Computing Conference 2017
18-20 July 2017 | London, UK

Predictive and Evolutive Cross-Referencing for Web Textual Sources

Abstract—**One of the main challenges in the domain of competitive intelligence is to harness important volumes of information from the web, and extract the most valuable pieces of information. As the amount of information available on the web grows rapidly and is very heterogeneous, this process becomes overwhelming for experts. To leverage this challenge, this paper presents a vision for a novel process that performs cross-referencing at web scale. This process uses a focused crawler and a semantic-based classifier to cross-reference textual items without expert intervention, based on Big Data and Semantic Web technologies. The system is described thoroughly, and interests of this work in progress are discussed.**

Keywords—Focused Crawler ; Ontology ; Classification ; Reasonning ; Adaptive ; Cross-Referencing

I. Introduction

Nowadays, discovering knowledge and insights over web data is a major task for most corporations to increase their efficiency. In the context of competitive intelligence, experts look for information in a high number of information sources everyday, and redirect important information to their clients. An important part of their work consists in gathering, analyzing, and summarizing information. This process is done manually by experts on daily basis. As the number of sources evolves rapidly, this process becomes overwhelming. In particular, Cross-Referencing is an important aspect of this process: multiple sources are required to synthetize information, as well as verify its veracity. Looking for a specific piece of information over time is a tedious and time-consuming task, especially considering the important volume of information to be read. Consequently, a loss of valuable information can occur when experts are unable to find multiple information sources that refer to a same piece of information.

In [1], a scalable Hierarchical Multi-label Classification system called Semantic HMC is described. Semantic HMC is an unsupervised ontology-learning process for Big Data. Built on top of Semantic Web and Big Data technologies, the classifier allows to perform Hierarchical Multi-Label Classification of items using Web-reasoning. The process achieves Value extraction from Big Data sources, and focuses on harnessing its Volume and Variety dimensions. Cross-referencing documents on the web is a notable way to tackle the Veracity dimension of such data sources. To perform cross-referencing of web items without overloading the experts, SemXDM extends the approach described in [1]. SemXDM uses web content mining along with web graph mining to efficiently extract valuable information from the web. Both web content and graph information are integrated in an ontology-described knowledge base to acheive item classification and discovery of relevant items.

Unlike previous approaches, the system is adaptive and scalable: ontology evolution is managed at crawling time, using the approach described in [2], while a scalable classification method based on web-reasonning is used. The system is modular, and designed as a set of modules, which revolve around an ontology-described knowledge base. Each module has specifics tasks to fulfill. Five distinct modules compose the process:

- Recommender module: this module is responsible for interacting with the user, processing user queries and returning the most appropriate response, i.e. the most relevant items for the query.
- Crawling module: this module harnesses the web searching for new content using a web crawler[1]. The crawling module takes a classified item as input, and outputs a set of similar items extracted from the web.
- Classification module: this module classifies new data items according to an ontology-described knowledge base at crawling time, using a web-reasonner. The ontology-described knowledge base is constructed automatically from high volumes of data following the approach of [1]. Based on the classifications of each items, a VSM-based scoring method determine their relevance.
- Maintenance Module: this module performs adaptive maintenance of the ontology-described knowledge base, using a scalable approach described in [2]. This approach maintains a coocurrence frenquency matrix, and propagates modification to the classification model based on matrix changes.
- Priority Module: after each crawl performed by the Crawling module, the frontier (set of unexplored links) of the crawl is updated in a web graph. The priority module aims to predict which links have the potential to lead to relevant items using contextual information of the links, based on the context graph approach[3].

To the extent of our knowledge, a scalable and adaptive focused crawling process based on unsupervised ontology-learning and web-reasonning is novel. Next section discusses related work in web crawling, focusing on the use of ontology and ontology learning in this context. The third section describes the process thoroughly. Finally, last section concludes and draws lines for current and future work.

[1]A web crawler is an autonomous agent that explores the web through links in webpages. Web crawlers are the basis of most web search softwares, such as search engines.

II. Related Work

A. Focused Crawlers

The objective of focused crawlers, or topic-oriented crawlers, is to limit the crawling process only to pages relevant to the topic. They maximize the percentage of relevant pages[4], and limit the total amount of data downloaded. They tipycally use a set of seed pages as input to start searching for more relevant data. Seed pages are usually given by the user, or selected among the best answers returned by a Web search engine [5], using the topic as query [6][7][8]. Good seed pages can be either pages relevant to the topic, or pages from which relevant pages can be accessed through hyperlinks either directly or indirectly (ie, through a small number of links). The frontier of the crawl, i.e. the set of candidates pages to download, is then defined by the set of links extracted from the initial set of seed pages. The main assumption behind focused crawlers is that relevant pages usually reference each other, thus forming groups (communites) of relevant pages. Thus, pages that can be accessed from a starting relevant page are more likely to lead to more relevant pages, and should be given download priority.

