NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 – "Semantic-based knowledge and content systems"

D 1.6.1 Predicting future structural changes in ontologies

Deliverable Co-ordinator: **Janez Brank**

Deliverable Co-ordinating Institution: J. Stefan Institute

Other Authors: Marko Grobelnik, Dunja Mladenić

This deliverable deals with the topic of prediction of structural changes in an ontology. We examine several different types of structural changes occurring in a large real-world ontology (the Open Directory Project topic hierarchy) over the course of several years. We show how the prediction of a particular type of structural changes, namely the addition of new subconcepts, can be approached as a machine-learning problem. We also present an experimental evaluation of our proposed approach.

Document Identifier:	NEON/2007/D1.6.1/v1.0	Date due:	August 31, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	August 31, 2007
Project start date:	March 1, 2006	Version:	V1.0
Project duration:	4 years	State:	Final
		Distribution:	Report/Public

NeOn Consortium

This document is a part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta} @open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 28 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 655 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Martino della Battaglia, 44 - 00185 Roma-Lazio, Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 1 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

Institut ‘Jožef Stefan’ (JSI)

Change Log

Version	Date	Amended by	Changes
0.1	06/08/2007	Janez Brank	Initial draft
0.2	03/09/2007	Janez Brank	Adding experiment results
0.3	06/09/2007	Marko Grobelnik	Overall revision
0.4	07/09/2007	Dunja Mladenić	Overall revision
0.5	16/09/2007	Janez Brank	Follow-up to reviewer's comments
1.0	05/10/2007	Janez Brank	Follow-up to QA comments

Executive Summary

This document discusses the problem of predicting the structural changes in an ontology. It addresses ontologies that contain instances in addition to concepts. The focus is on an ontology where the instances are textual documents, but the approach presented in this document is general enough to also work with other kinds of instances, as long as a similarity measure can be defined over them.

We examine the changes in the Open Directory Project ontology over a period of several years and analyze the most common types of structural changes that took place during that time. We then present an approach for predicting one of the more common types of structural changes, namely the addition of a new concept that becomes the subconcept of an existing parent concept and adopts a few instances of this existing parent concept.

We describe how this task can be formulated as a machine-learning problem and present an experimental evaluation of this approach that shows promising results of the proposed approach.

Table of Contents

List of Figures

List of Tables

1. Introduction

Many ontologies are not static objects. If an ontology is a shared conceptualization of a domain, it is not surprising that it may have to change in response to changes in either the domain itself, or in our understanding of it, or in the purposes with which we are building a shared conceptualization of it. Thus it is natural to ask whether such changes in an ontology can be predicted automatically as an aid to the people maintaining the ontology.

In this document we begin by discussing an example of a large real-world ontology whose evolution over the course of several years can be readily observed, namely the topic hierarchy of the Open Directory Project (ODP; see *http://www.dmoz.org/*). We identify the most common types of structural changes occurring in this ontology and analyze their frequency. Based on these observations, we decide to focus on trying to predict one specific type of structural changes: the addition of a new subconcept as a child of an existing parent concept, from which the new concept also takes a few instances. This is one of the more common types of structural changes in the ODP, and it is also the kind of operation that appears amenable to an automatic prediction approach.

We then discuss how the problem of predicting this kind of subconcept additions can be formulated as a machine learning task. The main challenge here is to describe a concept by a set of features in such a way that a predictive model (obtained through machine learning) will be able to predict, from these features, whether a new subconcept should be added below the given concept or not. Our approach is based on the assumption that the ontology contains not only concepts but also instances, and a new subconcept should be added if there exists a subgroup of closely related instances in the parent concept. We cluster the instances of the parent concept and compute several statistical properties of the resulting partition of the instances into clusters. In the case of the ODP, the instances are textual documents, so that techniques from information retrieval can be used for the needs of cluster analysis.

We also present an experimental evaluation of the proposed approach. Experiments on the ODP ontology show that this is feasible approach for predicting this type of ontology changes.

Finally we will discuss a few ideas for future work, especially with a view to predicting other types of structural changes that are not addressed by the approach presented in this report.

1.1 Connection to NeOn

The work presented here is of interest to NeOn because it illustrates and quantifies the process of (manual) ontology construction and evolution through time. The methodology we used to predict structural change could be incorporated into the NeOn software toolkit as a helpful aid to human editors of ontologies, thus providing functionality that is not widely available in other ontology editing software. The approach presented in this deliverable could be applied on many lightweight ontologies, especially those involving textual documents as instances. As such, it can be of interest for the FAO case study.

1.2 Related work

[12] and [10] defined three types of change discovery: structure-driven (where suggested changes are deduced from analyzing the ontology structure itself), usage-driven (changes are recommended by observing the usage patterns over time) and data-driven (which is based on changes in the underlying data that describes the domain of interest).

An example of work focusing on usage-driven change discovery is [9]. [8] discussed the incorporation of data-driven change discovery into a framework for learning an ontology from a corpus of textual documents.

For a recent overview of the area of ontology change, and its relationship with ontology evolution, merging, and integration, see the survey by Flouris et al. [13].

Within the NeOn project, ontology change has been discussed in the deliverable D1.6.1, “Dynamics of Metadata” [14], which defines a number of ontology change operations. The operations defined there are relatively low-level, whereas the changes which we attempt to predict in the work reported in the present deliverable are somewhat higher-level, and can be seen as aggregations of several low-level operations in the sense of D1.6.1. A more detailed discussion of this relationship will be presented in Section 2.4.

2. Identifying structural changes by comparing two states of an ontology

2.1 The Open Directory Project dataset

To investigate the issue of structural changes in an ontology, it is helpful to consider a real-world ontology for which it is possible to observe the changes through a period of time. Additionally, the ontology should be reasonably large, so as to provide a sufficient amount of data for the training of predictive models. We decided to use the topic ontology of the Open Directory Project (ODP, available from *http://www.dmoz.org/*).

In the ODP ontology, the concepts are actually topical categories; they are organized into a tree via the parent-child relationship. In addition there also exists another type of relationships, the "symbolic links", which point from a category to another category that is not a child of the first but deals with a related topic. Each category has a name and a short description (this description is not usually shown to the user while browsing the ODP web pages; however, it is available in the downloadable XML files containing a dump of the entire ODP ontology). In practice, different categories may have the same name, and to obtain a unique name for a category, one must concatenate the names of all the categories on the path from the root of the hierarchy to the category in question.

In addition the ODP ontology contains instances; these are actually links to external web pages. Besides the URL of the external web page, each instance also contains a title and a short textual description of the page, usually one or two sentences long. Thus we will regard each instance as a short textual document, and techniques from the area of text mining will be used in dealing with the data.

Note that our approach for predicting structural changes is crucially dependent on the fact that the ontology is well populated with instances. It is not, however, dependent on the fact that the instances are textual documents. As we will see later, the proposed approach only assumes that the instances can be clustered; for that, the only thing one really needs is a measure of similarity (or distance) between the instances.

The ODP ontology is interesting for our purposes because snapshots of the ontology at different points in time are available. The ODP makes available approximately one snapshot per month, usually near the beginning of the month and reflecting the state of the ontology on a particular day. Almost 50 such snapshots are available, one for each month from July 2003 onwards, as well as for a few earlier months (going back all the way to January 2001). These snapshots can be downloaded from *http://rdf.dmoz.org/rdf/archive/*.

One problem with the ODP dataset, from the point of view of predicting structural changes, is that any two consequent snapshots are approximately a month apart and a number of structural changes can take place during that time period. Sometimes several of these structural changes affect the same part of the ontology, and it isn't possible to uniquely determine the exact sequence of structural changes that took place. We developed a set of heuristics to compare two snapshots of the ontology and output a set of operations that could change the earlier snapshot into the later one. Of course, there is no guarantee that this is exactly the same sequence of operations that was actually performed by the human editors of the ODP ontology, as the same changes in the ontology can be effected through several different sequences of operations. In addition, the sequence of operations will depend on what set of elementary transformations one is willing to employ.

