Content and Type as Orthogonal Modeling Features: a Study on User Interest Awareness in Entity Subscription Services

George Giannakopoulos, Themis Palpanas
DISI
University of Trento
Trento, Italy
ggianna@disi.unitn.it, themis@disi.unitn.eu

***Abstract*—Real-word entities can be mapped to unique entity identifiers through an Entity Name System (ENS), to systematically support the re-use of these identifiers and disambiguate references to real world entities in the Web. An entity subscription service informs subscribed users of changes in the descriptive data of an entity, which is a set of free-form attribute name-value pairs. We study the design, implementation and application of an adaptable, push-policy subscription service, within a large-scale ENS. The subscription system aims to deliver ranked descriptions of the changes on entities, following user preferences through a feedback-driven adaptation process. Within this paper, we offer a novel approach based on the discrimination between the content (descriptive aspect) and the type (directly quantifiable or binary aspects) of information instances (i.e., entity changes in our case). We study and evaluate two different approaches of adaptation to user interests: one that only manages constant user preferences and one that adapts to interest shifts of the users. We evaluate the learning curve of the variations of the system, the utility of the content-type discrimination, as well as the effectiveness of modeling user behavior for random interest shifts of the user. The experiments demonstrate good results, especially in the system's content-aware adaptation aspect, which takes into account user interest shifts. We also extract a set of useful conclusions concerning the detection of user interest shifts, based on feedback, as well as the relation between the user interest shift frequency and the optimality of learning memory window size.**

I. Introduction

An Entity Name System (ENS) [2], [3], [4] is a system that aims to handle the process of assigning and managing identifiers for entities (e.g., people, locations) in the World Wide Web (WWW). These identifiers are global, with the purpose of consistently identifying a specific entity across system boundaries, regardless of the place in which references to this entity may appear.

In the *ENS* we are interested in the minimal amount of information based on which we are able to uniquely identify each entity in the system. This information is imported both automatically or manually to the system, but its update is based mostly on collaborative effort — which is also apparent, e.g., in Wikipedia.

The volume and volatility of the data contained within the *ENS* offer a challenge concerning both the maintenance and accessibility of the information. The maintenance of the entity data would be a rather expensive and cumbersome task, was it not designed as a joined effort with the *ENS* user-community. By relying on the collective wisdom and collaboration of users, we aim to induce and use their feedback in order to achieve high data quality. The premise is that users will be willing to "adopt" some entities, and make sure that their descriptions in the *ENS* are always accurate.

In support of this functionality, we are proposing and describing the adaptive aspect of an *Adaptive Entity Subscription System*, which allows a user to *subscribe* to entities of interest (e.g., one's own entity, or entity of one's home town) and get informed about *changes* on these entities. The subscription system helps clients follow the changes of entities in the *ENS* at the time they occur through asynchronous messages (e.g., via e-mail).

The personalized aspect of AESS takes into account what kinds of change are interesting for a given subscription client. The system ranks the information on changes in every sent message, according to user preferences. It further adapts to the individual use-patterns of each client and to user interest shifts, based on the information of the feedback. The aim is that the information flow to the user remains aligned to user interest. This kind of personalization provides ease of use and enhances the user's experience and interaction with the system, thus facilitating the update of and access to *ENS* data.

In the following paragraphs we elaborate on ENS and the subscription service notions to define clearly the setting of our study.

A. The Entity Name System

The *ENS* has a (distributed and replicated) repository for storing entity identifiers along with some small amount of descriptive information for each entity. The purpose of storing this information is to use it for discriminating among entities, not exhaustively describing them. Entities are described by a number of attribute-value pairs, where the attribute names and the potential values are user-defined (arbitrary) strings.

The ENS supports, through an *Access Services* layer, the search for an entity or the update of entities. The search allows the ENS clients to identify an entity based on cross-validated and updated data. Updates on the entities of the ENS repository can be performed either by inserting a new entity in the system or by changing some of the attributes of an existing entity. System administrators can even merge, split or delete entities.

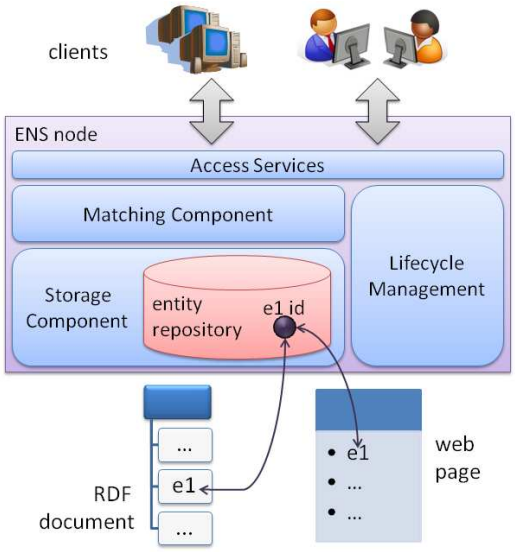


Figure 1. Schematic of the *ENS* and its interactions.

As shown in Figure 1, the end result is that all instances of the same entity (i.e., mentioned in different systems, ontologies, web pages, etc.) are assigned the same identifier. Therefore, joining these documents and merging their information becomes a much more simple and effective process than before.

Within this context, a client can be interested in receiving information on *changes* on the identifying information of a specific entity in the ENS. To this end, a client can *subscribe* to entities that are of some interest and get informed about changes on those entities. This subscription-based information service is described in the next paragraphs.

B. The Subscription Service

The administration and checking of entity information is a difficult task in itself, considering the scalability of an ENS. To alleviate the burden of such a task and to handle the information need of non-administration clients, we propose the use of a *change subscription system*.

The change subscription system helps clients follow the changes of entities in the ENS in the time they occur. The subscription system informs users through asynchronous messages (e.g., via e-mail) on sets of changes concerning entities the users have selected. Examples of subscription include subscribing to changes of the "current population" attribute of one's home town, or of the "phone" attribute of a former colleague.

The system we propose ranks the information on changes in the sent message according to user preferences on what is important, and is able to adapt to the individual use-patterns of each client. This ranking is based on a user model, which is created through machine learning and a feedback-based user modeling methodology. The system we propose further adapts to user interest shifts, based on the information of the feedback.

Within this work, we further study the relation between size of the system's memory window — which takes into account the last k instances of user feedback to create the user model — and the performance of the system. We conclude that:

- The performance of the system is highly correlated to the average frequency of user interest shifts.
- There exists a near-optimal operation range for the window memory size parameter, which is computed based on the estimated interest shift frequency.

The experimental results show that the proposed approach can successfully adapt to usage patterns, delivering to each user change notifications ranked according to one's interest. Note that we use ENS as a concrete example for ease of exposition, but our techniques can be applied to any adaptation process, where there are data that are hybrid in nature, i.e., they can be represented by a vector *and* a string representation of content.

In the following sections, we present related work (Section II), we define the problem tackled herein (Section III) and elaborate on our proposed methodology (Section III-A). Then, we present the performed experiments (Section V), followed by related work (Section II). We then conclude and present future work (Section VI).

II. Related Work

User-modeling has been used for several years (e.g., [5]) and applied to a variety of settings, from information retrieval [6] to filtering tasks [7], [8] and recommendations [9]. In this work, the information retrieved is defined a priori and is always sent to the user without any filtering, because the user is subscribed to specific entities. However, what is considered unimportant based on estimation of importance can easily be discarded, if needed, providing a filtering mechanism.

