Using DMoz for Constructing Ontology from Data Stream

Marko Grobelnik, Janez Brank, Dunja Mladenić, Blaž Novak, Blaž Fortuna
Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
{firstName.secondName}@ijs.si

Abstract. *This paper presents an approach for constructing an ontology from a stream of documents. Named entities extracted from the documents are used as instances of the ontology. Entities and co-occurring entity pairs are represented by feature vectors based on the content of the documents where they occurred. In general, concepts and relations can be formed into an ontological structure either by clustering or by classification into an existing topic hierarchy. We propose the latter using DMoz as an existing topic hierarchy. The approach is efficient and can scale to large data sets. We propose a framework that incorporates the stream mining process into a formal definition of the ontology. We describe a software component implementing this approach, and present experiments using a large collection of news.*

Keywords. Stream mining, stream ontologies, clustering of documents, Web directory.

1. Introduction

The data stream model [3,6] is rapidly becoming an important paradigm in data processing. With the developments in computing hardware it has become increasingly easy to generate and transmit enormous amounts of data in the form of continuous streams of measurements, event logs and other massive datasets. In order for this information to be of any use, a new generation of algorithms is needed to process them and extract the relevant knowledge from them while at the same time observing all of the constraints presented by such a dataset. For instance, having a stream of business news we may want to observe changes in the context in which two companies are mentioned over a specific time period.

The main characteristic properties of a data stream are the (theoretically) infinite amount of data ordered in a sequence and the fact that the source of the data may not be stationary - it might not be possible to describe the entire stream with the same model. The stream might be incurring gradual changes known as concept drift as well as completely changing its properties. Detecting and tracking such changes in the data is an important research topic.

One of the simplest approaches to dealing with data sequences whose properties are likely to change in time is to only observe a small set of nearby data points at a time, called a window. As new data becomes available it is added into the window, while the oldest data is discarded from consideration. In this way a model of the current state can easily be maintained using conventional algorithms that are not designed to work with infinite data streams, however no knowledge of the past beyond the window exists. This problem can be alleviated by maintaining a summary of the data that has been shifted out of the window. Most algorithms use a fixed window size due to a simpler implementation, but adaptive time window has also been used for learning in the presence of concept drift where the basic idea is to adjust the size of the window to the current extent of the concept drift; an example being the FLORA framework in [7].

It is apparent that in order to be able to process a stream in real time, only a constant amount of time and even less space may be used per each incoming data point. It must also be possible to generate and maintain some sort of a model or conceptualization in an online, incremental fashion, possibly observing and noting its changes through time. Because of the space constraint, almost any non-trivial task can only be solved approximately and every data point can only be stored in the memory for a limited amount of time. Stream mining algorithms are therefore necessarily single-pass (while allowing for a limited number of past data points to be kept available and accessed more than one time).

While this approach is mainly intended for data that comes naturally as a stream, it can also be applied where a finite dataset must be processed but the conventional algorithms would take a prohibitively long time and/or space and it is sufficient to only find an approximate solution.

Some examples of data streams are system event logs, sensor measurements, urban traffic logs, telephone call records, retail store transactions, ATM and credit card transaction logs, stock tickers, webserver logs, internet traffic logs, etc. For instance, AT&T collects 100GB of NetFlow (Cisco Systems' network accounting) data each day, which can obviously not be processed using conventional algorithms.

In this paper we address the problem of construction of ontologies from data streams using machine learning, which is to the best of our knowledge the first attempt in this direction. In Section 2 we describe the proposed approach, while architecture of the prototype system is given in Section 3. Section 4 gives conclusions and some directions for further development.

2. Approach description

We rely on a formal definition of ontologies that is a slight extension of the definition proposed in [2, 1]. In this formalization, the ontology (with datatypes) is defined as a structure $O = (C, T, R, A, I, V, \leq_C, \leq_T, \sigma_R, \sigma_A, \iota_C, \iota_T, \iota_R, \iota_A)$. It consists of (disjoint) sets of concepts (C), types (T), relations (R), attributes (A), instances (I) and values (V). The partial orders $\leq_C$ (on C) and $\leq_T$ (on T) define a concept hierarchy and a type hierarchy. The function σ_R: $R \to C^2$ provides relation signatures (i.e. concepts that may be linked by relation), while σ_A: $A \to C \times T$ provides attribute signatures (for each attribute, the function specifies to which concept the attribute belongs and what is its datatype). Finally, there are partial instantiation functions ι_C: $C \to 2^I$ (the assignment of instances to concepts), ι_T: $T \to 2^V$ (the assignment of values to types), ι_R: $R \to 2^{I \times I}$ (which instances are related by a particular relation), and ι_A: $A \to 2^{I \times V}$ (the value of each attribute for each instance).

