ALGORITHM FOR AUTOMATIC CLUSTERING OF LEARNING MATERIALS BASED ON THE USAGE STATISTICS

IVAN ČUKIĆ
Faculty of Mathematics, University of Belgrade;
Mathematical Institute, Serbian Academy of Sciences and Arts;
basysKom GMBH, Darmstadt, ivan@math.rs

Abstract: *Semantic web technologies brought to the desktop by the Nepomuk[1] project can prove to be useful in design of the next-generation e-learning systems due to a high level of data interchangeability between different components of the system and programs. In this article, we will present one of the potential applications of the so called "semantic desktop" when applied to usage behaviour and document classification. This research is implemented as a part of the KDE Contour system and has been successfully tested in real-world environment.*

Keywords: *Semantic desktop, literature classification, usage statistics*

[1] "NEPOMUK : Networked environment for personal ontology-based management of unified knowledge" - EU sponsored project for developing a comprehensive solution for extending the personal desktop into a collaboration environment. The project's website is located at: http://nepomuk.semanticdesktop.org/

1. INTRODUCTION

Current LMS (Learning Management System) developments focus on the best way to allow the administrator (teacher) to classify the literature for the user (students) to be able to have a complete control over the learning process.

This can be considered a good solution for the studies that don't involve doing any research on the subject as a part of the learning process, but to learn only what was already prepared as a part of the curriculum. For more advanced levels like the masters degree or a doctorate, this approach falls short. The reason for this is that the LMSs can not track external literature the user obtained without the help of the LMS.

The second issue with most present LMSs is that the classification needs to be done manually – either by an administrator, or by the user. Even if the LMS allows the user to add literature it adds an unnecessary step for a student researcher which breaks the natural flow of learning.

Document grouping is an important part of the system since it makes it easier for the student (or the teacher) to search and navigate the collection of literature. The proposal here is to implement a global operating system-wide user tracking mechanism built on the semantic desktop technologies, to be able to deduce which documents are important to the user and to be able to automatically group those that relate to each other. Although the algorithms can be applied inside a LMS, their true potential lies in analysis of completely unstructured data.

2. AUTOMATIC CLASSIFICATION

There are two main approaches to document classification. The first is based on the document content. That is, all the data like words, images, references need to be indexed and then some classification algorithm like TF-IDF or Okapi BM25F[1] (used in Lucene) can be applied.

In our case, the user doesn't expect the documents to be necessarily grouped based on the content, but rather on the relations she sees between them. Those relations can largely be deduced based on the usage statistics.

3. RECORDED EVENTS

In order to be able to explain the algorithms involved, some understanding of the required OS-wide infrastructure is needed. There needs to be a service that records the usage events sent by the applications or the desktop environment. Namely, the following types of events:

1. Accessed – document was accessed for an unknown interval of time
2. Opened–Closed – document was open for a specific interval of time
3. Focussed in–Focussed out – document had the user's focus (only one document can have the focus at a given time)
4. Modified – changes to the document were saved at some point in time

The usage events are explained in-depth in Appendix 1.

4. GROUPING BASED ON USAGE STATISTICS

Brute-force. One way of deciding which documents are related would be to sort them by the number of times they were open at the same time.

This approach has two significant disadvantages. The first is the complexity of searching for all usage events that have the similar timestamps and summing them to calculate the score – $O(n^2)$ where n is the number of recorded events, which is usually quite huge.

And the second is that this approach can spot only the direct connections between documents.

Simple linking. The alternative approach, as proposed by Chirita and Nejdl[2], would be to create links between documents that are accessed in sequence. So, for example, if the user opens a web page *A*, and then a file *B*, the system should create a *A*→*B* ink between them, with a weight coefficient representing how many times the *A*→*B* event was registered.

While this provides a very efficient ranking method, it doesn't use any contextual information and doesn't work in cases where the user opens one main file *A*, then opens *B* and *C* that are related to it. Instead of linking *A*→*C* it will link *B*→*C* which is a mistake in a learning environment since most subjects have one essential document, and then the less important ones are related to it, not that much related to each other.

5. GRAVITY-BASED GROUPING

The algorithm proposed here is based on the "brute-force" idea, but with a significantly smaller search space. The general idea is to find the most important documents, and then calculate how other documents relate to those. The justification of this approach is exactly what was the problem with using the "simple linking" algorithm in a real-world e-learning environment – the user has a couple important documents for each exam (main literature, paper or a presentation he is writing etc.) and a lot of less important ones that are tied to the aforementioned (referenced papers, specific algorithm implementations, related images etc.).