Feedback-based approaches have been proposed in information retrieval since the early 1970's [10]. These feedback methodologies have relied both on the vector space model [11], [6] or Bayesian modeling [12] to represent user needs. Our method is a content-based method, as far as the content vs. collaborative [13] perspective of the system is concerned. The novelty of our work lies in the fact that we express information instances taking into account their dual nature: *type* and *content*. For us, the *type* represents the

directly quantifiable (or simply binary) aspects of a piece of information, e.g., the number of words of a text, or whether a car is blue or not. The *type* of our instances, which are pieces of change information, is identified by vector features engineered for entity change subscription. The *content* represents the descriptive part of a piece of information, e.g., the name of a person, or the content of a web page. To represent what we define as content, we use an intermediate representation: the n-gram graph, which takes into account fuzzy matching of strings[1]. However, our methodology allows any kind of similarity metric to be applied between the content part of any two information instances. The function of similarity allows to add the *content* as a set of new dimensions to the feature space describing a change. Therefore, the representation of the user model is different in semantics from the representation used in most of the other approaches, because of the intermediate representation of the n-gram graphs.

Within this work we use incremental, explicit user feedback. Incremental feedback has been used in retrieval tasks [15] and in the context that we are applying it, which is actually a push-based information flow, it is the most appropriate approach. However, in our case there are no user queries to improve, like for example in [16], because the information delivered to the user is based on the changes of entities pre-selected by the user. There is also no relevant document set that we can use to expand our knowledge, in contrast to retrieval tasks. Therefore, we only take into account the user's history of feedback to corresponding instances. We take into account individual user needs and not "canonical" (i.e., group or community) user needs [5], [17].

Our adaptive subscription service aims to *rank* the information retrieved. Ranking has been in the user modeling community since the 90's [18] and has given rise to the recommender systems domain (see [13] for a survey). In this work, the ranking of information is treated as a *regression problem*, based on a mapping of qualitatively described user interest to real values (similar to the notion of utility [13]), so as to be genericly applicable. However, we do not search for a function $PREF(u, v)$ that indicates the certainty of whether u should be ranked before that v, as, e.g., is described in [18]: the ranking in our case comes as an emergent result of the regression method. Our approach evaluation takes into account cases where the user considers instances equally important, because we estimate error in a way that accounts for *acceptable error* in the regression. A base difference that exists between our system and recommender systems is that we only need to rank pieces of information that are designated by the subscription of the user; we do not recommend *new* information instances, relative to user preferences as is the case in some recommender systems [19], nor are we interested in *popular* recommendations, as in [20].

The importance of term coexistence as a model for user preferences in content filtering, as well as the ability of the system to adapt to user shifts, has also been studied through an analogy to the immune system in Nootopia [21]. There, a network (or graph) representation of terms, as well as their coexistence in interesting content, form the representation of what we consider the *content* of user preferences. The terms, represented as weighted nodes of the network, are connected through a weighted edge, if they appear in a given window of words within a text. The weights on the edges are a function of the frequency and distance of co-occurence of the connected terms. In Nootopia however, the interesting — non-interesting distinction is performed through a self-organizing model, that rejects unimportant pieces of information, which does not allow several levels of interest. Furthermore, the system uses whole words as the basic unit of analysis, whereas we use sub-word items. The advantage of the sub-word items is that one can decide on similarity between different word forms, without the need of preprocessing like stemming, lemmatization, etc. The process of checking against the preference graph in Nootopia is performed through a network activation process, which increases complexity, when compared to our Normalized Value Similarity algorithm. Finally, we take into account other aspects of the preference, represented as the *type* portion of the change and the final estimation of the graded interest is estimated in the vector space and not the space of graphs.

In this work, we also study the effect of *"memory" window*, i.e., the number of remembered feedback instances, to the performance of the system, also in the presence of *user interest shifts*. Contrary to [15], in the presence of user interest shift, taking into account only a limited number of feedback instances does better than using all available feedback data, because an interest shift can heavily alter the estimator function's efficacy. Our study indicates that there seems to be a steady functional relation that exists between the user expected interest shift and the memory window, in order to achieve near-optimal results. A highly related study, even in a similar task, has been performed in [22]. However, we do not use classes of documents to relate the user to class-based interest, as done in [22], and we do not use a binary decision of correctness as feedback. Furthermore, our information instances, i.e., changes, have some dimensions concerning their *type* and the model of the content is not based on bag-of-words. Finally, we do not study *how* to detect an interest shift, but the relation between an estimated interest shift frequency and the "memory" window. The detection of the interest shift can use any method from time-series analysis or any probabilistic model (e.g., Hidden Markov Model), but is not the focus of this work.

[1]The n-gram graphs can actually represent any type of information where neighbourhood can be defined between the features of a represented instance. For more details, see [14].

In the next section we formulate the problem of adaptive subscription as a problem of interest estimation, based on user-feedback.

III. Problem Formulation

The subscription service is a supplement to the usability of the ENS, tackling the problem of keeping consumers of information concerning specific entities updated. The system disseminates information to consumers[2] about changes on entities to which the consumers have subscribed.

The sending of information can be either synchronous, based on the time a change takes place, or asynchronous. Furthermore, the update can be sent for every individual change or for a set of changes, e.g., periodically per day. In our case we use an asynchronous model for the update, that takes place periodically and contains all the information the user has subscribed to. It should be noted that a user subscribes to an *entity*, which in turns implicitly defines the set of changes that will reach the user.

It is clear that a subscription service for changes on entities is not very different from any other kind of subscription service. Nevertheless, the *attributes* of the information on changes partially define the adaptive approach we present within this work (see Section IV-A). As we elaborate later in this article, changes can have various degrees of importance, based on their *type*. For example, a "delete" change can be more important than an "insert" change (also see Section IV-A). Then, the *content* of a change is a completely different aspect from its *type*. We consider the content of the change to be the difference between the attributes of the original state of an entity and the attributes of the new state. We argue that this *content* can provide more information than the *type* of the change alone (also see Section V).

A. Subscription Feedback and Adaptive Content Ranking

The problem we tackle within this work is bringing the information on changes to consumers, in such a way that will best suit individual consumers' needs. The main scenario we face wants a user-consumer to have subscribed to a set of entities from the ENS, in order to get informed about changes in these entities. However, each user may be interested in different kinds of changes. This problem needs an adaptable system to support the modeling and application of user needs and preferences. The system we propose is aware of user feedback in order to improve the users' experience and optimize the flow of information to each individual user.

The problem can be formalized as follows:
Given

- a set of users, i.e., user models $\mathbb{U}$
- a set of change descriptions $\mathbb{D} = \{< T, C >, T \in \mathbb{T}, C \in \mathbb{C}\}$ of type T and content C, where $\mathbb{T}$ is the set of possible types and $\mathbb{C}$ the set of possible contents
- a set of feedback indications $\mathbb{F} = \{< U, D, I >, U \in \mathbb{U}, D \in \mathbb{D}, I \in \mathbb{I}\}$, where $\mathbb{I}$ a set of importance indicators, either categorical (nominal-scale) or real

we need the system to optimize the estimation function of importance for a new description of a change D_n, $f(U, D_n|\mathbb{F}) : \mathbb{U} \times \mathbb{D} \rightarrow \mathbb{I}$, for our criterion J (described below) that aims at minimizing the following absolute difference: $|f(U, D_n|\mathbb{F}) - I_0(U, D_n)|$, where $I_0(U, D_n) \in \mathbb{I}$ is the user assigned interest value.

In this work the $\mathbb{I}$ set will be considered to be the set of real numbers, i.e., $\mathbb{I} \equiv \mathbb{R}$. This way we can provide a partial ordering of descriptions $D_i \in \mathbb{D}$. Our criterion J is based on whether the ordering of the set of descriptions a user is entitled to get (due to a subscription) is the one *expected by the user*. We do not consider that there is a unique ordering that will suit the user's needs. Instead, we measure the performance of the system based on a devised measure, called Ratio of Acceptable Errors (RAE), defined in Section V.

IV. Proposed Approach: Adaptive Subscription-Based Updates

The architecture we have devised consists of the *Change Queue*, the *User Profile Database*, the *Subscription Information Broker* and the *Adaptive Information Control* (see Figure 2).

The *Change Queue* is a change log or repository of changes retaining information on the time stamp of a change. The *User Profile Database*, matches a user-consumer to her profile and holds such information as the time stamp of the last update the user has received. The database can be distributed over several servers and a login-based approach allows mapping each user-consumer to a corresponding profile and the set of entities she has subscribed to. On the other hand, the sustem is designed in a way that there is no need to transfer profile information across different modules of the system, thus seriously reducing privacy concerns for this information.