2.2 Low-level structural changes

Changes in the ontology may be roughly divided into those that affect the categories (i.e. concepts) and those that affect the documents (i.e. instances). The latter group consists of the inclusion of new documents (e.g. newly discovered external web pages) into the ontology, removal of old documents (e.g. dead links), or rearrangement of existing documents within the ontology. We will not attempt to predict these document-level operations because, first of all, most of them cannot really be understood as causing structural changes in the ontology, and secondly, because they it would be difficult to predict them without additional (and often unavailable) external data. For example, a document may have been removed from the ontology because the external web page to which it pointed had changed or had gone offline; but this kind of information is not available within the ontology snapshots on the ODP web site. It could only be recovered if the past state of the external web pages in question was available, e.g. through the internet archive (*http://www.archive.org/*). Similarly, predicting the inclusion of new documents would require information about which web pages were available at a certain point in the past, so that they could have been discovered by the ODP editors and considered for inclusion in the ontology. However, we consider such questions to be outside the scope of this deliverable.

Thus we will focus on changes involving categories instead. In principle, one snapshot of the ontology can always be transformed into another one by a sequence of two elementary operations: addition and deletion of categories. By comparing the set of categories in one snapshot with the set of categories in the previous month's snapshot, it is easy to see which categories are missing and which are new. For example, the following list shows a subset of the changes that we may notice within the *Top/Computers* subtree of the ontology between April 3 and May 1, 2007. "DEL" indicates that a category was deleted (i.e. it was present on April 3 but not on May 1) and "ADD" indicates that it was added (i.e. it was present on May 1 but not on April 3):

DEL Top/Computers/Open_Source/Software/Games/FPS
DEL Top/Computers/Software/Internet/Servers/Directory/LDAP
DEL Top/Computers/Software/Internet/Servers/Directory/LDAP/Products
DEL Top/Computers/Software/Internet/Servers/Directory/LDAP/Standards_and_Organizations
DEL Top/Computers/Software/Internet/Servers/Directory/LDAP/Products/Related_Middleware
DEL Top/Computers/Software/Internet/Servers/Directory/LDAP/Products/Related_Client_Apps
ADD Top/Computers/Open_Source/Software/Games/Shooter
ADD Top/Computers/Programming/Languages/Smalltalk/Squeak/Croquet
ADD Top/Computers/Programming/Languages/Smalltalk/Squeak/Croquet/News_and_Media
ADD Top/Computers/Internet/Protocols/LDAP
ADD Top/Computers/Internet/Protocols/LDAP/Standards_and_Organizations
ADD Top/Computers/Internet/Protocols/LDAP/Software/Client
ADD Top/Computers/Internet/Protocols/LDAP/Software/Server
ADD Top/Computers/Internet/Protocols/LDAP/Software

As we can see from this list, it is unsatisfactory to describe the transformation of one snapshot to another solely through these two types of low-level operations. Although one can in principle transform the April snapshot to the May snapshot by deleting the first six categories and then adding the next eight ones, it is clear that the human editors working on the ontology must have really conceptualized their work as a sequence of more abstract, higher-level operations, each of which may then be manifested in one or more low-level additions and deletions of the type seen in our list above.

In our example, we can see that the removal of *.../FPS* and the addition of *.../Shooter* are really two related operations: in other words. "FPS" has simply been renamed "Shooter" (note that FPS is itself nothing but an acronym for "first-person shooter", a genre of computer games).

Similarly, the deletions and additions related to LDAP show us that the whole *LDAP* subtree, which had previously been a subtree of *.../Internet/Servers/Directory*, has been moved under *.../Internet/Protocols* instead. Additionally, the *Products* subtree has been renamed into *Software* and rearranged somewhat, while the *Standards_and_Organization* subtree has remained unchanged.

Finally, the *.../Squeak/Croquet*, with its *News_and_Media* child, is a genuinely new subtree, in which not only the categories themselves are new in the May 1 snapshot, but they also (as it turns out) contain documents that did not exist at all in the April 3 snapshot. This means that evidently an entirely new set of external web pages came to the attention of the ODP editors, dealing with a topic that has previously not been represented in the ontology. Thus not only the documents were added but a new category was created for them.

There also exist other types of structural changes not illustrated by the above example. One typical ODP phenomenon, which accounts for many additions of new categories, is the creation of new subcategories in advance, i.e. without at the moment having any documents to populate them with. The human editors of the ODP create such additions in the understanding that documents for them are likely to eventually appear but the structure of the subtree is clear and predictable enough that the subcategories may be created right away and will then be used to guide the inclusion of new documents in the tree. For example, by comparing the snapshots for April 1 and May 4, 2004, we see that 36 new children were added below the category *Top/Computers/Programming/Internet/-ASP/ASP.NET/Web_Hosting*. The names of these children are simply the letters *A* through *Z* and the digits *0* to *9*. Clearly the intention was to divide an existing list of internet hosting providers into smaller subcategories based on the first character of their name. Some of these subcategories, e.g. *J* and *Q*, were initially empty (i.e. no hosting providers had names beginning in J or Q), but they were created anyway for the sake of consistency.

Similar sets of subcategories (many of which are initially empty) are sometimes created corresponding to other interesting sets of concepts from the real world, e.g. names of geographical entities. A typical example is to have 51 new children suddenly created for a given category, corresponding to the U.S. states and the District of Columbia. In other areas of the ODP ontology, parts of the ontology may correspond to existing ontologies such as the hierarchy of taxonomic units used in zoology and botany. Suddenly a whole subtree of mostly-empty ODP categories may spring into existence when an editor has decided to import some part of the existing hierarchy of zoological orders, families, genera etc. into the ODP hierarchy. We consider such changes to be too strongly dependent on background knowledge and high-level abstract decisions by a human editor to be predictable by a computer. Therefore, our efforts to predict structural changes will focus on situations when a new category has been added and some documents from previously existing categories transferred into it; this suggests that the structural change in question was genuinely an editor's response to the available data, and it may therefore be predicted automatically given the same data. An example may be that an editor decides that a category has too many documents and is too diverse, and it may therefore be split into several subcategories corresponding to narrower subtopics, with the documents then divided among these subcategories.

2.3 Heuristics for the identification of higher-level structural changes

As we have seen in the previous section, low-level additions and deletions of categories can be easily observed by comparing two snapshots of the ontology, but the really interesting operations are more abstract and each such operation can give rise to several low-level additions and deletions. In addition, many such operations can take place in the period of time (e.g. a whole month) between two snapshots, and where several such operations affect the same part of the ontology, it can be difficult to identify the abstract operations given the set of low-level additions and deletions that can be discerned from the data. Thus, it is helpful to develop reasonably robust heuristics that can identify at least some of these higher-level operations, with the understanding that we cannot expect them to correctly identify them in all situations.

As it turns out, the largest group of low-level additions and deletions are actually due to the renaming of categories (e.g. *FPS* to *Shooter* in the example in the previous section). If a category with many descendants is renamed, this may manifest itself as a large number of low-level additions and deletions (as if each descendant was deleted and then re-created under a new path in the tree). Although this is a common operation, these are not really structural changes and so

we want to recognize them and exclude them from further consideration. For this purpose we use a heuristic based on the notions of precision and recall from information retrieval. Given a category *C* from the old snapshot that does not appear (under the exact same name) in the new snapshot, we consider the set *S* of all documents from this category and its descendants (in the old snapshot). For each category *C'* of the new snapshot, we can similarly form a set *S'* of all documents of this category and its descendants. Then the recall of *C'* with respect to *C* can be defined as $|S \cap S'|/|S|$, and the precision of *C'* with respect to *C* can be defined as $|S \cap S'|/|S'|$. If *C'* is to be recognized as a new incarnation of *C* (under a new name), it should ideally have high recall and high precision as well. In information retrieval, precision and recall are traditionally combined into a value called the F_1-measure, which is simply the harmonic mean of precision and recall: $F_1 = (2 \cdot precision \cdot recall) / (precision + recall)$. As the harmonic mean, F_1 is high only if both precision and recall are high. Insisting on a high recall is obviously desirable, but a good argument can be made for requiring high precision as well. For example, *C* may have been renamed into *C'* but some of its documents may have been moved into the parent of *C'*. Thus the parent will have higher recall than *C'*, but typically (since it contains many things that were never in either *C* or *C'*) much lower precision; thus, if we use F_1 instead of just recall, we will successfully avoid matching *C* with the parent of *C'* and will correctly match it with *C'* itself instead.

Note that it is possible that new documents were introduced into the ontology in the time between the old and the new snapshot, and some of these documents may have ended up in *C'*; these would increase the size of *S'* but not of $S \cap S'$ (as they did not appear in the old snapshot), whereby decreasing the precision. Thus, to prevent such new documents from unfairly affecting the match between *C* and *C'*, we take into *S'* only those documents that have already existed in the ontology at the time of the old snapshot.