The Fish-Search approach [9] assigns binary priority values to candidate pages, using keyword matching to determine page relevance. All candidates from relevant pages are assigned the same priority value. Best-First Crawlers [10] assign priority values to candidate pages by computing their text similarity with the topic by applying VSM . The N-Best First crawler [11] is a generalized version of Best-First crawler: at each step, the N pages (instead of just one) with the highest priority are chosen for expansion. Along the same lines, Intelligent Crawling [12] suggests combining page content, URL string information, sibling pages and statistics about relevant or not relevant pages for assigning priorities to candidate pages. This results into a highly effective crawling algorithm that learns to crawl without direct user training. Shark-Search can be seen as a variant of Best-First crawler with a more complicated priority assignment function. The Shark-Search method [13] uses Vector Space Model (VSM) [10] to assign non binary priority values to candidate pages. The priority values are computed by taking into account page content, anchor text, text surrounding the links and the priority value of parent pages (pages pointing to the page containing the links). InfoSpiders [11] use Neural Networks to assign priority values to candidate pages.

Best-First crawlers have been shown to outperform InfoSpiders, Shark-Search and other non-focused Breadth-First crawling approaches [11].

B. Semantic Crawlers

Standard crawlers (eg, Best-First) use classical information retrieval models such as VSM to determine page relevance compared to the topic. These models are based on lexical term matching, thus they do not take into account the semantics associated to the text. Semantic crawlers resolve this issue using term taxonomies or ontologies to describe documents. In term taxonomies (or ontologies), semantically related terms are linked together (e.g. with IS-A r other types of links). Terms conceptually similar to the terms of the topic are retrieved from the taxonomy or the ontology and are used for enhancing the description of documents. Document similarity is then computed by VSM or by specialized models such as the Semantic Similarity Retrieval Model (SSRM) [14], or the model suggested by [15] which have been shown to outperform VSM [16].

[17] uses an existing document taxonomy and seed documents to build a classification model. New documents are then classified into categories, i.e. nodes in the taxonomy. [18] considers an ontology-based algorithm to compute page relevance. Relevant entities, i.e. words occurring in the ontology, are extracted from the page and counted. The relevance of the page towards entities of interest for the user is then computed by using ontology graph measures (e.g. direct match, taxonomic and more complex relationships). The harvest rate is improved compared to a baseline focused crawler.

C. Learning Crawlers

To optimize chances to find relevant pages, Learning crawlers can learn on the topic from a set of example pages (training set). Typically, the user can provide a set of pages and specifies which of them are relevant to the topic of interest. Training may also involve learning the path leading to relevant pages. Early approaches in designing learning crawlers use classifiers such as Nave Bayesian classifier to distinguish relevant pages [17]; others suggest using decision trees [19], First Order Logic [20], Neural Networks and Support Vector Machines [21].

In [22] Support Vector Machines are applied to both page content and link context, and their combination is shown to outperform methods using page content or link context alone. The graph of paths leading to relevant pages is an important factor in focused crawling [23]. [3] introduces the concept of context graphs; first back links to relevant pages are followed to recover pages leading to relevant pages. These pages along with their path information form the context graph. The context graph method builds classifiers for sets of pages at distance $1, 2, ..., n$ from relevant pages in the context graph. The focused crawler uses these classifiers to establish priorities of visited pages, priorities assigned to links extracted from these pages. An extension to this method is the Hidden Markov Model (HMM) crawler [24][25], where a user browses the Web and determines if a page is relevant to the topic or not. The sequence of pages visited is recorded and is used to train the crawler to identify paths leading to relevant pages. [26] proposes a two classifier approach: the open directory (DMOZ5) Web taxonomy is used to classify pages as relevant or not, and to feed a second classifier which evaluates the probability that the given page will lead to a relevant page. Bergmark [27] suggests using document clustering information (e.g., the cluster centroids of training page sets) as topic descriptors. [27] uses tunneling enhancement to best-first focused crawler approach. Sometimes relevant information can be located only by visiting some irrelevant pages first. As the goal is not always to minimize the number of downloaded pages but to collect high-quality data, they propose to continue crawling even if irrelevant pages are found. [21] presents an extensive study of Learning Crawlers and the evaluation of several classifiers used to assign visit priority values to pages. Classifiers based on Support Vector Machines (SVM) seem to outperform Bayes Classifiers and classifiers

TABLE I. COMPARISON OF FOCUSED CRAWLER APPROACHES

	Zheng et al.[30]	Su et al.[31]	Dong et al.[29]	SemXDM
Learning Paradigm	supervised	unsupervised	semi-supervised	unsupervised
Classification	no	yes	yes	yes
Term Learning	no	yes	yes	yes
Relation learning	no	yes	yes	yes
Uncontrolled environment	no	no	yes	yes

based on Neural Networks. Hybrid crawlers [28] combine ideas from learning and classic focused crawlers. In [28], the crawler acts alternatively as a learning crawler guided by genetic algorithms, learning the link sequence leading to target pages, or as a classic crawler.