The *Subscription Information Broker* takes change information and stores them into the *Change Queue*. The broker is also the component that sends the information to the consumers. To do that, the broker requests from the *Adaptive Information Control* component the set of data that need to be sent to active users, on a per user basis. Furthermore, the *Subscription Information Broker* reports the consumer feedback to the *Adaptive Information Control* when the consumer uses the feedback mechanism, to allow for adaptation to consumer needs.

The *Adaptive Information Control* matches each consumer to a user profile, requests from the *Change Queue* the

[2]The terms *client*, *user* and *consumer* will be used interchangeably throughout the text.

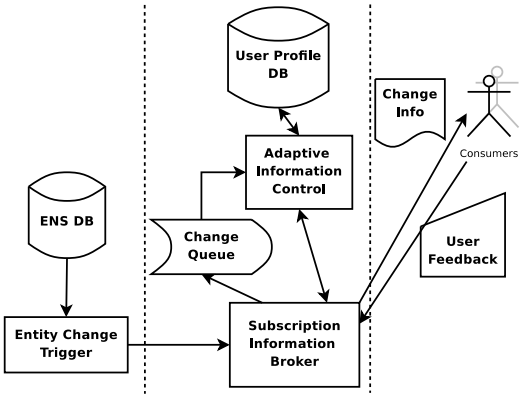


Figure 2. Schematic of the *Subscription Service* and its interactions.

changes that have occurred since the user's last update and ranks and groups the information that will be sent to the user. This information is then passed on to the *Subscription Information Broker* to be disseminated. The *Adaptive Information Control* also uses user feedback provided by the *Subscription Information Broker* to recompute and update the user profiles in the *User Profile Database*.

The asynchronous nature of the overall process, as well as the fact that only *Subscription Information Broker* interacts with external messages, allows for off-line and distributed processing, facilitating scalability. This is why no external access is granted to the subscription service database directly. Furthermore, the *Adaptive Information Control* module is a required medium between the feedback mechanism and the user model database, because of the required calculations for the update of the user model, especially as far as it concerns the content of a change (see Section IV-A2).

The adaptation of the aforementioned architecture to user needs relies on two main aspects of the *Adaptive Information Control*: the first is how changes' descriptions are represented to form a user model and the second is how the system adapts to user feedback.

A. Modeling Change and Information Importance

We define the description D of a change in a dual way, concerning the *type* and the *content* of the change. For each of these aspects we use a different kind of representation to address their individuality.

1) Type of a Change: The *type* T of a description can be viewed as a point in a multi-dimensional space, where the dimensions are:

- One dimension for each alternative of *entity change*: *deletion*, *splitting* of an an entity into two, *merging* of two entities in one, and *entity update*. A deletion change signals the removal of an entity from the ENS. A splitting change indicates the splitting of a single entity into two individual entities, when an entity is found to have been erroneously created with data from two different entities. A merging change describes the merging of two entities into a new one, when the original entities actually referred to the same real entity. An update indicates any change in the individual attributes of an entity. The value for each dimension can be either 1.0, indicating the corresponding type of change, or 0.0 otherwise.
  We have decided to use a different dimension for every change alternative, because enumerating the alternatives over a single real axis would imply a partial ordering between alternatives, which does not stand. An example case of a merging change will have the value 1.0 at the merging dimension and all the other alternatives' dimensions, i.e., deletion, splitting and update, set to zero.
- One dimension for each alternative of *attribute change*, which is the result of an *entity update* change: *deletion*, *insertion*, *update*. A deletion of an attribute means removing the attribute name-value pair as a whole from an entity. An insertion indicates the insertion of a new attribute name-value pair into an entity. An update is a change of a given attribute, using a new value. The update is further elaborated using the following dimension of *normality*.
- The normality of a type of change should indicate, in a quantitative manner, whether a given update appears to be expected. This normality can be judged by such processes as type checking for new values, or by whether a value holds similar qualities to other values for the same attribute. For example a grammar model for a given attribute can indicate whether the new value is normal or not. We use normality as a real value, normalized between 0.0 and 1.0, where 0.0 indicates maximum abnormality and 1.0 perfect normality.

We can now represent a type T of change in a vector space, but the content C of the change is a completely different challenge. In an ENS system, the state of an entity can be described as a set of attribute name-value pairs. Thus, C is determined as the difference between the two states.

Ideally, we want to be able to identify such preferences as "I am interested in changes that have to do with my entities' name or telephone number", or "I am interested about when the *inProduction* field of proteins has a value of *true*". To identify such preferences one must act on the string level and create a model for attribute names and attribute values that are of interest for the consumer. The string representation of the content is the attribute-value pair that was changed, in its changed version if such a pair is applicable to the change. Otherwise, the representation is an empty string.

2) Content of a Change: To model interesting attribute name-value pairs, we use the paradigm of character n-gram graphs [23], which can take into account substring matches and offer a set of operators that allow for an updatable model [14].

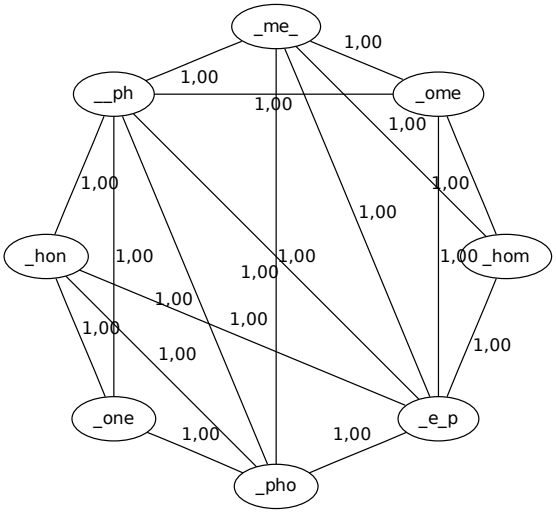


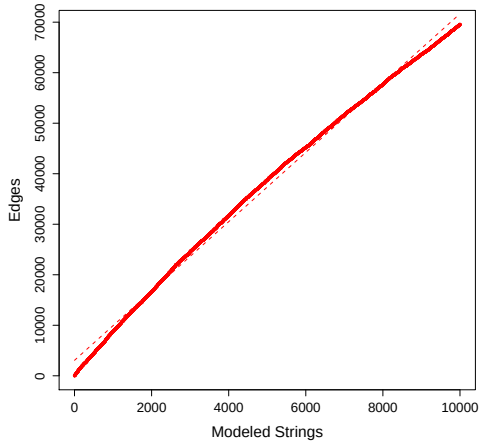
Figure 3. Sample N-gram Graph Representing the String "home_phone"

A *character* n-gram S^n contained in a text T can be any substring of length n of the original text. The *n-gram graph* is a graph $G = \{V^G, E^G, L, W\}$, where V^G is the set of vertexes, E^G is the set of edges, L is a function assigning a label to each vertex and edge, and W is a function assigning a weight to every edge. N-grams label the vertexes $v^G \in V^G$ of the graph. The (directed) edges are labeled by the concatenation of the labels of the vertexes they connect in the direction of the connection.

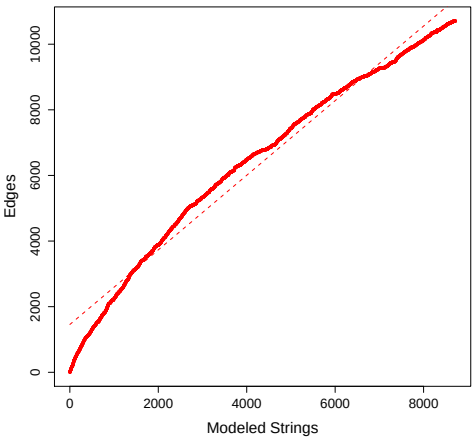
The graph is simply constructed by a running window over a given string, that analyzes the string into overlapping character n-grams and records information about which n-grams are neighbours within the window. The edges $e^G \in E^G$ (the superscript G will be omitted where easily assumed) connecting the n-grams indicate proximity of these n-grams in the text within a given window D_{win} of the original text [23]. The edges are weighted by measuring the number of co-occurrences of the vertexes' n-grams within D_{win}.