Thus, for each deleted category from the old snapshot, we find its best match (i.e. the one with maximal F_1) in the new snapshot. In principle, it is in possible that there is no really good match, e.g. if the category and its documents were really deleted from the ontology, rather than simply renamed. In our experience, such deletions are rare; however, since we often work with just a part of the whole ontology for reasons of faster experimentation (e.g. just the subtree rooted in *Top/Computers*, etc.), it can happen that a category is moved outside of the part of the ontology that is under consideration, which is thus effectively the same as if it had been deleted entirely.

For the purposes of detecting the renaming and moving of categories, we consider only matches with a recall of at least 90%. We will refer to these as “strong matches”. The next step is to combine the matches on the level of categories into matches on the level of entire subtrees. For example, if a deleted category *Top/A/B/C*, with children *Top/A/B/C/D1* and *Top/A/B/C/D2*, is found to match strongly with a new category *Top/E/C'*, and furthermore its two children match strongly with two new categories *Top/E/C'/D1'* and *Top/E/C'/D2'*, then it is reasonable to refer to this as a move operation on the entire subtree rooted in *Top/A/B/C*, rather than as a set of operations that happened individually and separately to *C*, *D1* and *D2*. In general, the subtree rooted in *C* may be deeper (i.e. there may be grandchildren and other descendants in addition to just children), so the heuristic we actually use is the following. We say that there is a strong match between the subtree rooted by *C* (in the old snapshot) and the one rooted by *C'* (in the new snapshot) if the following two conditions are met: (1) For each descendant *D* of *C* (in the old snapshot), there must exist an strong match *sm*(*D*) (in the subtree rooted by *C'* in the new snapshot); and (2) furthermore, for each such *D* we require that *parent*(*sm*(*D*)) = *sm*(*parent*(*D*)). In other words, we consider a strong match between subtrees to exist in cases when a strong match exists for each category in the subtree and the matches preserve the parent-child relationships. At the same time, our definition is robust in the sense that the addition of new categories into the subtree rooted by *C'*, or the merging of several old categories into a new one, does not prevent us from recognizing the strong match between the subtrees.

The strong matches between entire subtrees, once they have been identified, are a good first step towards the identification of several types of higher-level structural changes:

- If the subtree of *C* (in the old snapshot) strongly matches the subtree of *C'* (in the new snapshot), and *C'* did not exist in the old snapshot, and *C* and *C'* have the same parent,

and no other subtree of the old snapshot strongly matches that of *C'*, then we say that *C* has been renamed into *C'*.

- If the same conditions are true except that *C* and *C'* do not share the same parent, we say that *C* has been moved to become *C'*.
- If, on the other hand, *C'* has already existed in the old snapshot or it is new but some other subtree besides that of *C* has strongly matched the subtree of *C'*, then we say that *C* has merged into *C'*. Sometimes a category may merge into its parent, for example if the editor has decided that the previous subdivision was excessively fine-grained and the topics represented by the categories were too narrow. On the other hand, sometimes a category merges into some more distant relative rather than a parent. It can also happen that several categories merge into one.

As an example, the chart in Figure 1 shows the frequency of these various types of higher-level ontology changes within the *Top/Computers* subtree of the ODP ontology, over the last three years. As has been described above, all the category deletions that have been observed as low-level structural changes have now been explained as either renames, moves, or merges, with merges further divided into many-to-one merges, merges into parent and merges into other (nonparent) categories. What remains are the additions of genuinely new categories, rather than categories which appear new but are included in a strong subtree match with some formerly existing category (meaning that they are really the result of a rename, move or merge). It can be seen that additions are by far the most frequent structural changes, followed by renames and moves. Merges are comparatively rare. Since it is debatable to what extent a rename can be considered a truly structural change, and since moves are already fairly rare relative to the additions, we decided to concentrate on additions from now on as the most important and most frequently occurring type of structural change in the ODP ontology.

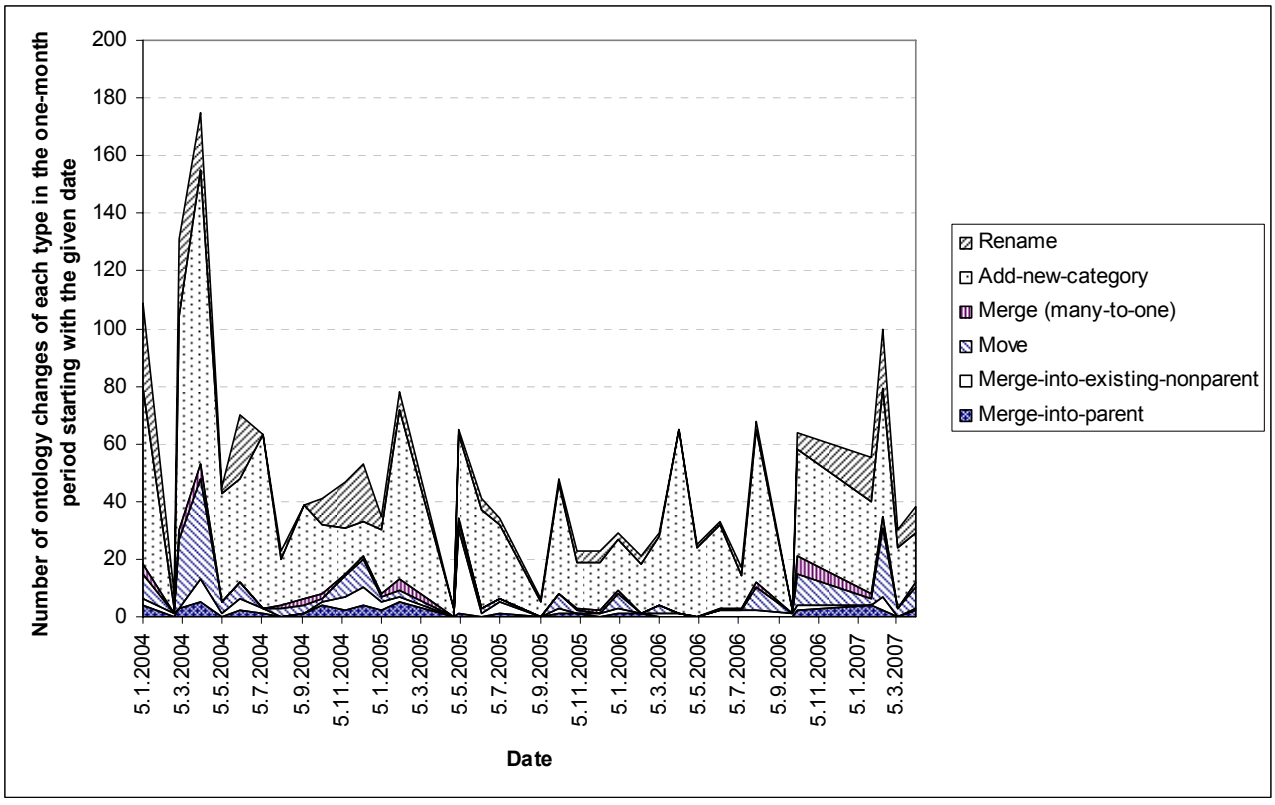

Figure 1: Frequency of various types of ontology changes.

2.4 Discussion: relationship to the ontology changes from D1.5.1

The NeOn deliverable D1.5.1, "Dynamics of Metadata" [14], has already presented a very thorough and detailed typology of ontology change operations. The operations that we focus on in this report differ from those of D1.5.1 in two main ways. Firstly, they are not as fine-grained; our operation of category addition, for example, corresponds to a whole sequence of the kind of operations discussed in D1.5.1 (creation of a new category; creation of a parent-child link between the new category and its parent; and the transfer of some of the instances of the parent into the new child category). Secondly, some of the D1.5.1 operations do not apply to the relatively simple ODP-like ontologies with which we deal here. For example, in ODP the categories and instances do not have structured attributes (with slots, etc.), and furthermore all instances have the same attributes (URL, title, and description) regardless of which category they belong to.

Our choice of the set of operations has been informed primarily by the following considerations:

- We wish to conceptualize the ontology editing process on roughly the same level where the human editors of the ODP operate. For example, an editor probably thinks "I will move the category C_1 to make it a parent of category C_2", rather than "I will remove the parent-child link between C_3 and C_1 and then add a parent-child link between C_2 and C_1". Since we would like to model the expertise of these editors, we would like to think in terms of the same operations that they probably use.
- Additionally, we require operations of the kind that can be observed in the available ODP data, and that can furthermore be modeled and predicted via a machine learning approach. For example, since in the ODP a new category is never added without also being clearly defined as the child of some other category, we cannot really model a "concept creation" operation separately from a "creation of a parent-class link" operation.