The formalization shown above was originally defined for a static ontology. In our case, however, we will be dealing with a stream ontology that changes with time. Therefore, we will use the superscript ^{t} to refer to the state of the ontology at time t.

2.1. Stream of documents

For the purposes of formalizing a stream ontology, we begin by introducing a stream of documents. This is a sequence $((t_1, D_1), (t_2, D_2), \ldots)$ of pairs, where each D_k is a set of documents and t_k is a timestamp for these documents. In principle, each D_k could contain a single document, but in practice it is often desirable to process documents in batches of several documents that are treated as having arrived at the same time. The batches of documents are given in increasing order of timestamps, i.e. $t_k < t_{k+1}$ for all k. $TS(d)$ is the timestamp of d.

Each document d is characterized by a feature vector $F^t(d)$ from some feature space. Typically, a bag-of-words representation would be used, such that the feature space is a multidimensional real space containing one dimension for each word that may occur in a document. $F^t(d)$ could then be a TF-IDF vector or some similar representation known from the field of information retrieval.

We use the notation F^t to emphasize the fact that this representation of the document could in principle change with time. For example, if a TF-IDF vector is used, the IDF part depends on the proportion of documents that contain a particular term. As this proportion can change with time, to take more recent documents into account, the TF-IDF vector of an old document can slowly change as well. However, in practice, once a sufficient number of documents has been processed, the IDFs may be expected to be fairly stable, so in practice one may consider the feature vectors $F^t(d)$ as independent of t.

2.2. Instances and instance profiles

The document d is characterized by a set of instances, $I(d) \subseteq I$, where I is the full set of instances for the problem domain we are dealing with. In practice, $I(d)$ could be a set of named entities extracted from the text of the document d. Notice that we deal with named entities rather then a complete set of terms to limit the set of possible instances while capturing the meaning. However, our approach is not conditioned on that and can be used for instances defined in different way eg., taking all noun phrases. The occurrence of an instance $e \in I(d)$ in the document d is characterized by an occurrence vector $F^t(e,d)$. We can define the profile of e at the time t as a weighted average as follows.

$$P^t(e) = \left(\sum\nolimits_{d \in D^t(e)} w(TS(d),t) F(e,d)\right) \Big/ \left(\sum\nolimits_{d \in D^t(e)} w(TS(d),t)\right)$$

Each occurrence of e in some document d contributes the vector $F^t(e,d)$, but the documents are weighted by their timestamps.

2.3. Concepts and the concept hierarchy

Contents should reflect the documents that occurred in the stream up to the time t, i.e. the set of documents D_t. Thus we have the set of instances, containing all instances that occur in some document from D_t. To define a set of concepts C_t and the is-a hierarchy of these concepts, hierarchical clustering over I^t could be used, with each instance $e \in I^t$ represented by its profile $P_t(e)$. This clustering also yields the assignment of instances to concepts.

In practice, several approaches could be used to efficiently obtain a suitable concept hierarchy at each point in time. An initial concept hierarchy could first be obtained by clustering the initial batch of documents D_1 at time t_1. Subsequently, the hierarchy would be updated after the arrival of each new batch of documents (e.g. D_k at time t_k). New documents may introduce new entities and change the profiles of previously known entities. In response to this, the hierarchy is updated by various incremental changes, such as splitting a concept (defining new subconcepts), merging a concept into its parent concept, or deleting a concept.

Clustering typically relies on some distance function on the feature space to determine which instances are ”close together” and should be placed into a common cluster. Thus, several concept hierarchies could be obtained by using different distance functions. Typical distance functions such as those used in information retrieval (e.g., cosine of the angle between two TF-IDF vectors) would most likely result in concept hierarchy that approximately expresses the “is-a-subtopic-of” relation between concepts.

Obtaining a concept hierarchy by clustering leaves partly open the question of naming the concepts. A simple but practical approach is to compute the centroid of the feature vectors of all the documents in a cluster and select a few words that have a high weight in this centroid. It may be advisable to explicitly avoid words that also have a high weight in the parent concept; in this way we can emphasize the characteristics which distinguish a concept from its neighborhood [8].