A graph sample, generated for 3-grams with a maximum neighbouring distance of 3 can be seen in Figure 3. In Figure 4a we also depict the growth of graph size, in terms of edge count, based on the number of represented *random* strings, which shows that the graph's edge count grows linearly to the number of modeled string instances. For *normal* text[3] the growth is more logarithmic than linear, as shown in Figure 4b. The logarithmic behaviour is more apparent when the vertices, i.e., possible n-grams, are used to indicate the growth of the n-gram graph (see Figures 5a, 5b).

[3]The text used was the first book of "The Book of the Aeneis", by Virgil, found online at http://www.ilt.columbia.edu/publications/Projects/digitexts/vergil/aeneid/book01.html.



(a) Random Strings - Edges

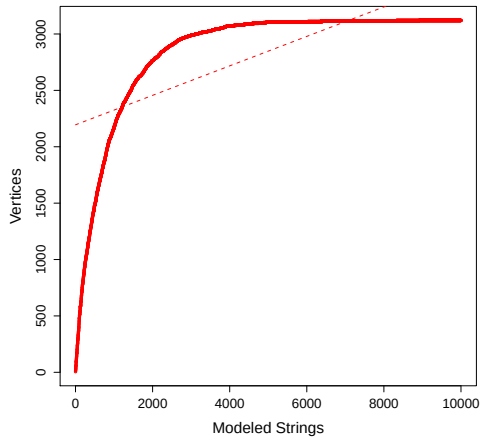


(b) Actual Strings - Edges

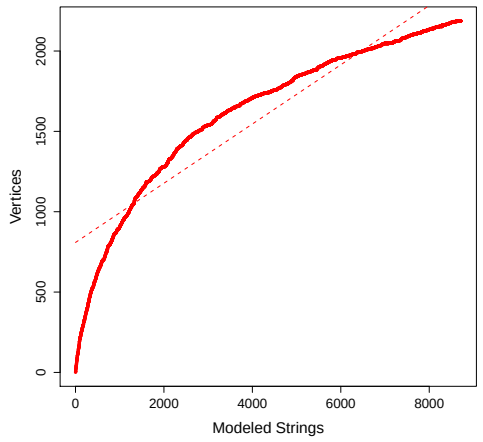
Figure 4. Edges of N-gram Graph Over the Number of Modeled Strings

We use the n-gram graphs within this work due to several of their traits like language neutrality and the fact that, when used for matching between strings, they offer a graded normalized indication of similarity. We also use the updatability of n-gram graphs, when applied as language models [14]. In other words, if we judge a set of attribute names-values as indicative of importance, we can create an n-gram graph that models the whole set and, thus, avoid keeping all the attribute names-values for matching. Furthermore, the model offers fuzzy matching and substring matching, which helps in open domains of attribute names and values, as is the case of an ENS.

To model the content C of changes a user is interested in, we create for each training instance C_i, given by the feedback process, a corresponding n-gram graph G_{C_i}. The



(a) Random Strings - Vertices



(b) Actual Strings - Vertices

Figure 5. Vertices of N-gram Graph Over the Number of Modeled Strings

graph is based on the string representation of the change.

The model graph construction process for each set of changes (e.g., of the uninteresting/interesting/critical classes of changes) comprises the initialization of a corresponding graph with the first string representation of content, and the subsequent update of this initial graph with the graphs of the other content instances in the class.

Specifically, given two graphs, G_1 and G_2, the first representing the training set of changes and the second a new instance, we create a single graph that represents the updated model graph G_1 with the graph of new evidence G_2:

$$update(G_1, G_2) \equiv G^u = (E^u, V^u, L, W^u) \qquad (1)$$

such that $E^u = E_1^G \cup E_2^G$, where E_1^G, E_2^G are the edge sets of G_1, G_2, respectively. In our implementation two edges are equal $e_1 = e_2$ when they have the same label, i.e., $L(e_1) = L(e_2)$.

The weights of the resulting graph's edges are calculated as follows: $W^i(e) = W^1(e) + (W^2(e) - W^1(e)) \times l$. The factor $l \in [0, 1]$ is called the learning factor: the higher the value of learning factor, the higher the impact of the second graph to the first graph. More precisely, a value of $l = 0$ indicates that the new graph will completely ignore the second graph. A value of $l = 1$ indicates that the weights of the edges of the first graph will be assigned to the weights of the resulting graph. As we need the model of a class to hold the average weights of all the individual graphs contributing to this model, the i-th graph that updates the class graph (model) uses a learning factor of $l = (1 - \frac{i-1}{i}), i > 1$. This creates a class model that acts as a representative graph for the class content instances.

As already indicated, we need to create more than one model graphs: one per feedback alternative of the user. Within this work we give the user three alternatives for feedback: unimportant information, important information and critical information (in order of increasing significance). Therefore, we create three corresponding model graphs. These graphs represent their corresponding content instances in the user model.

In our system the set of $\mathbb{I}$, which stands for importance, is the set of real numbers $\mathbb{R}$. We set three qualitatively mapped thresholds of importance: -1.0, which indicates unnecessary information, 1.0, which indicates useful information and 2.0, which indicates critical information. Of course, the set of importance alternatives could have as many elements as desired, keeping in mind that, if $\mathbb{I} \equiv \mathbb{R}$, using higher values for higher importance will probably provide better distinction. Given this kind of mapping, we need to be able to judge the importance $i \in \mathbb{I}$ of a new instance of changes, for a particular user.

To further clarify these definitions, we provide examples of changes that a user of the system could evaluate as:

- Critical: name deletion or garbaging names, deletion of default attributes of a type, e.g., deletion of the *longitude* and *latitude* of a location entity.
- Important: change of the longitude and latitude of a location entity with a new, normal value.
- Unimportant feedback: change in a field with a value that references some other ENS entity.

B. Ranking Using User Feedback

Having reported on the representation of changes, we can now describe the methodology for assigning importance values to entity changes, based on a user model.

The parts of the process that need to be described are

- What algorithm is used to *learn* which changes are important for a given user?

- What are the descriptive features that can be used to model a change?

The algorithm we use to learn the user model is actually that of Support Vector Machine Regression (SVR). SVMs have already been successfully used in a variety of applications and settings [24]. Within this work we use the LibSVM library [25] and especially its ϵ-SVR implementation of the algorithm found in [26].

The basic idea behind the ϵ-SVR is that, given a set of training data $\{(x_1, y_1), ..., (x_l, y_l)\} \in X \times R$,where X is the space of the input patterns we need to "find a function $f(x)$ that has at most" ϵ "deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible. In other words, we do not care about errors as long as they are less than" ϵ, " but will not accept any deviation larger than this" [27]. In our case, of change types T, $X \equiv \mathbb{R}^d$, which is the vector space we defined for the representation of types $T \in \mathbb{T}$, and $y_i \in \mathbb{I}$.

For the content C of changes, on the other hand, we first calculate the *size-normalized value similarity*[14] between the n-gram graph G_C of a judged C, with respect to each of the n-gram graphs of the user model $G_U^{important}, G_U^{unimportant}, G_U^{critical}$. This similarity value, which lies between 0.0 and 1.0, indicates what part of the graph of C can be found in the corresponding graphs of the model of the user. This set of similarities $\mathbb{S} = S^{unimportant}, S^{important}, S^{critical}$ is the second constituent of the representation of a description $D =< T, C >$ of a change with respect to a user model. To use this set of similarities, we integrate them within the vectors of the type as new dimensions-features. Therefore, when n-gram graphs are used, the overall importance of a change is estimated based on the *combined vector* for type *and* content in an extended input vector space X'. For an illustration of the overall process of using the graph models into the feature vector, also see Figure 6.