2.5 Different types of category additions

As we saw on Figure 1, the addition of new categories is the most common type of structural change, even after we exclude the categories that seem to be new but are really just old categories that have been renamed, moved or merged. In this section we will look at the additions of new categories in more detail. If we take the total over the entire three-year period covered by Figure 1, we find that there were 1115 category additions within the *Top/Computers* subtree during this period. Figure 2 shows how these additions can be divided into several kinds.

First of all, sometimes what is added is not just a simple leaf node of the tree but a whole subtree, consisting of a category and one or more children and possibly other descendants as well. Thus it turns out that approx. 20% of the newly added categories had a parent that was also newly added at the same time (or at least within the same month – remember that the snapshots of the ontology that we worked with are approximately one month apart from each other). We will not attempt to predict the addition of such categories, as it is challenging enough to predict the addition of an individual category, much less of a whole subtree. Thus the remaining groups of additions discussed in this subsection consist of new categories added to a previously existing parent.

Approximately 10% of the new categories were empty, i.e. they contained no documents at all. As has been discussed in the previous section, these are mostly caused by systematic additions of large groups of sibling categories, e.g. corresponding to U.S. states or to letters of the alphabet.

Approximately 20% of the new categories are not empty, but they contain only documents that did not exist in the ontology at the previous point in time for which a snapshot is available. This suggests that the category has been added on the basis of external web pages that were newly discovered and included in the ontology (e.g. the Croquet example from Section 2.2), or that the category has been moved into the *Top/Computers* subtree from a different part of the entire ODP ontology.

Approximately 44% of the new categories could reasonably be said to have been obtained by splitting a previously existing parent category. This group of additions was defined as follows: the new category (e.g. *C*) must be the child of a previously existing parent (e.g. *P*); it must contain at least one document from the old snapshot of the ontology (although it may also contain zero or more new documents), and of these documents from the old snapshot, the majority must come from *P* or one of its descendants, rather than from some other part of the ontology that does not lie below *P*. In other words, to consider an addition to be a split of an existing category, we require that the new child adopts more documents from the parent than from other parts of the old ontology. This is the type of additions that our prediction efforts will chiefly focus on. Unfortunately it turns out that most of the categories added in this way are fairly small; only approx. a third of them (16% of all additions) contain at least five documents from *P*.

Finally, the remaining additions result in categories that contain some mixture of old documents from *P*, old documents from other parts of the ontology, and entirely new documents. In approx. half of these (4% of all additions), the new documents predominate; in the others most of the documents are from the old ontology, but with those from *P* outnumbered by those from outside of *P*. These new categories are thus obtained by a combination of new data (web pages newly included in the ODP directory) and of rearranging and moving of existing data (previously existing documents), and it is not clear that they can be characterized in any unified way.

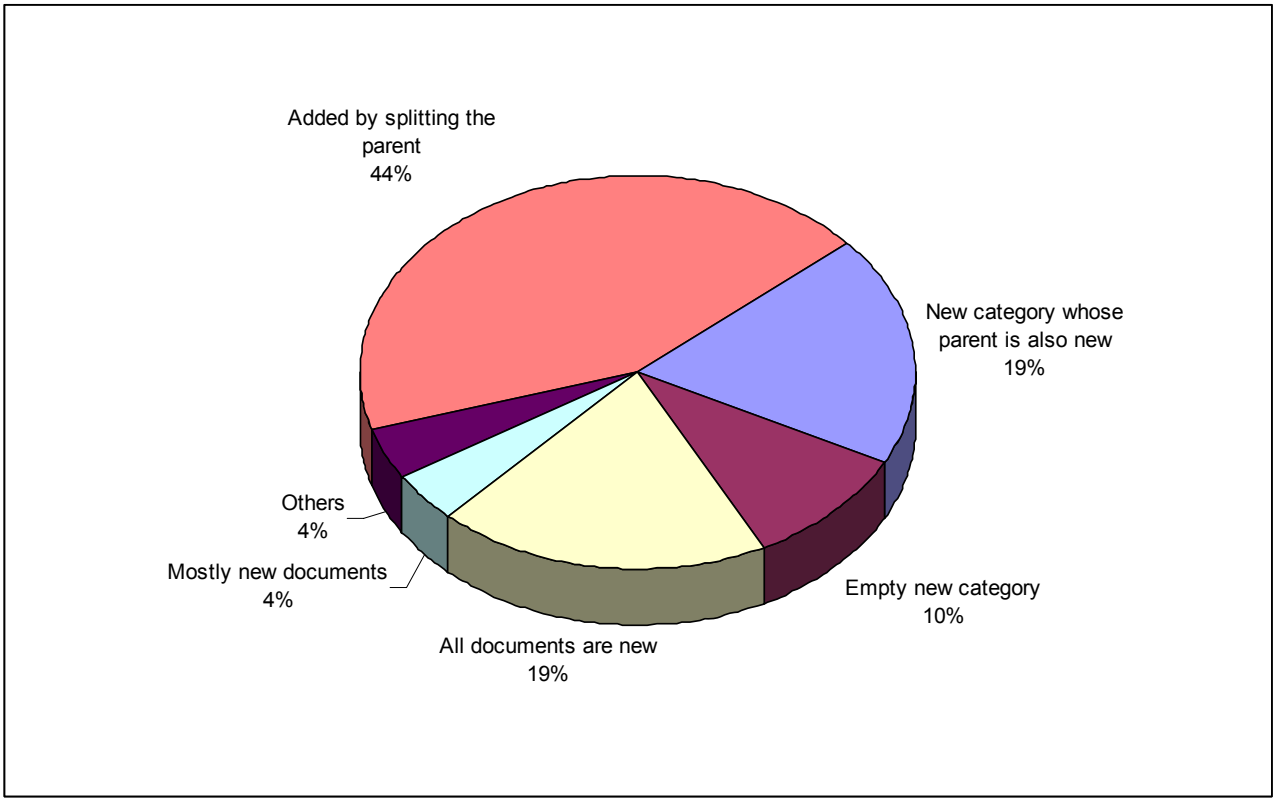

Figure 2: Frequency of various types of category additions.

2.6 Prediction of category additions as a learning problem

One can treat the problem of predicting category additions as a machine learning problem. Each example of the learning problem consists of a category and a point in time; the question to be answered is whether a new subcategory should be created below the given category at the given point in time. Thus, this is a binary (two-class) classification problem, with the positive class consisting of those examples where the addition of a child category is necessary, and the negative class consisting of those where it isn't.

The main open question at this point is how to describe each example by a set of features (or attributes) such that the resulting representation will be suitable as an input for a machine learning algorithm. The features should contain information that is relevant for making a decision whether a subcategory is needed or not. As discussed in Section 2.4, we ignore those additions of subcategories that are clearly based on background knowledge external to the ontology itself; the remaining additions must therefore be based at least partly on the actual contents of the ontology, i.e. the documents in the category below which a new subcategory is going to be added. Our approach is based on the idea that the human editors of the ODP probably suggest the addition of a new subcategory when they notice, within an existing category *C*, a few documents dealing with a reasonably well-defined narrower subtopic of the general topic of *C*. In this case a new subcategory would be added as a child of *C*, and the documents dealing with the subtopic thus identified would be moved into the new subcategory (whereas they had previously resided in *C* or possibly in one of its descendants). Since these documents all deal with a relatively narrow subtopic, one would hope that they are closely related to one another, use similar terminology, etc. If we represent them as points in a multidimensional space, we would expect to find them relatively closely together, closer than the average distance over all documents from *C* (which, covering a somewhat wider topic, would be expected to be dispersed more widely in space). To express this using data mining terminology: we would expect the documents of the new subcategory to form a cluster within the set of all documents of the parent category *C*. Thus, we turn to clustering as a technique that will help us assess whether such subsets of tightly related documents actually exist.