2.4. Instances co-occurrences

We use instance co-occurrences as the basis for modeling relations between instances. A co-occurrence $(e, \hat{e}) \in J(d)$ is represented by a *co-occurrence context vector* $F^t(e, \hat{e}, d)$.This vector is intended to represent the context in which the instances co-occur. It could be defined as simply equal to $F^t(d)$, or derived in some way. For example, if we require that two instances be mentioned in the same paragraph if they are to be treated as co-occurring, we can define $F^t(e, \hat{e}, d)$ as a vector representing the bag-of-words for the paragraph where the two instances co-occur in *d*.

A *co-occurrence profile* is defined similarly as the instance profile where *e* and $\hat{e}$ co-occur in document $D^t(e, \hat{e}) = \{d \in D^t : (e, \hat{e}) \in J(d)\}$.

2.5. Relations between instances

A relation can then be represented by a function (or, in machine-learning terminology, a model or classifier) that takes a co-occurrence profile of some pair of instances as input and returns a real number indicating to what extent the two instances seem to be related by the relation. That is, we have a predictor such that $f_r(P(e, \hat{e}))$ tends to be larger when the two instances really are in relation *r* and smaller when they are not in relation *r*.

In practice, we expect the definitions of the relations to be relatively stable, so that the function f_r does not change much with time (and does not need to be updated every time a new batch of documents arrives). On the other hand, the membership in the relation may change as new instances appear and the profiles of old instances change. Thus, at time *t*, the relation membership function can be defined as follows.

$$\iota_R^t(r) = \{(e, \hat{e}) \in I^t \times I^t : f_r(P^t(e, \hat{e})) > 0\}$$

In other words, the two instances are in relation *r* at time *t* if the model f_r for that relation assigns a sufficiently high score (greater than 0 in the above formula, although in principle a different threshold could also be used) to the co-occurrence profile of the two instances at time *t*.

An initial set of relations in time t_1 can be obtained by computing the co-occurrence profiles of entity pairs based on the initial batch of documents. These profiles can be clustered, yielding a hierarchy of clusters; each cluster gives rise to a relation and the hierarchy between clusters gives rise to a hierarchy of relations. The model f_r corresponding to a relation *r* can then be any function that can intelligently assess whether a vector belongs to the cluster *r* or not; various text-classification methods can be used for this purpose.

Subsequently, when enough new data arrives, the hierarchy of relations and their models f_r can be updated or built anew, using the most recent co-occurrence profiles.

An alternative approach to defining relations by clustering is to have a human expert provide some examples for each relation. Once some positive and negative examples are available (i.e. pairs of instances that are in this relation, and pairs that aren't), machine learning methods can be used to obtain a corresponding model f_r. Since these methods often require a considerable number of training examples, and annotating such examples manually can be expensive and time-consuming, an approach based on active learning may be advisable: the system trains the model on previously annotated examples and then asks for a small number of further annotations, carefully selecting those examples where the user's annotation is expected to provide as much new information as possible.

2.6. Relations between concepts

In the previous section, we described an approach that constructed relations with no special regard to the concepts: any pair of instances regardless of which concepts they belong to, might have been linked by a relation. It may be desirable to also introduce relations between concepts.

For each relation *r* over instances, we can define a relation $\hat{r}$ over concepts in the following way: the concepts $c, \hat{c} \in C$ are related by $\hat{r}$ iff
$\left|\{e \in \iota_C(c) : \exists \hat{e} \in \iota_C(\hat{c}), e r \hat{e}\}\right| / \left|t_C(c)\right| > \vartheta_1$ and
$\left|\{\hat{e} \in \iota_C(\hat{c}) : \exists e \in \iota_C(c), e r \hat{e}\}\right| / \left|t_C(\hat{c})\right| > \vartheta_2$
for two appropriately chosen thresholds ϑ_1, ϑ_2. In other words, we say that *c* and $\hat{c}$ are related by $\hat{r}$ if a sufficiently large proportion of instances from *c* are related (by *r*) to some instance from $\hat{c}$ and vice versa.

Relations between concepts defined in this way can be used as simply an additional component of the ontology, providing additional information about the problem domain; alternatively, they can also be used to constrain or refine the relations between instances that were introduced in the previous section.

3. Architecture

Our system based on the ideas presented in Section 2 has focus on being able to handle large quantities of data. Having a stream of documents up to time *t*, we take the set of documents from a window of a predefined size. We preprocess these documents, splitting them into words and then representing them by feature vectors using the bag-of-words representation. Additionally, named entities are extracted from each document to serve as instances in our ontology. Instance co-occurrences are defined as all co-occurrences of a pair of instances (named entities) within the same document. We then compute the instance profiles and instance co-occurrence profiles.