D. Ontology learning in focused crawling

The main limitations of ontology-based (semantic) crawlers is that the crawler's performance is highly dependant on the ontology used. Two notable issues emerge [29]:

- As ontologies are designed by domain experts, a discrepancy may exist between the domain experts understanding of the domain knowledge and the domain knowledge that exists in the real world.
- Knowledge is dynamic and is constantly evolving, compared with relatively static ontologies.

Because of these issues, ontologies sometimes cannot precisely represent real-world knowledge [29]. To resolve these issues, several approaches were interested in integrating semantic-focused crawling techniques with ontology learning techniques.

[30] proposes a supervised ontology-learning-based focused crawler that aims to maintain the harvest rate of the crawler in the crawling process. The main idea of this crawler is to construct an artificial neural network (ANN) model to determine the relatedness between a Web document and an ontology.

[31] describes an unsupervised ontology-learning-based focused crawler in order to compute the relevance scores between topics and Web documents. Given a specific domain ontology and a topic represented by a concept in this ontology, the relevance score between a Web document and the topic is the weighted sum of the occurrence frequencies of all the concepts of the ontology in the Web document.

[29] proposes a self-adaptive semantic focused (SASF) crawler, by combining the technologies of semantic focused crawling and ontology learning. Semantic focused crawling technology is used to solve the issues of heterogeneity, ubiquity and ambiguity of mining service information, and ontology learning technology is used to maintain the high performance of crawling in the uncontrolled Web environment. This crawler is designed with the purpose of helping search engines to precisely and efficiently search mining service information.

Based on the litterature review, two limitations of focused crawlers have been identified.

- Few approaches combine web content mining and web graph mining to harness the web efficiently.

TABLE II. CLASSIFICATION MODEL CONCEPTS

DL concepts	Description
$Item \sqsubseteq \exists hasTerm.Term$	Items to classify (e.g. document) has terms
$Term \sqsubseteq asString.String$	Terms (e.g. word) extracted from items
$Label \sqsubseteq Term$	Labels are terms used to classify items
$Label \sqsubseteq \forall broader.Label$	Broader relation between labels
$Label \sqsubseteq \forall narrower.Label$	Narrower relation between labels
$broader \equiv narrower^{-}$	Broader and Narrower are inverse relations
$Item \sqcap Term \equiv \bot$	Items and Terms are disjoint
$Item \equiv \exists hasTerm.Term$	Relation that links data items to the terms
$Label \sqsubseteq \forall hasAlpha.Term$	Terms used to create Alpha rules
$Label \sqsubseteq \forall hasBeta.Term$	Terms used to create Beta rules
$Item \sqsubseteq \exists isClassified.Label$	Relation that links items to labels

- Ontology-based approaches have advantages but often fail to adapt to the uncontrolled web environment, where content is dynamic and constantly evolving.

Table I shows the differences of the system with similar ontology-driven focused crawlers.

To tackle these challenges, a novel web mining system called SemXDM (Semantic Cross-Referencing Data Mining) is proposed, extending the approaches of [1][2]. This process uses a focused crawler and a semantic-based classifier to cross-reference data items without expert intervention. Next section describes the SemXDM system and details each of its components.

III. SEMXDM: SYSTEM DESCRIPTION

SemXDM uses Data Mining methods along with Semantic Web technology to perform cross-referencing of data items at web scale. An item is any type of web document which contains text, including (but not restricted to) web pages. As in [1], an ontology-described knowledge base is used to describe, and classify the items. The ontology-described knowledge base consists in a DL ontology with $\mathcal{ALCI}$ expressivity described in table II. This ontology has two purposes: first, it serves as a classification model, composed of a label hierarchy taxonomy, along with classification rules, as exposed in the approach of [1]. Secondly, data items ($Item$ class) are integrated in the ontology at the assertion level, thus the ontology is a repository from which data items can be retrieved.

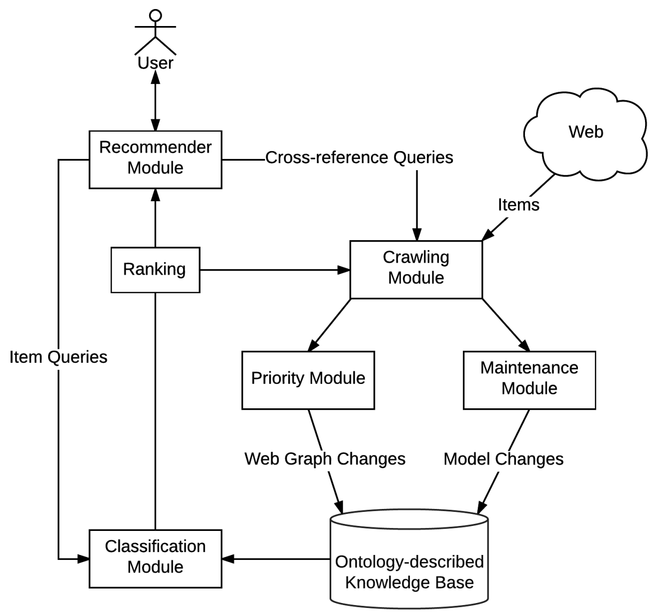
Figure 1 describes the system and the role of each module. The next subsections describe each of the system's modules individually.