2.6.1 Representing documents with the bag-of-words model

We begin by representing each document by a vector using the vector space model (also known as “bag of words”), which is an established approach in information retrieval. The vector **x** representing a document *d* contains as many components as there are different words in our dataset. (This means that the number of components is very large, but fortunately the vector is very sparse: for any particular document *d*, most of the words don’t appear in it, and thus most of the components of its corresponding vector are 0.) For a word *w*, the vector **x** that represents a document d contains a component

$$x_w = TF(w, d) \cdot IDF(w).$$

Here, $TF(w, d)$ is the *term frequency* of *w* in *d*, i.e. the number of occurrences of the word *w* in the document *d*. $IDF(w)$ is the inverse document frequency of the word *w*, and is defined as

$$IDF(w) = \log(DF(w) / N),$$

where $DF(w)$ is the document frequency of *w*, i.e. the number of documents that contain *w*, and *N* is the total number of all documents. (The purpose of IDF is to reduce the influence of words that appear in a large number of different documents, on the assumption that such words are not useful for distinguishing between documents.) Finally, the vector **x** is divided by its own length, so that henceforth it has a Euclidean length of 1; this is useful because we want the feature vector **x** to reflect the subject of a document, rather than its length.

The bag of words representation provides a useful and convenient way of measuring how closely related two documents are: given the vectors **x** and **y** representing two documents, we can compute the cosine of the angle between them:

$$\cos(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y} / (|\mathbf{x}| \cdot |\mathbf{y}|),$$

which is simply the same as the dot product $\mathbf{x}^T\mathbf{y}$ if the two vectors are normalized to unit length ($|\mathbf{x}| = |\mathbf{y}| = 1$). The more closely related the subject of the two documents is, the smaller the angle between their vectors **x** and **y** is likely to be, and the higher the cosine will be.

2.6.2 Clustering

Consider the set of all documents that have been assigned to a category *C* or any of its descendants. This is the set within which we would like to find any tightly coupled cluster; this would help us decide whether any new subcategories should be introduced below *C*.

We will use the well-known *k*-means clustering algorithm to cluster our set of documents. The variant we use is known as hierarchical 2-means clustering (also known as bisecting *k*-means [7]): begin by treating the whole set of documents as a single cluster; then, at each step, select a cluster and split it into two. We use the following termination criteria: we stop the process when there are 10 clusters, we do not try to split clusters containing less than 5 documents, nor do we split a cluster if it turns out that one of its two resulting subclusters would contain just one document.

For a cluster *A*, we define its *centroid*, which is simply the average of its elements, normalized to unit length:

$$centroid(A) = \Sigma_{\mathbf{x} \in A}\, \mathbf{x} \,/\, |\Sigma_{\mathbf{x} \in A}\, \mathbf{x}|,$$

and the variance, which is simply the average distance between the documents in the cluster and its centroid. Since the cosine measures similarity, we define the distance as 1 – cosine:

$$var(A) = (1/|A|)\, \Sigma_{\mathbf{x} \in A}\, [1 - \cos(\mathbf{x}, centroid(A))].$$

During clustering, we select, as the next cluster to be split, the one that has the greatest variance.

To split a cluster *A* into two subclusters A_1 and A_2, the 2-means algorithm takes the following steps:

1. Perform an initial partition of *A* into two subclusters, A_1 and A_2.
2. Let $\mathbf{c}_1 := centroid(A_1)$ and $\mathbf{c}_2 := centroid(A_2)$.
3. Let $B_1 := \{\mathbf{x} \in A : \cos(\mathbf{x}, \mathbf{c}_1) > \cos(\mathbf{x}, \mathbf{c}_2)\}$ and $B_2 := A - B_1$.
4. Let $A_1 := B_1$ and $A_2 := B_2$.
5. Repeat steps 2 through 4 until the termination criterion is met.

The termination criterion is usually defined to stop when a certain number of iterations has been performed, or when the number of reassignments (i.e. $|A_1 \cap B_2| + |A_2 \cap B_1|$, which is the number of documents that were moved from the first subcluster to the second one or vice versa) drops below a threshold. Our criterion was to stop after five iterations or if there were no reassignments in the last iteration.

Various ways have been proposed to define the initial partition of *A* into A_1 and A_2 (in Step 1) [6]. For example, one may partition the same way as in Step 3, but using two randomly selected documents as $\mathbf{c}_1$ and $\mathbf{c}_2$. However, a slightly more sophisticated approach that often works better is to use principal component analysis (PCA) to find the first principal component of *A*, i.e. the direction in which the cloud of points in *A* exhibits the maximum variance. Let $\mathbf{p}$ be a unit-length vector in this direction; then we see that

$$\mathbf{x} = (\mathbf{x}^T\mathbf{p})\, \mathbf{p} + \mathbf{y}, \text{ for a certain } \mathbf{y} \text{ that is orthogonal to } \mathbf{p}\text{: } \mathbf{y}^T\mathbf{p} = 0.$$

Thus, $\mathbf{x}^T\mathbf{p}$ is the orthogonal projection of $\mathbf{x}$ onto $\mathbf{p}$. We can then compute the average of the projections of all points from *A*:

$$\mu = (1/|A|)\, \Sigma_{\mathbf{x} \in A}\, \mathbf{x}^T\mathbf{p},$$

and we can define an initial partition of *A* into two subclusters as

$$A_1 = \{\mathbf{x} \in A : \mathbf{x}^T\mathbf{p} < \mu\} \text{ and } A_2 = A - A_1.$$

It often turns out that this initial partition is already quite good and very few reassignments are performed in the subsequent iteration(s) of the 2-means algorithm.

2.6.3 From the clustering to a feature vector for the category

Let A be the initial set of all documents in the category C (and its descendants), and let $P = \{B_1, ..., B_k\}$ be the partition of A into k disjoint clusters obtained using the hierarchical 2-means algorithm. We will use the following features to describe this partition:

(1) One feature is the average cosine between each document and the cluster to which it belongs:

$$(1/|A|) \sum_{B \in P} \sum_{\mathbf{x} \in P} \cos(\mathbf{x}, \mathit{centroid}(B)).$$

This is a measure of how tight clusters we have obtained by partitioning the initial set A into k clusters.

(2) We find the cluster with minimum variance:

$$B = \arg\min_{B' \in P} \mathit{var}(B').$$

We use, as features, the following properties of this cluster:

- The size of this cluster. Instead of using $|B|$ directly, we use $\log |B|$, to prevent large clusters from having an excessive influence on the range of this feature.
- The relative size of this cluster, i.e. $|B| / |A|$.
- The variance of this cluster, $\mathrm{var}(B)$.
- The variance of this cluster, relative to that of the whole set: $\mathrm{var}(B) / \mathrm{var}(A)$.

(3) We find the cluster with the maximum average intra-cluster similarity:

$$B = \arg\max_{B' \in P} \sum_{\mathbf{x}, \mathbf{y} \in B'; \mathbf{x} \neq \mathbf{y}} \cos(\mathbf{x}, \mathbf{y}) / (|B'| \cdot (|B'| - 1)).$$

For this cluster B, we use four features analogous to those described above in (2) for the minimum-variance cluster. The idea here is that the average intra-cluster similarity is another measure of cluster compactness, and these features may therefore help the classifier identify categories with a compact subset of documents that would be a suitable basis for creating a new subcategory.

In this way we have described the partition P by nine features. Every time that our hierarchical clustering algorithm splits a cluster, the partition changes (one of its clusters gets replaced by two smaller ones), and we add, to the feature vector for the category under observation, the nine features describing the new partition. We let the clustering continue until there are ten clusters, which means that in the end the category is described by a 90-dimensional feature vector. (It is possible for the clustering algorithm to stop before ten clusters have been obtained, due to the other termination criteria described in Section 2.6.2. In this case we repeat the features of the final partition as many times as necessary to bring the feature vector to the full 90-dimensional form.)

2.6.4 Training a classifier

We decided to use the support vector machine (SVM [4]) learning algorithm to train classifiers for this learning problem. The SVM is a state-of-the-art learning method that has been found to perform well in many areas, including on tasks with a considerable number of features and training examples.

The scoring function trained by the SVM algorithm has the form

$$f(\mathbf{x}) = b + \Sigma_{i=1..n}\, \alpha_i\, y_i\, K(\mathbf{x}_i, \mathbf{x}),$$

where $\mathbf{x}_i$ is the i-th training example, y_i is the label of the i-th training example (y_i = +1 for positive examples and –1 for negative ones), and K is the kernel function, which must define before the training can begin; it can be any function that satisfies certain mathematical criteria. The values α_1, ..., α_n and b are the output of the learning algorithm. The values of $f(\mathbf{x})$ can be used to rank examples (the higher the $f(\mathbf{x})$, the more likely $\mathbf{x}$ is to be positive); or, to obtain binary predictions, one would predict all examples $\mathbf{x}$ for which $f(\mathbf{x})$ exceeds a certain threshold (usually 0) as positive, and all other examples as negative.