While in the original version of the system we created a single graph for both attribute names and attribute value modeling, experiments (see section V) indicated that these two models needed to be separated. Therefore in a second version of the adaptivity process, each level of importance uses two graphs to represent attribute names and attribute values.

C. Shift-Aware Adaptive Subscription System (SAAS)

The problem we have described so far does not take into account changes of user interest over time. However, in real world scenarios interest shifts occur. This means that a user does not have a single behavior over time, but that at random times she changes the target (i.e., desired) ranking of information.

To tackle the problem of shifting interest we decided to study the case where the system only remembers the feedback instances of the last k iterations. This means that

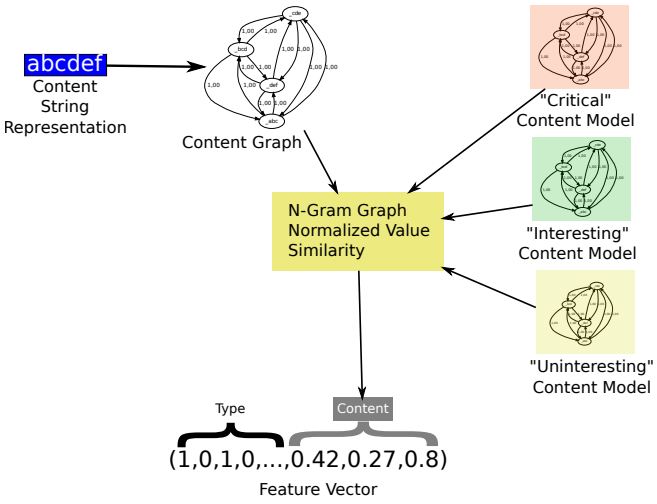


Figure 6. The Process of Content Features' Extraction

the system "forgets" everything before that. This introduces the notion of system *memory* to our approach. The system memory for iteration i is therefore defined as the set of feedback instances provided by the user within the time interval $[i - k, i - 1]$. This system memory describes the training set of the learning algorithm. The implementation simplicity of the system memory approach makes it easily integrated in a system like the ENS.

The full system that also takes into account interest shifts has been named *SAAS*.

V. Experiments and Evaluation

In order to evaluate our methodology, and given the fact that OKKAM is not yet at a production state, which means that users have been limited in number, we had to generate a synthetic set of entity changes, much like other works on adaptive systems (e.g., [28], [29]). In the following subsections we describe the data, the process and methodology of evaluation, as well as the results of our experiments.

A. Data

The information concerning user behavior is twofold in our case. We try to replicate the behavior of the users of the ENS who change entity data. Then, we create profiles for the behavior of subscribers using the adaptive subscription service. Correspondingly the data we generate are of the following two aspects:

- ENS User Behavior Data.
- Entity Subscribers' Feedback Data.

1) ENS User Behavior Data: As behavior of the users that change the entities' data, we generate a number of changes' descriptions D — 10000 instances split into sets of 1000 instances to provide for 10-fold validation. To generate this kind of dataset, we randomly create changes based on a

User type (Prob.)	*Change type*	*Probability*
Benevolent (0.95)	Attribute change (normal)	0.60
	Attribute insertion	0.30
	Attribute deletion	0.10
Sys.admin.(0.03)	Entity merge	0.45
	Entity split	0.45
	Entity deletion	0.10
Malevolent (0.02)	Attribute change (abnormal)	0.70
	Attribute deletion	0.30

Table I
ENS USER BEHAVIOR: PROBABILITY DISTRIBUTION OF CHANGE EMISSION

selection from the following user behaviors: benevolent user changes, system administration changes and malevolent user changes. The probabilities of emission per user profile and change type are elaborated on in Table I.

2) Entity Subscribers' Feedback Data: The second part of the evaluation dataset consists of subscriber feedback. We consider a few representative cases of subscribers to minimize the evaluation overhead, while providing useful insight on the adaptivity of the system:

- a subscriber who is interested in the deletion of attributes, to make sure no information concerning the entities he has subscribed to are missing. This scenario is expected to mostly use the type of the change description as a discriminating factor of importance.
- a subscriber mostly interested in the changes of names entities. This subscriber scenario is expected to mostly use the content of a change description as a discriminating factor of importance.
- a subscriber who is interested in whether any entity he has subscribed to has its "isDeceased" status changed. This scenario aims to represent the difficulty of users interested in specific attribute *values*.
- a subscriber who has the role of a validator of data in OKKAM, who wants to be aware of abnormal changes in default attributes, such as deletion or change with an abnormal value. This subscriber scenario is expected to use both the type and the content of a change description as a discriminating factor of importance.

A more detailed description of what each user finds interesting and critical can be found in Table II. All changes not noted within a profile are considered uninteresting for the profile. The profiles have been chosen so that they are require different kinds of information to be determined, concerning either the type or the content of the change.

Before the actual evaluation, we use the learning methodology and produce corresponding results, also supplying feedback for every step. This process is reiterated for every subscriber scenario for the whole set of change data. The information supposedly sent to the user is annotated with the iteration number, simulating the time-stamp of the change. For our case, iteration i is considered complete when the system gets the feedback of that i-th step from the user.

Subscriber	*Importance*	*Description*
Type-based	Critical	Attribute deletion.
	Interesting	Entity deletion.
Attribute name-based	Critical	Any change concerning an attribute that contains the string "name".
	Interesting	(None)
Attribute name-value pair-based	Critical	Attribute change or insertion on "isDeceased" attribute, with a new value of "true".
	Interesting	Attribute change or insertion on "isDeceased" attribute, with a new value of "false".
Complex	Critical	Default attribute (some attributes in the ENS are considered default — e.g., the name of a person entity — while all the others non-default) update or insertion with an abnormal value.
	Interesting	Default attribute deletion or normal update.

Table II
PROFILE DESCRIPTIONS. *Note:* ALL CHANGES NOT NOTED WITHIN A PROFILE ARE CONSIDERED UNINTERESTING FOR THE PROFILE.

B. Evaluation Methodology

To evaluate the system learning effectiveness, as well as whether the addition of content-related features is useful, we experiment on different aspects of the system response. We determine how quickly the system learns, by "emitting" changes to the supposed user — in groups of ten — and measuring how well the systems adapts to the feedback. We consider that the user feeds back the system after every new emission, by indicating the importance of all the items in the last group. The performance of the system for every emission is, at this point, based on the mean absolute error of the importance estimation. To judge the learning curve of the system, we study the magnitude of the mean absolute error as related to the current number of emissions. We also define the Rate of Acceptable Errors measure as an alternative measure of performance.

1) Rate of Acceptable Errors (RAE): We define the system performance for a given change emission to be the number of times a ranking error has exceeded 0.5. Given our $\mathbb{I}$ values, errors beyond this 0.5 margin *may* cause an error. Errors below this margin cannot cause an error by themselves. So the performance is *the percentage of the importance estimation in a given set that have their absolute error below* 0.5. Thus, a value of 1.0 in performance indicates a ranking that is ideal, while a value of 0.0 indicates a ranking that will have several errors. We call this measure *Ratio of Acceptable Errors (RAE)*. The formula, for a given set $\mathbb{D}_0$

Subscriber	*Graphs*	*Correlation (p-value)*
Type-based	✓	**-0.3398559** ($< 10^{-2}$)
		-0.2993715 ($< 10^{-2}$)
Attribute name-based	✓	**-0.3734062** ($< 10^{-2}$)
		-0.03564642 (0.2601)
Attribute name-value pair-based	✓	**-0.08581718** ($< 10^{-2}$)
		-0.02968072 (0.3484)
Complex	✓	**-0.5989662** ($< 10^{-2}$)
		-0.03393356 (0.2837)

Table III
CORRELATION BETWEEN EMISSION-ITERATION NUMBER AND MEAN ABSOLUTE ERROR PER SUBSCRIBER PROFILE AND METHOD. *Note:* HIGH STATISTICAL CONFIDENCE IS INDICATED USING **BOLD WRITING**.

of descriptions, their corresponding sequence of importance estimations $\tilde{\mathbb{I}_0} = \{y_l | y_l = f(D_l, U|\mathbb{F}), D_l \in \mathbb{D}_0\}$ and actual importance values $\mathbb{I}_0$, is:

$$RAE(\tilde{\mathbb{I}}, \mathbb{I}_0) = 1.0 - \frac{\sum_{i \in (1,\ldots,|\mathbb{I}|)} \lfloor min(|\tilde{\mathbb{I}}(i) - \mathbb{I}(i)|, 0.5) + 0.5 \rfloor}{|\mathbb{I}|} \quad (2)$$

where $\tilde{\mathbb{I}}(i)$, $\mathbb{I}(i)$ is the i-th element of the corresponding sequence, $\lfloor x \rfloor$ is the floor operator, $|\mathbb{X}|$, gives the number of elements of a sequence $\mathbb{X}$, $|x|$ is the absolute value of a number x and $min(x, y)$ is the minimum function.