A. Recommender Module

The recommender module interacts with the user and respond to its queries. As depicted in Figure 2, the user can issue two types of queries :

- Item Query: this type of query aims to search for items according to an input query in the

Fig. 1. SemXDM components



form of a set of Terms. The search for relevant items is performed offline, ie. all returned items are integrated in the ontology-described knowledge base. It takes a set of Terms $\omega_{term_i} = (term_1, term_2, ..., term_n)$ as input, and outputs the set of items $\omega_{item_j} = (item_1, item_2, ..., item_n)$ where $\forall j$, $\exists$ $item_j$ $hasTerm(term_i)$, and $term_i \in \omega_{term_i}$. All items in the set ω_{item_j} can then be ranked using VSM and similarity measures surch as cosine similarity.

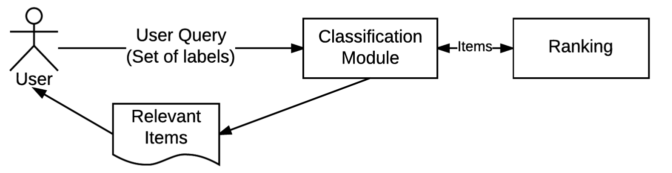
- Cross-reference queries: this type of query takes a classified document in the ontology described knowledge base as input. The query is sent to the crawling module, which aims to search for similar items on the web: relevance of discovered items by the crawling module is computed as the similarity between the initial item (query) and newly discovered items. Let ω_{term}^{init}, the set of terms of the inital item in the query, and ω_{term}^{new}, the set of terms extracted from each new item, the similarity between both items can be computed as the cosine similarity between both vectors. The similarity calculation between crawled items and user queries is described in section III.C.

B. Crawling Module

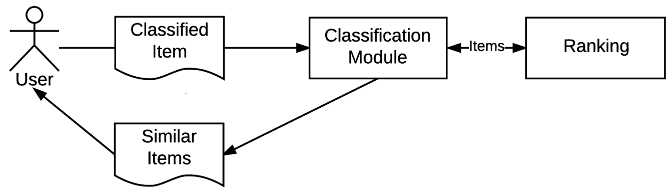
The crawling module uses one or multiple web crawlers to mine the web, searching for relevant items. It is triggered when a cross-referencing query is received from the recommender module. Web crawling is an iterative process: at each step, a fixed set of unvisited URLs, called the frontier, is considered for future fetching[2]. The most promising URLs are fetched, which originates a new frontier to be explored in the next step. Figure 3 describes a generic crawling process.

[2]fetching is the action of downloading a web page to extract its content

Fig. 2. Recommender Module: user interaction



(a) Item Query



(b) Cross-Reference Query

As exploring huge portions of the web is a tedious task for one machine, this process is usually parallelized. This allows respecting access policies while retaining high performance. Parallel crawlers are commonly used in the litterature, where several web crawlers harness the web at the same time. Concurrence issues is generally avoided by maintaining a database of crawled items (crawl database), along with a database of observed hyperlinks (link database).

As described in [2], an item is classified based on relevant terms in its content. A Term-based inverted index is used to store new data items and extract relevant terms for classification. After each crawling step, the Term-based inverted index is updated with all newly acquired items from the web. The index allows for efficient retrieval of items by Term. The inverted index is composed by a set of item vectors,