Since the feature space used in our representation is relatively modest (90 features, as opposed to e.g. thousands of features as is commonly the case in text and image categorization settings), we decided to use the radial basis function (RBF) kernel:

$$K(\mathbf{x}, \mathbf{x}_i) = \exp[-\gamma\, ||\mathbf{x} - \mathbf{x}_i||^2]$$

Here, $\gamma > 0$ is a constant parameter that we must select before starting the SVM training algorithm. Note that, if we think of it as a function of $\mathbf{x}$ while $\mathbf{x}_i$ is held constant, this is a bell-shaped function that has a maximum of 1 at $\mathbf{x} = \mathbf{x}_i$, and elsewhere it decreases towards 0 as $\mathbf{x}$ moves further away from $\mathbf{x}_i$. The parameter γ influences the breadth of the bell of this bell-shaped function. The effect of this kernel in the formula for $f(\mathbf{x})$ is that each training example $\mathbf{x}_i$ votes for its own class (y_i) with the weight α_i, but its influence decreases as the distance of $\mathbf{x}$ from $\mathbf{x}_i$ increases. Thus, the SVM model with this type of kernel effectively becomes a softer version of a nearest-neighbour classifier, but one in which the influence of the neighbours has been carefully selected by the learning algorithm (via the values of α_i). With this class of models, having a small number of features is not necessarily a problem as long as we have a sufficient quantity of training examples.

3. Experimental evaluation

3.1 The dataset

In this section we describe our experimental evaluation of the proposed approach for the prediction of category additions. We used the *Computers* subtree of the Open Directory Project ontology. In the period under consideration, i.e. from January 2004 through October 2006 (there being no snapshots of the ODP from November and December 2006), the Computers subtree grew from 7,732 categories to 8,309 categories, while the number of documents on average tended to decrease rather than increase, eventually shrinking from 143,760 documents in January 2004 to 133,595 documents in October 2006.

During this period, there were 964 category additions, 198 category renames, 134 category moves, and 153 merges of various types. See Figure 1 for a chart showing the number of different operations in each month.

Of the category additions, 482 were such that the new category is added as the child of a previously existing parent category and more documents have been moved into the new category from the parent (or its previously existing descendants) than from other parts of the hierarchy. This, as described in Section 2.4, is the type of additions that we will be trying to predict. However, it turns out that even in these cases, the number of documents moved from the parent to the new child category is often quite small. The hypothesis underlying our approach to the prediction of category addition is that the human editors of the ODP notice, in an existing category, a group of documents dealing with some narrower subtopic and then decide to create a new subcategory and move those documents into it. This hypothesis means that our approach can not be reasonably expected to perform well in situations where only e.g. one or two documents have been moved from the parent to the new child, since in this case there is effectively no subgroup of closely related documents that could have been detected in the old parent category (and then be used to predict an addition). Therefore, for the purposes of defining our classification problem, we limit ourselves to the additions of categories in which at least five documents were moved from the parent category into the new child category (in addition to that, some documents may also have been moved into the new child from the parent's descendants or from entirely different parts of the ontology, and some documents completely new to the ontology may also have been added; but we do not set any additional constraints regarding the number of such documents). This leaves us with 107 category additions as the basis for our prediction task.

The question that our predictive model will attempt to answer is this: "given a category, should any new subcategories be added below it, as its children, during the next month?" Since there are two possible answers to this question, yes or no, this will be a binary (i.e. two-class) classification problem. A category at a given point in time is a positive example if some children (matching the criteria described above) have indeed been added to it between that point and the next point in time for which an ontology snapshot is available (i.e. approximately one month later). According to this definition, the above-mentioned 107 additions give rise to 98 positive examples (this is less than the number of additions because sometimes several children are added to the same parent in a certain month).

But when is a category a negative example? For example, suppose that a comparison of the snapshots for March 2005 and April 2005 shows that no suitable children have been added to category *C* in the intervening period, but the comparison of the snapshots for April 2005 and May 2005 shows one such addition. This suggests that the category *C* such as it was in April 2005 is a positive example for the purposes of our machine learning problem; but is it reasonable to say that *C* such as it was in March 2005 is a negative example, just because no additions were made to it between March and April? After all, the additions to the ODP ontology are handled by human editors, many of whom look after a number of different categories and may overlook something. The category *C* has not necessarily changed much from March to April; perhaps our editor would have already made the addition to *C* in March rather than in April, but he or she simply hadn't

noticed that there exists a compact subgroup of documents that calls for the introduction of a new subcategory. Therefore, to avoid having an excessively narrow definition of the negative set, we declare a category to be negative at a certain point in time only if no suitable children have been added to it at that point or in the preceding or following three months. Despite this constraint, the vast majority of categories are treated as negative examples at any particular point in time, since the category additions are rare relative to the total number of categories. In total, we could obtain more than 168,000 negative examples from the Computers subtree in the period 2004–2006. To speed up the experiments and to prevent the positive examples from being completely overwhelmed by the negative ones during the training process, we randomly selected three times as many negative examples as there are positive examples. We then divided the resulting data into a training set (all examples from the years 2004 and 2005) and a test set (all examples from the year 2006). Thus, we end up with a training set containing 74 positive and 222 negative examples, and a test set containing 24 positive and 72 negative examples.

3.2 Experimental setup

As has been discussed in 2.6.4, we will be using the SVM algorithm to train classifiers. We use the SVM^{light} implementation of SVM by Thorsten Joachims [5]. The SVM finds a classifier by solving an optimization problem that maximizes a combination of two goals: the classifier should have a wide margin, and it should make as few mistakes on the training set as possible. These two goals are combined via an error cost parameter, traditionally denoted by C. The greater the C, the more effort the learner will place on avoiding errors on the training set; on the other hand, this leads to a greater risk of overfitting the training data.

For relatively unbalanced datasets, i.e. those where the positive examples are heavily outnumbered by the negative ones, it is often beneficial to treat errors on positive training examples as more problematic than those on negative training examples. Thus, one effectively uses two different error costs: the baseline cost C on the negative examples and its multiple $j \cdot C$ on the positive examples.

Thus, C and j are two tunable parameters which we will select via five-fold cross-validation on the training set. A third tunable parameter is γ, the width of the Gaussian functions in the RBF kernel (see Section 2.6.4). We tested the following parameter values: $C \in \{0.1, 1, 10, 100, 1000\}$; $j \in \{1, 2, 3, 5, 10, 20, 50, 100\}$; and $\gamma \in \{0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5\}$. For most combinations of these parameter settings, training a model cost less than a second, so that most of the time was actually spent on generating the features.

3.3 Evaluation measures

To evaluate the output of the classifiers, we use well-known evaluation measures from the area of information retrieval [2]: the breakeven point and the area under the ROC curve.

A classifier, via its outputs $f(\mathbf{x})$ as described in Section 2.6.4, effectively introduces a ranking of the examples from the test set (or whatever other set of examples it is used on). By selecting a threshold and predicting all the examples above the threshold as positive and all those below the threshold as negative, we can divide the examples into four groups: true positives, false positives, true negatives, and false negatives. These can be arranged in a *contingency table*:

True class	Predicted class	
	Positive	Negative
Positive	TP (true positives)	FN (false negatives)
Negative	FP (false positives)	TN (true negatives)

The following useful evaluation measures can be computed from the contingency table:

$precision = TP / (TP + FP)$
$recall = TP / (TP + TN)$ (also known as "the TP rate")
$FP_{rate} = FP / (FP + TN)$

If we slowly decrease the threshold, more and more examples get predicted as positive; *TP* and *FP* grow, while *TN* and *FN* decrease. As a result the recall increases, while the precision tends to decrease (although not necessarily in a monotonic way). The point where precision and recall are equal is known as the *breakeven point* (BEP), and is a useful evaluation measure [3]. It indicates what sort of tradeoffs between precision and recall are possible with the ranking produced by the given predictive model. To have a precision greater than the BEP, we will have to accept the fact that the recall will be less than the BEP; on the other hand, to have recall greater than the BEP, we will have to accept a precision that will be less than the BEP. Thus, the higher the BEP, the better our ranking is because it doesn't force us into compromises as uncomfortable as a ranking with a low BEP.

