The number $\sum_k Y_{mk} + N_{mk}$ has information about how many times the question m has been asked, and thus gives information about when $Y_{mk} - N_{mk}$ is 'large'. We consider a question m selective enough if, say,

$$q_m > 0.2 \max_j q_j.$$

This will also include questions that have not been asked frequently.

Now among this set of selective questions the question that best bisects the set of most popular objects is the m that minimizes

$$\frac{|\sum_k^n (Y_{mk} - N_{mk}) p_k|}{\sum_k^n Y_{mk} + N_{mk}}. \tag{3.2}$$

In the case of equal popularity $p_k = p_j$ the expression (3.2) is zero precisely if the total number of yes-answers to question m equals the total number of no-answers question m. That means that question m is optimally selective. Any deviation in (3.2) from zero means a non equal distribution of yes-no-answers, which is less preferable. The inclusion of the factor p_k accounts for the presumed popularity of the object.

3.3.5 Answering a question

Now suppose that the user, when prompted with the selected question m, answers 'yes'. The popularity index p is then updated for this user, depending on the relation between objects and question, by $(\forall k)$

$$\begin{aligned} p_k &:= p_k + w_m \frac{Y_{mk} - N_{mk}}{\sum_j Y_{mj} + N_{mj}}, && \text{if } Y_{mk} - N_{mk} \geq 0, \\ p_k &:= p_k / (1 + w_m \frac{Y_{mk} - N_{mk}}{\sum_j Y_{mj} + N_{mj}}), && \text{if } Y_{mk} - N_{mk} \leq 0. \end{aligned} \tag{3.3}$$

Here w_m is a weighing factor, defined as

$$w_m = 1 + |\sum_j \text{sgn}(Y_{mj} - N_{mj})|. \tag{3.4}$$

This update of the popularity for this user is motivated by the following. The weight w_m is chosen for the selective character of the question, discriminating between different objects. A straightforward example: the question 'are you interested in object number j' is more selective than 'are you interested in the category of objects $j \in \mathcal{J}$'. In the first case $Y_{mk} - N_{mk}$ is probably positive only for $k = j$ and in the second case $Y_{mk} - N_{mk}$ generally is positive for every $k \in \mathcal{J}$. The expression for w_m in (3.4) measures the impact that the answer should have; the more impact we want the answer to have the larger it is. Note that (3.4) is one possible way to capture this; others weights with the same purpose exist and for practical implementations w_m may require tuning.

The numerator $Y_{mk} - N_{mk}$ in (3.3) is chosen for the selective character for this specific object k . If half of the users say 'yes' to this question and the other half say 'no', the answer of the user does not give any information about this object, and hence p_k remains unchanged. Finally the denominator in (3.3) provides a normalization for the number of times this question has been asked.

Completely similarly, if the answer of the user was 'no', p_k is updated with the roles of N and Y reversed.

Now that p is updated we may select the next question and repeat the procedure until the object is found. In practice a shortlist of probable objects will be presented to the user after each question, so that the object can be selected as soon as it appears in the shortlist.

3.3.6 Object found

Three things need to be done when the object is found. The first is to present the object to the user. Then the system should check on related relevant objects. This is done by checking the correlation between the columns in the matrix; if this correlation is high, then the user may also be interested in this related object. Then finally, the matrices Y and N must be updated for the self-learning effect. This entails an update of the kth column only, where k is the number of the object that was found. Practically this means adding to a component of the Y matrix in case of a yes-answer, a number one over the number of questions asked to this user. The update automatically increases the popularity p_k of the object for when the next user arrives.

In order to keep a realistic value of the popularity indices, it may be desirable to let the matrices age, for example by multiplying the matrix elements periodically with a forgetting factor. This should allow the system to forget what was very popular at a specific time, such as the files for the Clinton-Lewinsky trial, or in the Overijssel case, the Enschede fireworks disaster. By using a forgetting factor, small updates of the popularity are more easily recognized.

3.3.7 Database initialization

To initialize the matrices Y and N one could take all elements in the matrices equal to one,

$$Y_{mk} = N_{mk} = 1, \qquad \forall m, k. \tag{3.5}$$

It is important that all matrix elements are positive initially, in order to recognize all the objects and questions. Since it will take a long time before the system will really react in a sophisticated manner, this initialization should be avoided whenever possible. It may be quite easy to get information about the popularity, based on previous experience, and initialize the Y and N accordingly. This works best in combination with a forgetting factor.

3.3.8 Adding objects and questions

The easiest way to replace or to add objects, is by copying the column of a related object. In time, the matrices will update themselves for the new object.

If there are no related objects, the columns of Y and N may be initialized with 1's in every entry of the column, but preferably some prior knowledge is used to artificially increase several entries. Plugging objects through artificial increase of column entries can also be a useful tool to make certain objects popular in a short time, for example for an event or after a disaster.

If two rows strongly correlate (in both Y and N) then the corresponding questions may be similar and possibly one of the two may be removed without affecting the performance of the system. The detection of such pairs of rows may be automated. What also may be automated is the need for adding questions. Indeed if an object exists whose corresponding column of $Y - N$ is small in some weighted sense, then this points out that no question exists that may lead the user to this object. This necessitates adding a question that does lead to this object.

3.4 Concluding remarks

Based on the game 'Twenty Questions' a decision procedure was proposed that brings users quickly at the site (object) of interest. The procedure was tailored to the Skopos Portal problem, for which it could not be assumed that users have a clear object in mind when they start the search.

The procedure does not rely on stochastic models and this may be a draw-back. The procedure has been tested and it appeared to work fairly well, but the scope of the test was quite limited. Large scale implementations will require tuning of several of the submodels. In particular the impact weight w_m that has been proposed here is just one of the many choices and more appropriate choices may exist.

Acknowledgment

This report is based on discussions between the authors, Herman Ellenbroek, Tjerk Vonck, Jan Schut, Jaap Molenaar and Matti Heiliö.

Bibliography

[1] C. Sprangers. *Intermediair* **38**. September 2000.

[2] M. Szu-Whitney and G. Whitney. *Portals and Corridors: A Visionary Guide to Hyperspace*. Frog Limited, 2000.

[3] `www.20q.net`

[4] `www.atomica.com` (Formerly known as gurunet.)

[5] `www.flyswat.com`

[6] `www.lexibot.com`