Fig. 3. Generic Crawling Process

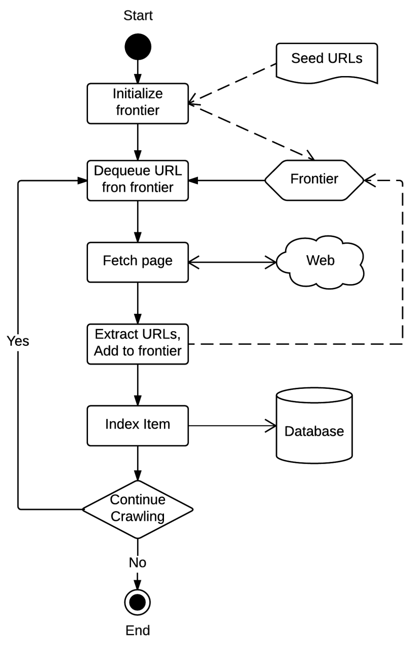
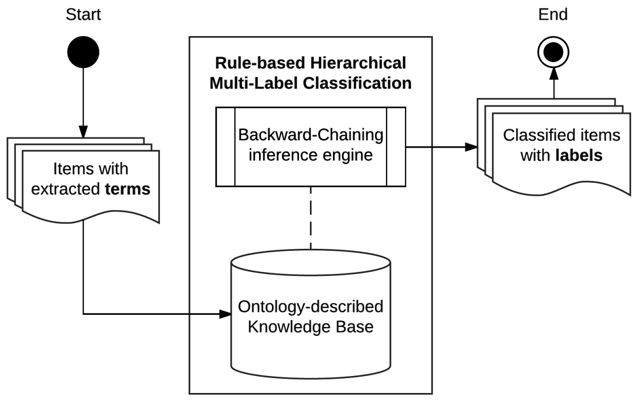


Fig. 4. Classification Module


Start
End
Rule-based Hierarchical Multi-Label Classification
Backward-Chaining inference engine
Items with extracted terms
Classified items with labels
Ontology-described Knowledge Base

one vector for each term $\in$ Term, in the form: $v_{term} < item_1, item_2, ..., item_n >$. Terms are extracted from each item in the form $v_{item_i} < term_1, term_2, ..., term_n >$ using a term extraction approach proposed in [1], which includes spelling correction, stop-word and synonym detection. Labels corresponds to the most relevant terms, as described in [1], with *Label* $\sqsubseteq$ *Term*.

C. Classification Module

The objective of the classification module is to determine the relevance of new items at crawling time, using a Hierarchical Multi-Label Classification method described in [1]. In this approach, relevant terms are first extracted from items as Labels (*Label* class in the ontology). Then, a label hierarchy and classification rules are constructed from the whole item set. Finally, a web-reasonner is used to classify new items according to the label hierarchy and classification rules. This step is referred to as the Realization step. Realization populates the ontology with items and then for each item determines the most specific label and all its subsuming labels.

In SemXM, a predefined classification model (label hierarchy and classification rules) is used. The output of the crawling module is used as the input data to be classified using the Realization method.

The realization step includes two sub-steps: population and classification. The ontology-described knowledge base is populated with new items, and their relevant terms at the assertion level (Abox). Each item is described with a set of relevant terms $\omega_{\gamma}^{(item_i)}$ such that:

$$\omega_{\gamma}^{(item_i)} = \{term_j | \forall term_j \in Term \wedge \gamma < tfidf_{(item_i, term_j, C)}\} \tag{1}$$

where γ is the relevance threshold, $\gamma < tfidf_{(item_i, term_j, C)}$, $term_j \in Term$, $item_i \in Item$ and $tfidf$ is the term-frequency inverse document frequency value for this term in the set of crawled documents.

The classification sub-step performs the multi-label hierarchical classification of the items. Out-of-the-box tableaux-based or resolution-based reasoner's such as Pellet [32], FaCT++ [33], or Hermit [34] are sound and complete to high expressive ontology, such as OWL2 SROIQ(D), but on the other hand they are not highly scalable and cannot handle huge volumes of data. Instead, we propose to use rule-based reasoning that is less expressive but scales better. Rule-based reasoning applies exhaustively a set of rules to a set of triples (i.e. the data items) to infer conclusions [35], i.e. the item's classifications.

The rule-based inference engine uses Horn clause rules to infer the subsumption hierarchy (i.e. concept expression subsumption) of the ontology and the most specific concepts for each data item. This leads to a multi-label classification of the items based in a hierarchical structure of the labels (Hierarchical Multi-label Classification). To infer the most specific labels, the rule generation process described in [1] is used. As described in section 1, the initial rules are generated prior to crawling. These rules are translated into rules in the Semantic Web Rule Language (SWRL). The main interest in using SWRL rules is to reduce the reasoning effort, thus improving the scalability and performance of the system. The rules define the necessary and sufficient terms of an item $item_i$ be classified with a label $label_j$. Two types of rules are defined:

Alpha rules are based on single, highly relevant terms sufficient to classify an item with a label, such as:

$$\begin{gathered} Item(?it), Term(?t_1), Label(?t_1), hasTerm(?it, ?t_1) \\ \rightarrow isClassified(?it, ?t_1) \end{gathered} \tag{2}$$

Beta rules are based on a combination of several terms to classify an item with a label, such as:

$$\begin{gathered} Item(?it), Term(?t_1), Term(?t_2), Label(?t_3), \\ hasTerm(?it, ?t_1), hasTerm(?it, ?t_2) \\ \rightarrow isClassified(?it, ?t_3) \end{gathered} \tag{3}$$