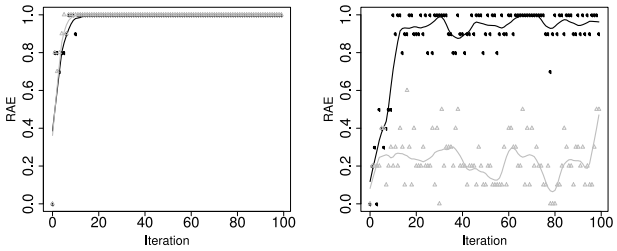
To determine whether the content aspect of the system is a valuable resource, we trying two alternatives. One uses the graph similarities as a feature, while the other does not. When we use content, we expect that the system will perform better for subscriber profiles that indeed use content criteria. For other profiles, we expect no loss in performance.

C. Learning Different User Profiles

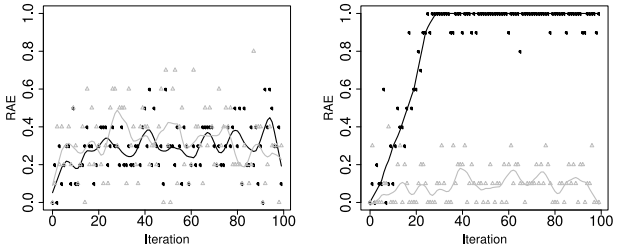
Table III indicates the Pearson correlation [30] between the number of emission, i.e., the training iteration, and the mean absolute error. A negative correlation indicates that, after performing more training the error is diminished. We see that in all cases, when using graphs, the system learns, with statistical support. In all the cases where we expected that the content should be used, indeed the system cannot learn when the content-sensitive methodology is not applied.

RAE as a function of time, as illustrated in Figure 7, indicates the learning curve of the system. The points in the graphs indicate the RAE for a given iteration. We note that there can be several RAE for a given iteration, since we have applied 10-fold validation of the results. The lines represent an approximation (LOWESS smoothing [31]) of RAE over iterations. Dark lines refer to the RAE when using graphs, while gray lines represent the performance without them.

It appears that, when the content methodology is used, all profiles are feasible to learn. At this original version, the most difficult case was the attribute name-value-based one (see Figure 7d), because we used to represent the name-value

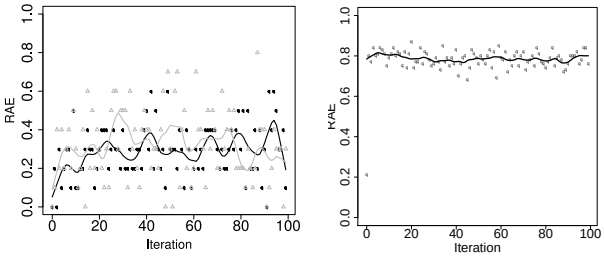


(a) Type-based (b) Attribute Name-based



(c) Attribute Name-Value-based (d) Complex

Figure 7. The learning curve of the first version of the system for different profiles.



(a) Common Graph (b) Separate Graphs

Figure 8. The learning curve in the original and the enhanced version of the system.

pair as a single string and, therefore, generate features of similarity for them in common, inducing noise in the n-gram graph pattern matching. In Figure 8 we see the evolution from the common graph to the separation of graphs for the attribute name and the attribute value, leading to new corresponding features in the vector space. It is obvious that the original problem has been adequately faced and the learning of the originally difficult case, now takes minimal iterations to occur.

In general, very few iterations (< 10) are enough for the profile to be learned. Even the complex profile is learned in about 20 iterations. This shows that the presented method-

ology is very promising in learning subscriber profiles on changes, but a new version should take into account the flaws of the first. Furthermore, in the original version there was no support for changing interest over time.

To study also the more realistic case when a user changes her interests, we created a *shifting-interest user* profile in our data generation process. This user changes her interests either in a periodic or a random manner. Upon the iteration of the interest shift the user is replaced by one of the basic profiles (see Table II) in a round robin manner. We will call this user *changing user*. In the second version of the system, we also split the representation of interest for attribute names and attribute values, to check whether all profiles could be effectively learned now.

D. Learning Changing Users' Preferences — Periodic Change

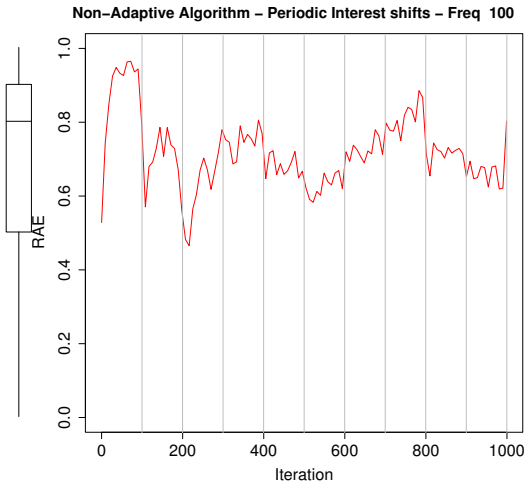
In the case of the periodic change we generated the feedback for the changing user, who every 100 iterations shifted her interest profile. Given our previous experiments, we had seen that the system takes about 20 iterations to learn a profile and, thus, the selection of 100 iterations gave enough time for the system to show its behavior.

In order to benchmark our shift-aware version of the system we first used the non-adaptive algorithm, where the system takes into account *all* the feedback from the user so far. This approach, presented in [1], can be effective for non-changing interests, given that the system, once reaching top performance, will stop taking new instances as data. It is important to note here that the feedback of the user can give a direct evaluation of each iteration, therefore providing information about whether the system has reached high performance.

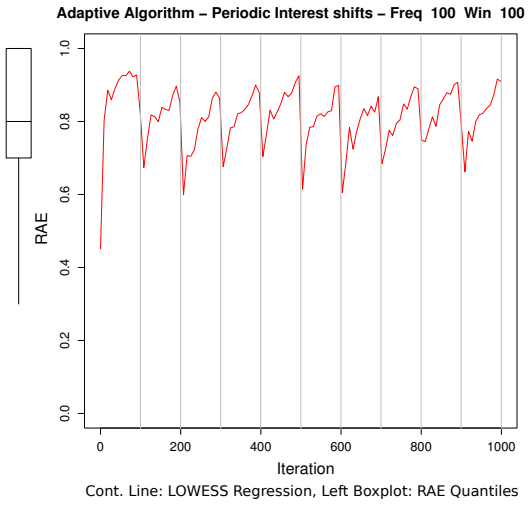
In Figure 9a, we present the behavior of the shift-unaware version of the system. The boxplot on the left indicates the quantiles of the distribution of the values. More precisely, for a population of N ordered sample values, it indicates the values at positions 1 (min. value), $\frac{N}{4}$, $\frac{2N}{4}$ (median), $\frac{3N}{4}$, N (max. value). The vertical gray lines indicate the interest shift. We repeat that, after every interest shift, the user follows another profile. The first thing that is apparent in the figure is that, even though the algorithm starts really well, after the first interest shift it never manages to reach the original levels the performance.