We classify the instance profiles and co-occurrence profiles into a static (i.e. not time-dependent) pre-existing hierarchy of topics (in our case the DMoz hierarchy). Each of these topics thus functions as a concept or a relation in our ontology based on the document stream. Thus, at any given point in time, the set of instances, together with the hierarchy of concepts, the hierarchy of relations, the assignment of instances to concepts, and the assignment of the pairs of (frequently co-occurring) instances to relations, forms the state of the stream ontology at that particular time.

For testing and evaluation, we use the Reuters Corpus Volume 1, a collection of approx. 800,000 English-language news articles covering the year from 20 August 1996 through 19 August 1997.

3.1. Feature vectors

Each document *d* is represented by a TF-IDF vector *F*(*d*). Typically a document contains only a small subset of all possible words; that is, most of the components of *F(d)* are zero and only the non-zero ones need to be stored. This sparse representation is very important in allowing one to handle large sets of documents.

In our implementation, not only individual words are used as features; in addition, bigrams and 3-grams, i.e. sequences of two or three adjacent words, are also treated as features. For all these features (including individual words), there is the further requirement that the feature must appear in at least five documents, otherwise it is ignored. As a pre-processing step, all words are stemmed using the Porter stemmer.

3.2. Named entities as instances

As suggested in Section 2, we use named entities as the instances of our stream ontology. The named-entity extraction approach is simple but very efficient at handling large amounts of data.

Whenever one, two or three adjacent words in the document begin with a capital initial, this group of words is considered to be a named entity. There are two exceptions to this rule: firstly, the sequence of two or three adjacent capitalized words may be interrupted by a non-capitalized word if the latter belongs to a predefined list of exceptions (consisting of non-capitalized words that are often a part of named entities, such as "of" and "von"). Secondly, a capitalized word that appears at the beginning of a sentence (and is not part of a group of two or three adjacent capitalized words) is only treated as a named entity if the same word appears capitalized somewhere else in the document collection where it is not the first word of a sentence. This approach has obvious drawbacks such as not being able to handle named entities with names longer than three capitalized words, but the assumption is that we will be dealing with large amounts of data and can rely on statistics to weed out much of the noise.

When comparing two entity names (occurring in the same document) to see if they should be treated as identical, the first word of any multi-word entity name is ignored. This allows e.g. "President Clinton", "Bill Clinton" and "Clinton" to be recognized as referring to the same entity.

This approach to named entity extraction, although simple, has the advantage of being efficient and inexpensive to implement. It does not rely on parsing or POS tagging, nor require a database of known named entities and their attributes. The same approach to named entity extraction has been used and evaluated in [4], where it achieved a precision of 73% and a recall value of 96%.

Thus, for a document *d*, we define *I(d)* simply as the set of named entities occurring in these documents. In principle, the occurrence vector could be defined in other ways, e.g. by building a TF-IDF vector for the paragraph containing the named entity rather than for the whole document, but the documents in the Reuters corpus are relatively short and a named entity is often mentioned in several parts of the document, so the simpler definition $F(f,e) = F(d)$ seems sufficient. Likewise, due to the relative shortness of documents we treat any two named entities as co-occurring as soon as they appear within the same document.

When observing the situation at time *t*, we take into account all documents that occurred at most 10 days before *t*. Notice that this is a sliding-window approach.

3.3. Using the DMoz hierarchy for relations and concepts

We use the DMoz web directory (available at www.dmoz.org) as an aid in identifying relations and concepts. This is a large hierarchy of topics, containing approx. 700,000 topics and over four million links to external web pages. Each topic and each link are described by a title and a short description. The topics are structured into a tree, with a "Top" topic at the root of the tree, 16 topics on the second level (immediately below the root), and so on. We use the topics with at least 100 documents to identify concepts and relations (there are approx. 22,000 such topics in the version of DMoz we used). The rationale for this is that smaller topics are too specific and it would also be difficult to train classifiers for them because of the small amount of training documents. In practice, it may be also advisable to ignore the largest topics because they are too general.

For each of these topics, we train a classifier by computing the centroid of all documents belonging to the subtree rooted by this topic. The classifier then uses the nearest-neighbor method to find the topics whose centroids are closest to a given new document. The emphasis here is on efficiency and scalability rather than maximizing classification accuracy at all costs. (The details of this method have been described in [5]).