In addition, the following SWRL rule is used to classify an item with any subsuming label:

$$\begin{gathered} Item(?item), Label(?label_1), Label(?label_2), \\ broader(?label_1, ?label_2), \\ isClassified(?item, ?label_1) \\ \rightarrow isClassified(?item, ?label_2) \end{gathered} \tag{4}$$

To apply the rules in the reasoning process, either forward-chaining (i.e. materialization) or backward chaining can be used. Based on these two types of rule-based reasoning, two types of classification are possivle: classification before query time and classification at query time. Classification at query-time is more suited in the context of this work, because items must be re-classified as the ontology-described knowledge base evolves (see next section). Thus, a backward-chaining inference engine is used to obtain determine the classifications of each item. The combination of the crawler and classification modules is a focused crawler based on an ontology-described knowledge base, learned automatically from high volumes of data. Figure 5 shows the integration of the classification process in the crawling module.

After an item is successfully classified with terms and labels, it is possible to determine its relevance compared to an initial set of terms. This initial set of labels can be a user query, or the terms of another item classified by the system.

Fig. 5. Crawler and Classifier Integration

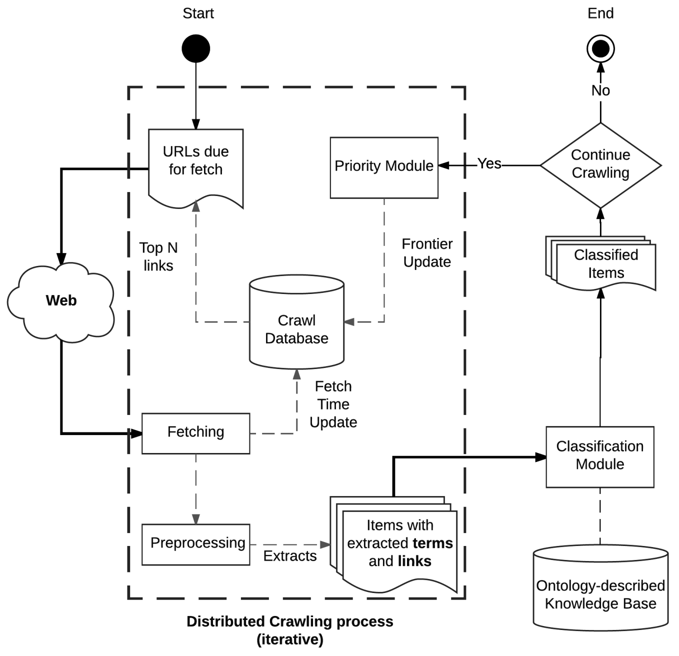


Fig. 6. Maintenance Module

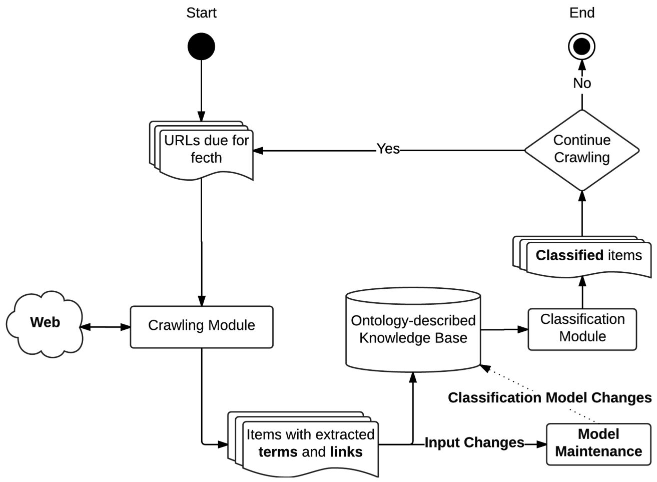


TABLE III. INPUT CHANGE TYPES

Input Change Type	Description
newItem	New item is discovered and added to the inverted index
newTerm	New term is detected in the Item
coOcurrence	Document frequency of a term has changed
appliedModification	Modification Request applied with success

To determine the relevance of new data items in the system, the cosinus distance between a predefined set of terms(ie. the initial query) and the item can be calculated as the vectorial product between them. The result is the **relevance score** $Relevance_i$ of the $item_i$, with $Relevance_i \in [0, 1]$.

As described in previous section, focused crawlers lack ways to adapt to the uncontrolled web environment[29]. According to [29], the classifier used in a focused crawler must adapt to a constantly changing environment, where features are evolving. Secondly, focused crawlers should learn from experience how to find relevant information, using the graph of the web (hyperlinks) [3]. To leverage theses issues, a Maintenance module and a Priority module are described in the next sections. The objective of the Maintenance module is to adapt the ontology-described knowledge base over time based on new data. The Priority module learns ways to find relevant information, by enhancing the priority assignment of items.