To compare the shift-unaware version to the final SAAS, we performed experiments with windows of various sizes, but first we tested the system with a window equal to the shift frequency[4]. The result is illustrated in Figure 9b. It is obvious that the performance of the SAAS is much more robust than that of the shift-unaware version.

[4]Within the text we will use the term frequency to actually describe the *period* of the shift. So 1 shift every 100 iterations will be described as having a frequency of 100.



(a) Shift-Unaware



(b) Shift-Aware

Figure 9. RAE over Time (LOWESS regression). The boxplots on the left of each diagram indicate the distribution of values.

In order to determine how sensitive the system is to the selection of the window we conducted a whole set of experiments for a range of different windows. The results, together with the performance of the shift-unaware version, are shown in Table IV. As can be seen, even the extreme cases where the window is really small do better *on average* from the shift-unaware version. This has to do with the fact that too much noise is induced in the training set if the memory holds all the history of feedback. Therefore, the shift-unaware version can be both ineffective and expensive in terms of storage when compared to SAAS.

Another important conclusion however from the table was that, even though the system is generally robust, the performance *does vary* according to the window size. The next step was to study what is the correlation of the window size to the performance. In fact, we studied the correlation between

Shift-	Window	Quantiles (Measurements in %) 1st	Median	Mean	3rd	Max
Unaware	N/A	50.0	80.0	70.1	90.0	100.0
Aware	5	60.0	80.0	73.0	90.0	100.0
Aware	10	60.0	80.0	74.9	90.0	100.0
Aware	20	70.0	80.0	78.1	90.0	100.0
Aware	40	70.0	80.0	79.4	100.0	100.0
Aware	80	70.0	90.0	80.3	100.0	100.0
Aware	90	70.0	80.0	80.0	100.0	100.0
Aware	100	70.0	80.0	79.4	100.0	100.0
Aware	110	70.0	80.0	78.8	100.0	100.0
Aware	120	70.0	80.0	78.3	100.0	100.0
Aware	150	60.0	80.0	76.3	90.0	100.0

Table IV
RAE PERFORMANCE QUANTILES OF SHIFT-AWARE VS SHIFT-UNAWARE VERSIONS. PERIODIC INTEREST SHIFT — EVERY 100 ITERATIONS.

Window	Quantiles (Measurements in %) 1st	Median	Mean	3rd	Max
10	60.0	80.0	76.2	90.0	100.0
20	70.0	80.0	78.3	100.0	100.0
40	70.0	80.0	79.6	100.0	100.0
60	70.0	80.0	78.9	100.0	100.0
80	70.0	80.0	80.3	100.0	100.0
100	70.0	90.0	79.9	100.0	100.0
120	70.0	80.0	77.5	100.0	100.0
140	60.0	80.0	77.2	100.0	100.0
160	60.0	80.0	74.9	90.0	100.0

Table V
RAE PERFORMANCE QUANTILES FOR RANDOM INTEREST SHIFT — EVERY 50-150 ITERATIONS UNIFORMLY

the ratio of the window to the interest shift frequency.

E. Learning Changing Users' Preferences — Random Change

Given the fact that we wanted to study what is the effect of having a small or large window as related to the size of the change frequency, we decided to make the experiment even more demanding: the user would randomly make the interest change. To create this random change we used a random sampler[5] that, given a mean of the change frequency M, would cause an interest shift in the range of $[\frac{M}{2}, \frac{3\times M}{2}]$.

First, we studied the performance of the SAAS for the *random* change with $M = 100$. The results can be seen in Table V.

To further delve between the relation of window size, M and RAE performance we repeated the experiments with $M = \{50, 75, 100, 125, 150\}$ iterations. The results can be found in Figure 10, where we depict the average level of performance (i.e., average RAE) for different $\frac{Window\ Size}{M}$ ratios, given various window sizes. From the diagram, we noted two things:

[5]Overall the sampler is adequately unbiased. For a study of the bias of the sampler, please see Appendix A.

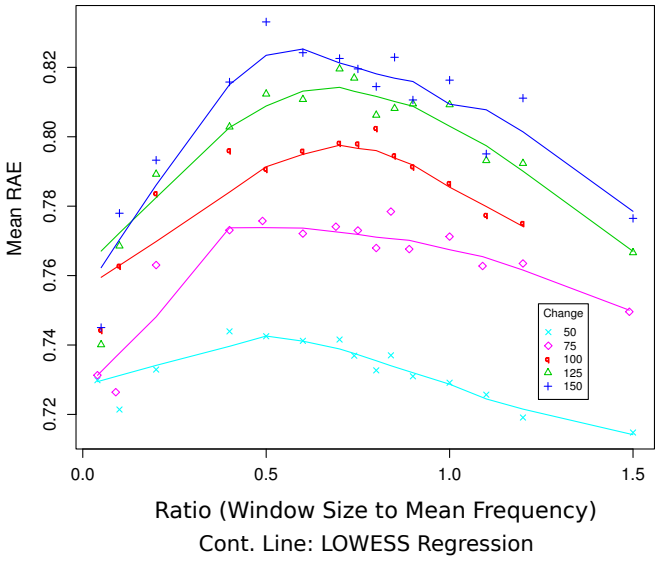


Figure 10. **RAE** Over **Memory Window to Mean Frequency Change Ratio** for SAAS.

- There seems to be a correlation between the average RAE performance of the system and the average frequency of the shift. This correlation was also proven statistically by a Pearson correlation test between the performance of the system and the average frequency of the shift. The test showed that there is a very strong linear correlation (approximately 0.8) with high statistical confidence (p-value was much lower than 10^{-2}. This simply verifies that a users that do not change frequently their interest will be easier to model.
- There is a range for the ratio of window size to expected change frequency, which appears to hold high performance, regardless of the frequency itself. To better illustrate and estimate this range we provide Figure 11, where we present the overall — i.e., over all window sizes — relation between RAE and $\frac{Window\ Size}{M}$ ratio. Given that the estimated user interest shift frequency is F, then it appears that the range between $0.4 \times F$ and $0.8 \times F$ offers near-optimal results. This indicates that it is enough to use the expected value of the frequency to get near-optimal results. Given the fact that less memory means less space, one could select to use $0.4 - 0.5$ of the estimated user interest shift frequency to have low spatial cost with near-optimal performance.

VI. CONCLUSION AND FUTURE WORK

We propose an adaptive subscription service architecture, concerning the update of clients of an ENS with information on entity changes. The subscription service, called *SAAS*, uses information from user feedback to model user needs, taking into account *both the type and the content* of changes. The system appears to learn even complex user preferences

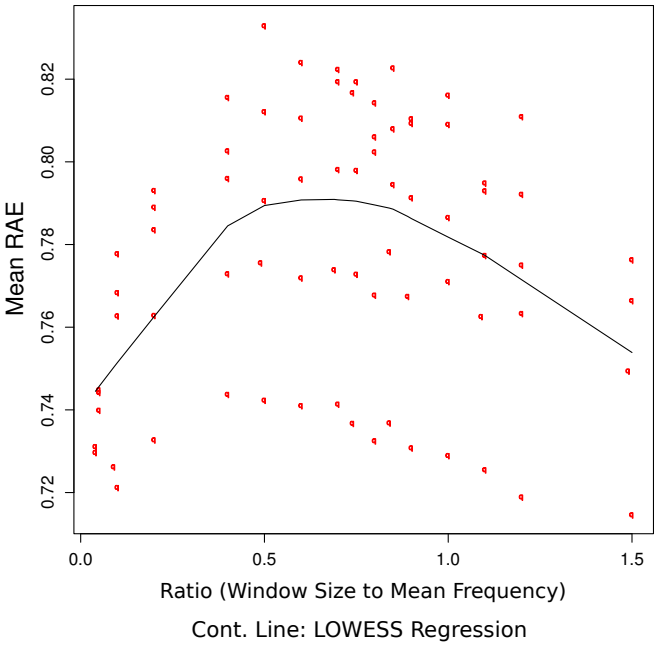


Figure 11. **Overall RAE** Over **Memory Window to Mean Frequency Change Ratio** for SAAS.

in a low number of feedback iterations, also coping with user interest shifts. Our system performs well, even with randomly changing user interest. It should be noted that in our experiments we used a radical shift of interest every time, which made the problem even more demanding than an average real case scenario. Nevertheless, the average performance is high and the system appears robust and able to cope with all different user profiles.