Document importance quotient. First of all, we need to be able to tell which documents are important to the user. For that, we need a scoring formula that fulfils the following criteria:

1. If a document *A* has been opened more times than *B*, it should have a higher score;
2. The longer the document was kept open, the higher score it should receive;
3. The longer the document had the focus, the higher score it should receive;
4. The importance of a document should deteriorate with time if it has not been used.

Lets define the document importance quotient *S* as:

$$S = \sum_{i=1}^{n} e^{-d_i} e^{k_i \log(l_i)} \tag{1}$$

Where d_i is time passed since the i^{th} event happened, k_i is coefficient depending on the type of the event, and l_i is the time length of the event.

In the case of the "Accessed" and "Modified" events, which don't have the time length like "Opened – Closed" and "Focussed in – Focussed out" do, l_i can be replaced by a predefined constant. The actual values of the constants can be tweaked to achieve the desired importance for the different events, and to set the speed of time-deterioration.

It is trivial to check that this formula has all the required attributes. What's more, the system doesn't need to calculate the whole sum on every new event, it just needs to process the events that happened since the last time the score was updated.

$$S_{new} = S_{old} \cdot e^{-d} + \sum_{i=n_1+1}^{n_2} e^{-d_i} e^{k_i} \tag{2}$$

where *d* is the time passed since the last score update.

It is worth noting that the calculated score is the score at the time of calculation. After that, the score starts deteriorating, so to obtain the current score at any given time, the following formula can be used:

$$S_{current} = S \cdot e^{-d} \tag{3}$$

where *d* is the time passed since the last score update.

TL algorithm. Let *D* be a set of all documents. A simplified version of TL algorithm goes as follows:

1. Sort the documents in *D* by score, descending
2. Calculate L_k – a set of *k* documents that have a significantly higher score than the others. Note that *k* should be kept significantly smaller than |*D*|
3. For each d_i in in L_k *(i = 1, ..., k)*, create a new group G_i that contains only d_i
4. For each document group G_i, find all the documents in *D* that have been used at the same time as any of the documents in G_i, and add them into G_i
5. Repeat the previous step while $D \neq \cup_{i=1}^{k} G_i$
6. Check the similarities between the resulting groups, and merge groups that contain mostly the same documents.

Remarks. The algorithm above doesn't need to finish, there is a possibility that the condition in the step 5 will never be met. In practice, it is sufficient to limit the repetition of the step 4 a predetermined number of times (for example, three times), effectively removing the potential endless loop. Alternatively, the loop can be ended also when no new documents have been added to any of the groups in the last iteration.

Step 4 can be optimised even further by only finding all documents in D that have been used at the same time as the high scored documents added to G_i in the previous step. Mind that when G_i has more items, it is possible to make a time-interval union of all events for those documents, and not process each one individually.

Pros and cons. The first benefit over the "brute-force" algorithm is the speed. Instead of analysing every pair of documents in *D*, it analyses only pairs that contain highest-scored documents. This number can usually be kept lower than eight per subject.

The second benefit is that it detects indirect connections with connection chains that have the length up to how many times the step 4 was repeated. Whats more, if we add the contextual information of in which iteration a document was added to G_i, we can get a nice hierarchical organisation in each of the groups. It also detects groups that have more than one highest-scored documents.

The main flaw of the TL algorithm is that it doesn't keep the connection information between documents once those are inserted in a group. If this is a desired feature, instead of repeating only the step 4, the whole algorithm can be repeated for each group. This way the main groups will be divided into subgroups, subgroups divided further, etc. thus retaining all the hierarchical information.

6. APPENDIX 1 – STANDARD ONTOLOGIES AND QUERIES USED

The whole system is implemented on well established semantic technologies like RDF and Nepomuk. Most of the ontologies used are a standard defined by the Nepomuk project. Everything else is well-defined in a publicly available KEXT ontology. The relevant parts of the ontologies are presented here in TriG[2] format, while the queries are in written in SPARQL[3].

One of the consequences of using semantic web technologies is that a document, as the term is used in this paper, can be any RDF resource – that is, anything that has an URI – be it an image, a web page, a person, a message or even an application.