Using this battery of classifiers, a co-occurrence profile vector is classified into one or more third- and fourth-level topics from the DMoz hierarchy (together with the size constraint mentioned above, this leads to a set of approx. 6,800 topics). (A constraint on the maximum distance between the vector and the topic centroid is used to determine the number of topics to which the vector should be classified.) Thus each of these topics is treated as a name of relation in our stream ontology. The fact that two instances are in relation *r* thus means that the two named entities frequently occur together in documents about the topic associated to *r*. Similarly, individual entity profiles can likewise be classified into these DMoz topics, which also function as concepts for our stream ontology.

In this way, at any time *t*, we can efficiently and effectively obtain an assignment of the entities into a hierarchy of DMoz-based concepts, and likewise an assignment of co-occurring entity pairs into a hierarchy of DMoz-based relations. Of course other alternative approaches could also be considered, e.g. clustering the profile vectors hierarchically to obtain concept and relation hierarchies. However, using the DMoz concepts has its advantages. Each DMoz concept already has a concise human-readable name (and an accompanying description). Using this predefined hierarchy also means that it is easier to ensure a reasonable amount of continuity in our stream ontology: the set of concepts and relations is always the same, but the assignment of entities to these concepts and relations may change with time.

From the point of view of the formalization described in Section 2.5, one can say that using DMoz gives us a relatively inexpensive way of obtaining a function f_r that predicts whether a pair of entities is related by the relation r or not. Other approaches could evolve manually preparing labeled training data or using active learning; this could improve the performance or support the detection of different types of relations r (not covered by DMoz), but the cost of obtaining the models f_r would be much higher.

3.4. Illustrative example

For illustration, we take a look at the most frequently occurring instances (named entities: New York, Japan, London, UK, France, German, Australia, Europe, United States) and the relations between them based on documents from four different ten-day periods. By comparing outputs of the system visualized as graphs, we could see how the relations between the entities change through time. For instance, the relation between France and UK is in the first time period described by keywords such as Society, Government, Regional, etc., while later in changes to mostly business topics Investing, Business, Stocks, Bonds, etc. The graph itself is not included here for the sake of space.

4. Conclusion and Future Development

We proposed an approach to construct and update an ontology from a stream of documents. Our assumption is that a suitable concept hierarchy for the domain of interest is available (in our case, the DMoz hierarchy) and that the instances we are interested in are actually named entities occurring in these documents. The approach proposed has a great emphasis on efficiency, scalability, and being able to process large quantities of data (hundreds of thousands of documents). The continuity of the ontology through time is ensured by the fact that the same underlying topic hierarchy is used all the time as the basis for the assignment of instances to concepts and of instance pairs to relations.

In future work, we will explore the possibility of constructing a concept hierarchy and a relation hierarchy by clustering the appropriate instance profiles and instance co-occurrence profiles.

5. Acknowledgements

This work was supported by the Slovenian Research Agency and the IST Programme of the European Community under SEKT Semantically Enabled Knowledge Technologies (IST-1-506826-IP) and PASCAL Network of Excellence (IST-2002-506778).This publication only reflects the authors' views.

6. References

[1] Brank, J., Grobelnik, M., Mladenic, D. A Survey of Ontology Evaluation Techniques, Proceedings of IS-2005, 105-108, 2005.

[2] Ehrig, M., Haase, P., Hefke, M., Stojanovic, N. Similarity for ontologies: a comprehensive framework. Proceedings of the 13th European Conference on Information Systems, 2005.

[3] Gaber, M.M, Zaslavsky, A., Krishnaswamy, S. Mining data streams: a review. SIGMOD Rec., 34(2):18–26, 2005.

[4] Grobelnik, M., Mladenic, D. Visualization of news articles. Informatica, 28(4):375–380, 2004.

[5] Grobelnik, M., Mladenic, D. Simple classification into large topic ontology of Web documents, Journal of Computing and Information Technology – CIT 13, 2005, 4, 279 - 285.

[6] Muthukrishnan, S. Data streams: algorithms and applications. Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms, 413–413, Society for Industrial and Applied Mathematics, 2003.

[7] Widmer, G., Kubat, M. Learning in the presence of concept drift and hidden contexts. Journal of Machine Learning, 23(1):69–101, 1996.

[8] Fortuna, B., Mladenic, D., Grobelnik, M. Semi-automatic construction of topic ontology. Proc. of the 2nd workshop on Knowledge Discovery and Ontologies at ECML/PKDD-2005.