D. Maintenance Module

In an uncontrolled web environment, content is dynamic, and constantly evolving: the data is nonstationary (also referred as evolving or drifting) where the probabilistic properties of the data change over time. Theses changes in the data distribution can induce more or less radical changes in the target concept, which is generally known in literature as concept drift [2]. In nonstationary data distribution, the performance of a non-adaptive classification model trained under the false stationary assumption may degrade the classification performance over time.

[2] proposes a scalable Adaptive Learning process to consistently adapt an Ontology-described classification model used for hierarchical multi-label classification regarding a non-stationary stream of unstructured text data in Big Data context. This approach shows enhancement of the classifier's performance when the classification model is updated with a stream of data. The crawling module outputs new data items, which are used as the input of the maintenance process. The Maintenance Module integration is depicted in figure 6.

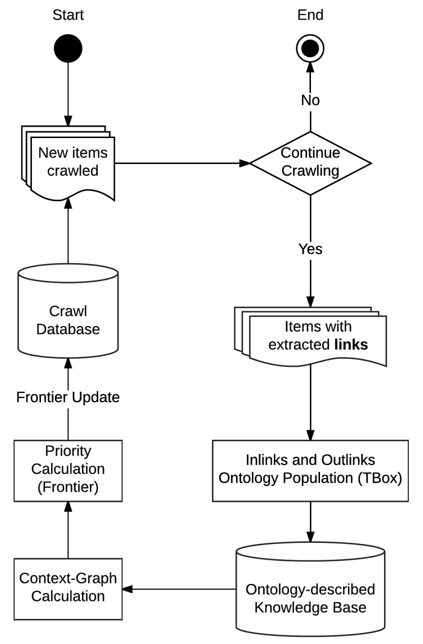
The adaptive learning process focuses on adapting the ontology-described classification model regarding a stream of unstructured text documents in Big Data context. Three characteristics of data stream processing are addressed in the adaptive learning process:

- Feature-evolution that occurs when new terms/words (features) emerge from the data stream and old features fade away;
- Concept-evolution occurs when new classes/labels used to classify items emerge in the underlying stream of text items;
- Concept-drift where the probabilistic properties of the data change over time and induce more or less radical changes in the target concept/label (i.e. changes in the Alpha and Beta sets and in the hierarchical relations).

Using the approach described in [2], the adaptive learning process manages the impact of changes in the classification model according to the specific semantic of the classification process. Input changes are detected from input data (e.g. crawled data) by change sensors. Table III describes the different types of input changes.

Detected and validated Input Changes generate Modification Requests at crawling time to adapt the classification model. Table IV describes the different types of modification requests. Modification requests are finally propagated to the classification model as Model Changes, i.e. Ontological

Fig. 7. Priority Module


Start
End
No
New items crawled
Continue Crawling
Yes
Crawl Database
Items with extracted **links**
Frontier Update
Priority Calculation (Frontier)
Inlinks and Outlinks Ontology Population (TBox)
Context-Graph Calculation
Ontology-described Knowledge Base

changes performed to the ontology-described classification model.

As the process uses classification at query-time to classify items, the evolution of the ontology-described classification model impacts automatically the classifications of the items.

E. Priority Module

The objective of the Priority Module is to combine web graph information and content information retrieved on the web to enhance and guide the search of the Crawling Module, by defining the priority of the link frontier. While the classification module computes scores for classified items, the priority module computes expected scores of items in the frontier, i.e. items that have yet to be downloaded. The Priority Module update web graph information after each mining operation (crawl) in the link database, as described in figure 7. The webgraph is composed of the complete set of discovered items, along with web graph information, i.e. outlinks and inlinks between items.

For each data item $item_i$, the Priority Module populates the ontology-described knowledge base with new inlinks and outlinks at the assertion level. Table V describes the DL

TABLE IV. INPUT CHANGE TYPES

Modification Request	Type Description
AddTerm	Add a new term to the classification model
AddLabel	Add a new Label to the classification model
DeleteLabel	Delete a Label from the classification model
AddHRelation	Add a Hierarchical Relation between two Labels
DeleteHRelation	Delete a Hierarchical Relation between two Labels
AddAlphaTerm	Add a Alpha Relation between a Term and a Label
DeleteAlphaTerm	Delete a Alpha Relation between a Term and a Label
AddBetaTerm	Add a Beta Relation between a Term and a Label
DeleteBetaTerm	Delete a Beta Relation between a Term and a Label

TABLE V. WEB GRAPH INFORMATION DESCRIPTION

DL concepts	**Description**
$Link \sqsubseteq asString.String$	URL of data items
$Item \equiv \exists hasLink.Link$	Relations that links items to their URL
$Item \equiv \exists hasOutlink.Item$	URL redirecting to another data item
$Item \equiv \exists hasInlink.Item$	URL redirecting from another data item
$hasOutlink \equiv hasInlink^{-}$	hasInlink and hasOutlink are inverse relations
$Item \sqcap Link \equiv \perp$	Items and Links are disjoint
$Term \sqcap Link \equiv \perp$	Terms and Links are disjoint

concepts added to the knowledge base to describe the items and their links.