We studied the relation between the memory window size parameter of the user modeling process to the average performance of the system. There we detected a near-optimal range for the parameter, which can be calculated as a simple function of the estimated average change frequency of the user interest. We showed that the mean performance of the system is highly correlated to the interest shift frequency, but the system remains robust enough even for frequent interest shift changes.

In the future, we plan to study on information theoretic grounds whether one can determine a priori the performance of adaptive systems, which use user feedback history, based on a few easily extracted parameters of the system. For unchanging user behavior, we may also need to keep the memory window constant, i.e., not dependent on the current iteration, until a drop in performance indicating user interest shift occurs. This means that need to integrate the ability of detection of user interest shift into the system to improve on our performance. This interest shift detection should probably be based on the grounds of time-series analysis of the feedback data. It will also be important to compare our methods with an alternative that should use active learning to augment the training set with useful additions. Finally, it is important to use the content and type discrimination in other applications of the user modeling domain and determine whether there is a generic gain to using this approach.

ACKNOWLEDGMENT

This work is partially supported by the by the FP7 EU Large-scale Integrating Project OKKAM "Enabling a Web of Entities" (contract no. ICT-215032). See http://www.okkam.org.

REFERENCES

[1] G. Giannakopoulos and T. Palpanas, "Adaptivity in entity subscription services," in *Proceedings of ADAPTIVE2009*, Athens, Greece, 2009.

[2] P. Bouquet, H. Stoermer, and B. Bazzanella, "An entity name system (ENS) for the semantic web," in *ESWC*, 2008, pp. 258–272.

[3] T. Palpanas, J. A. Chaudhry, P. Andritsos, and Y. Velegrakis, "Entity data management in OKKAM," in *DEXA Workshops*, 2008, pp. 729–733.

[4] B. Bazzanella, J. A. Chaudhry, T. Palpanas, and H. Stoermer, "Towards a general entity representation model," in *SWAP*, 2008.

[5] E. Rich, "Building and exploiting user models," in *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 2*, 1979, p. 720722.

[6] P. Langley, "Machine learning for adaptive user interfaces," *Lecture notes in computer science*, pp. 53–62, 1997.

[7] M. Shepherd, C. Watters, and A. Marath, "Adaptive user modeling for filtering electronic news," in *System Sciences, 2002. HICSS. Proc. of the 35th Annual Hawaii Int. Conf. on*, 2002, pp. 1180–1188.

[8] X. Zhou and T. Huang, "Relevance feedback in image retrieval: A comprehensive review," *Multimedia systems*, vol. 8, no. 6, pp. 536–544, 2003.

[9] D. Bonnefoy, M. Bouzid, N. Lhuillier, and K. Mercer, "More like this or Not for me: Delivering personalised recommendations in multi-user environments," in *User Modeling 2007*, 2007, p. 8796.

[10] J. Rocchio *et al.*, "Relevance feedback in information retrieval," *The SMART retrieval system: experiments in automatic document processing*, pp. 313–323, 1971.

[11] C. Buckley, G. Salton, and J. Allan, "The effect of adding relevance information in a relevance feedback environment," in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Dublin, Ireland: Springer-Verlag New York, Inc., 1994, pp. 292–300. [Online]. Available: http://portal.acm.org/citation.cfm?id=188490.188586&type=series

[12] P. Zigoris and Y. Zhang, "Bayesian adaptive user profiling with explicit & implicit feedback," in *Proc. of the 15th ACM Int. Conf. on Information and knowledge management*. ACM New York, NY, USA, 2006, pp. 397–404.

[13] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[14] G. Giannakopoulos, "Automatic summarization from multiple documents," Ph.D. dissertation, Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece, http://www.iit.demokritos.gr/˜ggianna/thesis.pdf, April 2009.

[15] J. Allan, "Incremental relevance feedback for information filtering," in *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. Zurich, Switzerland: ACM, 1996, pp. 270–278. [Online]. Available: http://portal.acm.org/citation.cfm?id=243274&dl=

[16] A. Leuski and J. Allan, "Interactive information retrieval using clustering and spatial proximity," *User Modeling and User-Adapted Interaction*, vol. 14, no. 2, pp. 259–288, Jun. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:USER.0000028978.09823.47

[17] M. McTear, "User modelling for adaptive computer systems: a survey of recent developments," *Artificial intelligence review*, vol. 7, no. 3, pp. 157–184, 1993.

[18] W. Cohen, R. Schapire, and Y. Singer, "Learning to order things," *J Artif Intell Res*, vol. 10, pp. 243–270, 1999.

[19] K. Onuma, H. Tong, and C. Faloutsos, "TANGENT: a novel,'Surprise me', recommendation algorithm," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM New York, NY, USA, 2009, pp. 657–666.

[20] M. Vojnovic, J. Cruise, D. Gunawardena, and P. Marbach, "Ranking and Suggesting Popular Items," *Online im Internet: http://research. microsoft. com/˜ milanv/popularity. pdf [10.06. 2008]*, 2007.

[21] N. Nanas and A. de Roeck, "Autopoiesis, the immune system, and adaptive information filtering," *Natural Computing*, vol. 8, no. 2, pp. 387–427, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1007/s11047-008-9068-x

[22] W. Lam and J. Mostafa, "Modeling user interest shift using a bayesian approach," *Journal of the American Society for Information Science and Technology*, vol. 52, no. 5, pp. 416–429, 2001.

[23] G. Giannakopoulos, V. Karkaletsis, G. Vouros, and P. Stamatopoulos, "Summarization system evaluation revisited: N-gram graphs," *ACM Trans. Speech Lang. Process.*, vol. 5, no. 3, pp. 1–39, 2008.

[24] M. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent systems*, vol. 13, no. 4, pp. 18–28, 1998.

[25] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[26] V. Vapnik, "Structure of statistical learning theory," *Computational Learning and Probabilistic Reasoning*, p. 3, 1998.

[27] A. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[28] J. Allan, "Incremental relevance feedback for information filtering," in *Proc. of the 19th annual Int. ACM SIGIR Conf. on Research and development in information retrieval*. ACM New York, NY, USA, 1996, pp. 270–278.

[29] U. Cetintemel, M. Franklin, and C. Giles, "Self-adaptive user profiles for large-scale data delivery," in *Data Engineering, 2000. Proc. of 16th Int. Conf. on*, 2000, pp. 622–633.

[30] M. Hollander and D. Wolfe, "Nonparametric statistical methods," *New York*, p. 518, 1973.

[31] W. Cleveland, "LOWESS: A program for smoothing scatterplots by robust locally weighted regression," *American Statistician*, pp. 54–54, 1981.

APPENDIX A.
THE BIAS OF THE RANDOM SAMPLER

To determine whether we have a bias concerning the frequency sampling from the uniform distribution over the values [70,210], we perform bootstrapping over our sample values for all folds and we measure the mean difference between the input value and a randomly generated, equally sized, sequence of uniformly sampled numbers from the same interval (generated by the R statistical package). In every bootstrap, a new sequence of random numbers is used. This is repeated 20000 times and we can then estimate the mean difference between the means, within a 99% interval.

The two-tailed equi-tailed interval returned for the 99% confidence level indicates that it is most probable that the mean difference between the numbers generated within the experiment and a gold-standard generator range from (approximately) -13 to to 2, which is a pretty good indication that our generated values can be considered as uniformly distributed within the 50, 150 interval.