Document usage events. As already stated, the usage statistics are collected in a dedicated system service. An application can report events to the service in three levels of detail.

Level 0. The lowest level is to report only that a document was "Accessed" by a certain application, without any other details.

For example, the user clicks a PDF file in the file manager which in turn opens that file in a PDF reader. The file manager can send an event that the file was accessed by the PDF reader. This level should be implemented by the parts of the workspace, the desktop environment – normal applications should implement at least Level 1.

Level 1. This level provides more fine-grained events such as "Opened", "Closed" and "Modified".

This way, the system will know for how long a document was kept open and whether it was edited or just read. Applications like the file manager from the above example can't register these events since those don't have the way of telling when the document was closed.

Level 3 contains "Focussed in" and "Focussed out" events which can tell the service that the document in question actually had the keyboard input, and wasn't minimized or located under some other document.

Ontology. The events are modeled in the NUAO[4] as sub-classes of *nuao:DesktopEvent*. The "Accessed" and "Opened – Closed" events are modelled with *nuao:UsageEvent*. In the case of "Accessed", the start time (*nuao:start*) and end time (*nuao:end*) are the same.

The focus events are modeled with *nuao:FocusEvent* that is registered as a child of the bigger *nuao:DesktopEvent*. The same is true for the "Modified" event with a single difference that it has no duration, but only the timestamp of the modification.

Although the events are well defined in the ontologies developed in the Nepomuk project, the structures for remembering the calculated document scores are not. These, along with the information about the classes are defined in the KEXT (KDE extensions) ontology.

The main item in the ontology is the so called Activity which, in the context of an e-Learning environment can be considered to be a class, or some specific part of the class if it covers more disjunctive topics.

```
kext:Activity
        a rdfs:Class ;
        rdfs:subClassOf rdfs:Resource ;
        rdfs:label "activity" .
```

An activity has an id (*kext:activityIdentifier*), and other properties like the name and icon inherited from *rdfs:Resource*. Each event is stored with the information about the activity it belongs to. The property that links events (and other types of resources) is *kext:usedActivity*:

```
kext:usedActivity
        a rdf:Property ;
        rdfs:label "used activity" ;
        rdfs:comment "The activity that was active when
                resource was created. This is mostly
                used for graphs or desktop events."
        rdfs:domain rdfs:Resource ;
        rdfs:range kext:Activity ;
        nrl:maxCardinality 1 .
```

The calculated cache is stored in instances of the *kext:ResourceScoreCache* class:

```
kext:ResourceScoreCache
        a rdfs:Class ;
        rdfs:subClassOf rdfs:Resource ;
        rdfs:label "Resource score cache" ;
        rdfs:comment "For storing the automatically
                calculated score based on the usage
                statistics" .
```

Each cache is unique for the triple (resource, activity, agent). Resource (the document) is stored in the *kext:targettedResource*, the agent (application that opened the document) in *kext:initiatingAgent* and the activity is stored in the previously defined *kext:usedActivity*. The actual score is stored as *kext:cachedScore* and *nao:score* (Nepomuk Annotation Ontology[5] namespace).

```
kext:targettedResource
        a rdf:Property ;
        rdfs:comment "Resource for which the score
                is calculated." ;
        rdfs:domain kext:ResourceScoreCache ;
        rdfs:label "resource" ;
        rdfs:range rdfs:Resource .

kext:initiatingAgent
        a rdf:Property ;
        rdfs:comment "Relates the score to the agent
                initiating the events." ;
```

[2] TriG is a machine and human readable syntax for serializing Named Graphs and RDF datasets developed at Freie Universität Berlin, the specification is available at http://www4.wiwiss.fu-berlin.de/bizer/TriG/

[3] SPARQL is a W3C standard language for querying RDF available at http://www.w3.org/TR/rdf-sparql-query/

[4] NUAO – Nepomuk User Action Ontology is avaliable at http://oscaf.sourceforge.net/nuao.html

[5] NAO – Nepomuk Annotation Ontology – http://oscaf.sourceforge.net/nao.html

```
        rdfs:domain kext:ResourceScoreCache ;
        rdfs:label "involved agent" ;
        rdfs:range nao:Agent .

kext:cachedScore
        a rdf:Property ;
        rdfs:subPropertyOf nao:score ;
        rdfs:comment "The automatically calculated
                score" ;
        rdfs:domain kext:ResourceScoreCache ;
        rdfs:label "calculated score" ;
        rdfs:range xsd:float .
```