As an alternative to the precision and recall, one can use the TP rate and the FP rate. These are typically plotted on a graph, the FP rate on the horizontal axis and the TP rate on the vertical axis. As we decrease the threshold, both the TP rate and the FP rate grow, starting at $TP_{rate} = FP_{rate} = 0$ (when nothing is predicted to be positive) and ending at $TP_{rate} = FP_{rate} = 1$ (when everything is predicted to be positive). This produces a monotonically rising curve, known as the ROC curve (receiver operating characteristic) [1]. The area under the ROC curve is another succinct way to summarize the quality of the predictive model. This area turns out to be equal to the probability that, given a randomly chosen positive example and a randomly chosen negative example, the model would assign a higher score $f(\mathbf{x})$ to the positive example than to the negative one. A model that always gets this right would achieve an area under ROC equal to 1.

As a baseline, a model that ranked the examples in random order would achieve a breakeven point equal to the proportion of the positive examples relative to all examples (which is 0.25 for our dataset), and the area under its ROC curve would be 0.5. A perfect model would achieve a score of 1 according to both measures.

3.4 Results

We used stratified 5-fold cross-validation (CV) on the training set to investigate the influence of *C*, *j*, and γ parameters. (Stratified means that when dividing the documents into five folds, care has been taken to ensure that the proportion of positive documents in each fold is roughly the same as the proportion of positive documents in the dataset as a whole. This is particularly important in cases when the overall proportion of positive documents is quite low, as it indeed is in our dataset.) As described in Section 3.2, we investigated 5 values of *C*, 8 values of *j* and 15 values of γ. This results in 600 combinations of parameter settings. We then select the combination that performed the best during cross-validation on the training set; using this combination of parameter settings, we train the final model on the entire training set, and this model would then be evaluated on the test set. The results are summarized in the following table:

Table 1. Performance of models selected with various model selection criteria.

Model description	Performance on the validation set during 5-fold cross-validation		Performance on the test set	
	BEP	A. u. ROC	BEP	A. u. ROC
Highest BEP during CV	0.5148	0.7796	0.7083	0.8893
Highest a.u.ROC during CV	0.5021	0.7850	0.6667	0.8738
Highest BEP on the test set	0.4717	0.7436	0.7500	0.9011
Highest a.u.ROC on the test set	0.4768	0.7495	0.7500	0.9155
Random ranking	0.2500	0.5000	0.2500	0.5000

The first row, “highest BEP during CV”, refers to the models having the greatest breakeven point during cross-validation. There were three models (i.e. three different combinations of parameter settings) with the maximum BEP here, so the other columns of the table show average performance over these three models. The same approach has been used in the other rows.

The rows referring to the highest BEP/a.u.ROC on test set indicate what the best models among those tested here are capable of, with the caveat that we aren’t able to identify these models without peeking at the test data. Comparing these results with the results from the first two rows tells us how much room for improvement there is if we can select our models using some better criterion than cross-validation on the training set. We can see that the difference is not really very large here, and by selecting our models through cross-validation we obtain models that also perform quite well on the test set.

For comparison, the last row of the table shows the performance of a hypothetical model that doesn’t learn anything and instead just outputs random scores for all the examples. The values in this row can be derived from the properties of the dataset and the formulas in Section 3.3.

(The fact that the performance on the test set is better than the one during cross-validation is probably due to the fact that only 80% of the training set are used to train each model during cross-validation – the remaining 20% are used as the held-out validation set. On the other hand, the models that were evaluated on the test set were trained on the entire training set, so it is reasonable that they perform better.)

3.4.1 Model selection criteria

An interesting question at this point is whether it's better to use the maximum BEP or the maximum area under ROC as a criterion when selecting the parameter setting. The following figure shows a dot for each of the 600 models obtained by training an SVM under different parameter setting; the BEP is the x-coordinate, and the a.u.ROC is the y-coordinate. As we can see, the two measures are really fairly closely correlated, especially among the best models. In a situation with relatively few positive test examples, the BEP has a relatively small set of possible values and thus it is perhaps a less than ideal measure for evaluation of the performance on the test set.

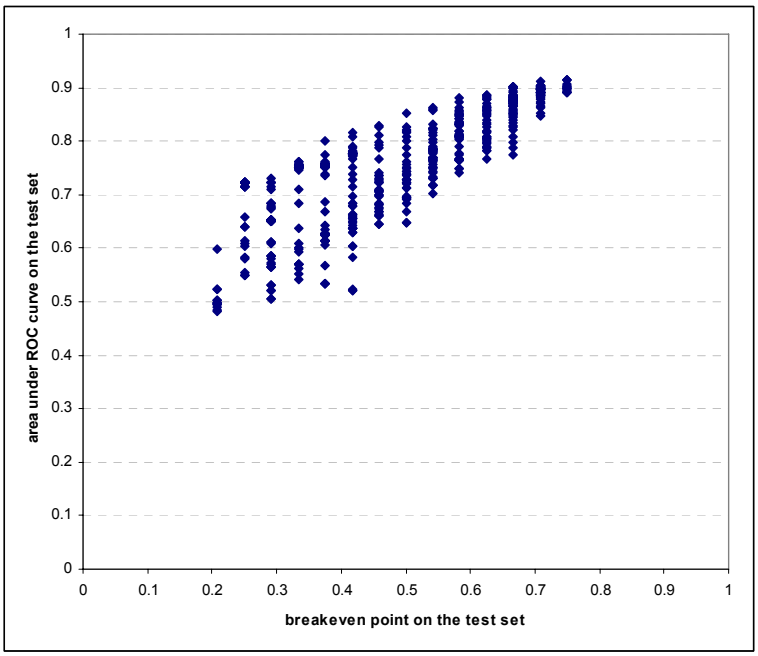

Figure 3: Breakeven point and the area under ROC curve for 600 SVM classifiers obtained under different parameter settings.

It is true that in the experiments presented in the table above, selecting the parameter settings that maximize the BEP during the cross-validation resulted in models that performed slightly better on the test set than if the parameter settings were selected by maximizing the a.u.ROC during cross-validation. However, our impression is that it would be unreasonable to conclude that this should be generally the case in problems of this type; further experiments with more extensive data sets would be required before this could be confirmed one way or another.

3.4.2 Parameter tuning

Until now we have been looking for the best combination of parameter settings by allowing all three parameters to vary – C, j, as well as γ. But what if we hold one of these parameters fixed at some specific value and then examine only the models obtained by varying the other two parameters? The following charts show the results. For each value of each parameter, we select the other two parameters so as to maximize the a.u.ROC measure during cross-validation. We then report this a.u.ROC value, as well as the a.u.ROC achieved by the same combination of parameters on the test set.

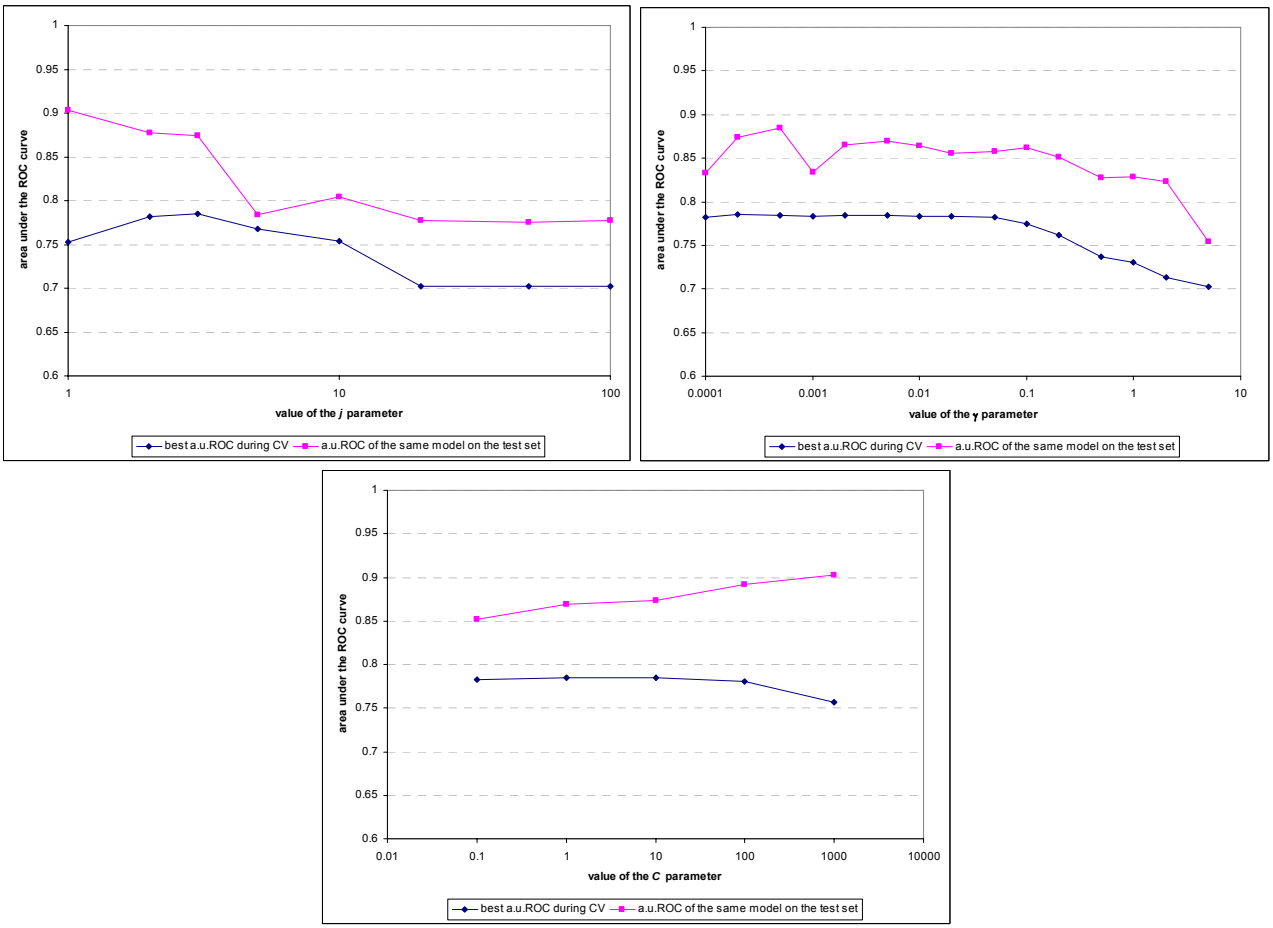

Figure 4: Classification performance as a function of one parameter (*j*, γ or *C*) if the other two parameters are tuned using fivefold cross-validation.

Regarding the *j* parameter, the best results were achieved with *j* = 3, which is reasonable since this is exactly the ratio of the number of negative examples to the positive ones. Since there are three times as many negative examples as there are positive ones, it makes intuitive sense to treat errors on the positive examples as three times more serious than the errors on negative examples. This intuition is borne out by the facts, and it turns out that the farther away *j* was from 3 (either greater or smaller), the worse the performance of the resulting models was.

Observations regarding *C* and γ are less useful because, when training SVM models, the optimal values of these parameters depend considerably on the properties of the dataset, i.e. the number of features, the domains of the features, and *C* is further influenced by the type of kernel and its parameters (e.g. γ in our case). In general, we can say that in both cases (*C* as well as γ) there is actually a fairly broad range of parameter values where good performance can be achieved. The results for *C* are somewhat surprising because there, cross-validation appears to mislead us: the larger values of *C* lead to poorer performance during cross-validation but actually better performance on the test set.

4. Conclusions and Future Work

In this deliverable we have described an approach for predicting a subset of structural changes in an ontology. Our approach aims to predict the addition of categories within a hierarchy of documents, under the assumption that the new category is a child of an existing parent category and that it contains at least a few documents that were formerly members of the parent category. We have described how this task can be formulated as a machine learning problem and presented experiments that show that the prediction of this type of changes is feasible.

There are several directions along which this work could be extended. A possible next step is to try to devise more features with which to describe a category (in addition to the clustering-based features that we use currently). This could improve the performance of our current machine-learning approach to the prediction of category additions. In particular, our current features are based in the idea of finding clusters of documents within the set of documents of an existing category. However, some of these clusters may already correspond nicely to the existing descendants of this category, and so they should not be taken as evidence that yet another subcategory needs to be added.

Another interesting extension would be to try predicting not just whether the addition of a new category is warranted, but also which documents it should include, and perhaps which keywords it should be described by. This would be in effect a second step, after the machine learning classifier has suggested an addition. At that point we could look for clusters in the existing set of documents of the parent category, find those that do not overlap well enough with any of the existing descendants of that parent category, and propose them as new children.

In addition to this, it would be interesting to address other types of ontology changes. Currently, we have addressed the addition of categories, and even that only under certain conditions. Some of the additions that are not handled now include those where the new category contains few (if any) documents from the old ontology, but contains some entirely new documents instead. This scenario could be addressed by looking at the set of documents added to the ontology between one snapshot and the next; one could take those documents that do not fit well enough into any of the existing categories; by clustering these documents, one could attempt to identify candidates for new categories containing these new documents.

Moving and merging of existing categories, which is the other major class of structural changes that we have observed in the ODP ontology, could be addressed by comparing each category to other categories, e.g. via the cosine between centroids or (more likely) via some more sophisticated measures. This would indicate whether it matches it current parent well enough or whether it should be moved somewhere else.

Finally, although this is not strictly speaking a *structural* operation, it would be interesting to also address the operation of category renaming, since this is also a fairly common operation in the ODP ontology. Here one possibility would be to employ a term-level approach: describe the role of each term within a certain category by a set of features and then build a predictive model that will tell if this term would be suitable as one of the keywords in the name of the category. The features would need to look at not just the occurrence of this term within the documents of that category, but also within its children, siblings, and parent – the name of a category must be able to differentiate it from all these neighbours. This model could be used to evaluate all the terms with respect to their suitability as keywords in the name of a given category; then, if the terms that are actually used in the current name of the category receive a much lower score than some other terms, it would be a good moment to recommend that the ontology be renamed. However, a significant downside of this approach is that the name of a category, especially a higher-level category, may contain more abstract terms that are not necessarily very prominent in the actual documents contained within that category (and its descendants). It might be necessary to use WordNet or some other similar resource to find the connection between such abstract terms and the words that actually occur in the documents of the ontology.

Another interesting direction for future work would be to consider the problem of structural change prediction in semantically richer ontologies. Our present approach assumes a very simple ontology, consisting of a hierarchy of classes, with zero or more instances assigned to each class. But an ontology may also contain other types of relations, attributes, logical statements and axioms describing the instances and the classes, and so on. This simultaneously opens up a much wider space of possible structural changes and makes the problem of automatic prediction of such changes more difficult. It remains to be seen to what extent approaches along the lines of the work presented in this deliverable can be useful when dealing with structural change in more expressive ontologies.

5. References

[1] F. Provost, T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203-231, March 2001.

[2] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979

[3] D. D. Lewis. *Representation and Learning in Information Retrieval*. Ph.D. Thesis, Univ. of Massachusetts, Amherst, MA, USA, 1991.

[4] C. Cortes, V. Vapnik. Support-Vector Networks. *Machine Learning* 20(3):273-297, September 1995.

[5] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Proceedings of the 10th European Conference on Machine Learning (ECML-98)*, Chemnitz, Germany, April 21-23, 1998, pp. 137-42.

[6] T. Su, J. G. Dy. In search of deterministic methods for initializing K-means and Gaussian mixture clusterind. *Intelligent Data Analysis* 11(4):319-338, 2007.

[7] M. Steinbach, G. Karypis, V. Kumar. A comparison of document clustering techniques. *Proceedings of the 6th KDD Workshop on Text Mining*, Boston, MA, USA, August 20-23, 2000.

[8] P. Cimiano, J. Völker. Text2onto - a framework for ontology learning and data-driven change discovery. *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005)*.

[9] P. Haase, Y. Sure, J. Völker. Management of dynamic knowledge. *Journal of Knowledge Management*, 9(5):97-107, 2005.

[10] L. Stojanović. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe, 2004.

[11] J. Völker, Y. Sure. *Data-driven change discovery*. Deliverable 3.3.1, SEKT Project (EU IST-2003-506826). July 22, 2005.

[12] A. Maedche, B. Motik, L. Stojanovic, R. Studer, R. Volz. Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems*, January/February 2003.

[13] G. Flouris, D. Plexousakis, G. Anto. A Classification of Ontology Change. *Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop*, Pisa, Italy, December 18-20, 2006.

[14] D. Maynard, W. Peters, M. d'Aquin, M. Sabou, N. Aswani. *Dynamics of Metadata*. Deliverable 1.5.1, NeOn Project (EU IST-2005-027595). March 30, 2007.