Score calculation. The score is calculated by retrieving all events that haven't yet been processed with the next query:

```
select distinct ?r where {
        ?r a nuao:DesktopEvent .
        ?r kext:usedActivity <activity> .
        ?r nuao:targettedResource <document> .
        ?r nuao:initiatingAgent <application> .
        ?r nuao:end ?end .
        FILTER(?end >= <lastModifiedTime>) .
}
```

And for each of the resulting events, the score is increased like this (C++ code):

```
score += ::exp(- days / 32.0) * intervalLength / 60.0;
```

TL algorithm. As previously stated, the first step of the algorithm is to get the highest scored documents with the following SPARQL query, and from the resulting list to only select only those that have a significantly higher score than others.

```
select distinct ?resource,
   (
      (
         SUM (
            ?lastScore * bif:exp(
               - bif:datediff('day', ?lastUpdate,
               currentDateTime)
            )
         )
      )
   ) as ?score
) where {
   ?cache kext:targettedResource ?resource .
   ?cache a kext:ResourceScoreCache .
   ?cache nao:lastModified ?lastUpdate .
   ?cache kext:cachedScore ?lastScore .
   ?cache kext:usedActivity <activity> .
}
GROUP BY (?resource)
ORDER BY DESC (?score)
```

For each of the resulting documents, an instance of a *kext:UsageGroup* is created. Each group is divided into layers (*kext:UsageGroupLayer*) – one layer per algorithm iteration. The iteration number in which the layer is created is stored as *kext:layerOrder*.

```
kext:UsageGroup
        a rdfs:Class ;
        rdfs:subClassOf rdfs:Resource ;
        rdfs:label "Resource group" ;
        rdfs:comment "A group of resources that are
                used together" .

kext:UsageGroupLayer
        a rdfs:Class ;
        rdfs:subClassOf rdfs:Resource ;
        rdfs:label "Layer of the resource group" ;
        rdfs:comment "A group of resources that are
                used together with the resources
                from the previous layer" .

kext:layerOrder
        a rdf:Property ;
        rdfs:comment "The order of the layer
                in a group" ;
        rdfs:domain kext:UsageGroupLayer ;
        rdfs:label "layer order" ;
        rdfs:range xsd:int .
```

Every *kext:UsageGroupLayer* is linked to the *kext:UsageGroup* it belongs to via the *nie:isPartOf* (Nepomuk Information Element namespace) property, just as is every document is to the corresponding *kext:UsageGroupLayer*.

In order to retrieve only the documents that belong to only one layer, as used in the TL algorithm, the following query is suficient:

```
select ?document where {
        ?document nie:isPartOf ?layer .
        ?layer kext:layerOrder "<level>"^^xsd:int .
        ?layer nie:isPartOf <group> .
}
```

To retrieve all documents that belong to a group, regardless of the layer, the following can be done:

```
select ?document where {
        ?document nie:isPartOf ?layer .
        ?layer a kext:UsageGroupLayer .
        ?layer nie:isPartOf <group> .
}
```

We didn't need to specify the type of *?layer* in the former query since *kext:layerOrder* is defined only on *kext:UsageGroupLayer* instances.

7. CONCLUSION

Semantic web and related technologies have recently had a lot of impact on the e-Learning environments. While most of these developments need an explicit effort from the content creators to insert RDF meta-data into lectures or other kinds of documents, or to create RDF-triples linking the documents together, this can sometimes be done automatically from the gathered usage-statistics.

This paper presented a rather efficient way of doing this by recognizing the important documents and grouping the less important ones around them.

Apart from helping in navigation through vast amounts of data, this approach allows the lecturers to retrace the students' usage patterns to be able to see to which topics they gave too much or too little attention, to see what literature should be added to the default setup for the exam and how those should be grouped in the LMS.

LITERATURE

[1] Hugo Zaragoza, Nick Craswell, Michael Taylor, Suchi Saria, and Stephen Robertson. Microsoft Cambridge at TREC-13: Web and HARD tracks. In Proceedings of TREC-2004.

[2] Paul-Alexandru Chirita and Wolfgang Nejdl. L3S Research Center / University of Hanover: Analyzing User Behavior to Rank Desktop Items.