As depicted in table V, the *hasInlink* and *hasOutlink* properties define web graph information for each item in the ontology-described knowledge base. As new information is gathered by the crawling module at crawling time, outlinks and inlinks are updated. Outlinks are updated based on extracted links in the item's content. Inlinks are updated at the same time as the inverse of outlinks. However, as the update of inlinks does not rely on any external process (i.e. a *link:query*[3] request to a search engine), the set of inlinks is incomplete. The result of the ontology population step is a set of hasInlinks and hasOutlinks relations $\omega_{in}^{item_i} = |\{item_1, item_2, ..., item_n\}|$ and $\omega_{out}^{item_i} = |\{item_1, item_2, ..., item_m\}|$, where n is the total number of discovered inlinks, and m the total number of outlinks extracted from each item's content.

To predict potential paths leading to relevant items, the context graph approach described in [3] is used. After each crawl, the different layers of the context graph are recomputed after each crawl as described below.

First, a relevance threshold θ is defined by the user. The set of highly relevant items $\omega_{rel}^{item_i}$ is considered, where $Relevance_i$ is the relevance score of $item_i$, and $Relevance_i > \theta$. In the first crawling step, this setcorresponds to user-provided items, ie. seed items.

Then, from the set of highly relevant items $\omega_{rel}^{item_i}$, each layer L_j of the context graph is defined by the set of items at distance j from $\omega_{rel}^{item_i}$, i.e. $L_j = \omega^{item_k}$ where $\forall k$, $\exists$ path $p(item_k, item_i)$, where $p(item_k, item_i)$ is the shortest path from $item_j$ to $item_i$ using *hasInlinks* properties, and $|p| = j$, the number of $hasInlinks$ properties in the path. For example, $L_1 = \omega^{item_k}$ where $\forall k$, $\exists$ $item_k$ $hasInlink(item_i)$.

Each Layer of the context graph is updated based on the set of terms within its range: L_j is determined as a set of couples $< term_k, value_k >$ where:

- $\forall j$, $item_j \in L_j$
- $\forall k$, $term_k \in Term$
- $\forall k$, $\exists$ $item_j$ $hasTerm(label_k)$
- $\forall j$, $value_k = |item_j|$, $item_j$ $hasTerm(term_k)$

This originates a weighted vector v_i for each layer L_i, where each dimension of the vector is a *Term*. Each vector v_i is recomputed after each crawl, as new relevant items and links are discovered and added to the web graph. The probability of an item being in layer L_i can then be approximated by

[3]Search engines provide a way to extract all the links leading to a web page using a *link:query* request

calculating the distance between the vectors v_i and the set of labels extracted from the item. A vector-based distance such as cosine distance can be used for that purpose.

The resulting probability can be used to predict the distance from a page to a relevant page, and re-rank the frontier according to the new web graph after each crawl. Data items that are potentially closer are then boosted by increasing their priority in the frontier queue.

IV. Conclusion

In this paper we present our vision to perform cross-referencing of web documents using a novel focused crawler approach called SemXDM. SemXDM tackles several issues in focused crawling, using machine-learning, ontologies and Big Data technologies. The process is unsupervised, thus no expert intervention is required. A scalable classifier based on an ontology-described knowledge base allows classifying web documents at crawling-time using a web-reasonner. Also, ontology evolution at crawling time is achieved using an adaptive maintenance process, and web graph mining is used in conjunction with web content mining to enhance the efficiency of the cross-referencing process. Item classification is performed at query-time, thus the item's classifications change with the ontology, as more information is integrated in the ontology-described knowledge base.

Our current research efforts focus on the implementation of the process. The system and is implemented as a Java application and deployed on a Hadoop Cluster[4], which integrates the different modules with several scalable tools from the Semantic Web and Big Data domain. Apache Nutch[5] is used a scalable and distributed crawler. Derived classes from the Nutch API allows to integrate the other modules. Apache Solr[6] is used to build the inverted index of crawled items, while Apache Tika[7] allows to parse and extract information from items. Finally, Stardog[8] is used as a scalable triplestore to integrate the ontology-described knowledge base.

In future work we aim to evaluate the process using appropriate web crawling metrics (eg. harvest rate, fallout rate, precision, recall). We expect to analyze the impact of both ontology evolution and graph-based enhancement on the crawler's performance.

References

[1] R. Peixoto, T. Hassan, C. Cruz, A. Bertaux, and N. Silva, "Semantic hmc: a predictive model using multi-label classification for big data," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 2. IEEE, 2015, pp. 173–179.

[2] R. Peixoto, C. Cruz, and N. Silva, "Semantic hmc: Ontology-described hierarchy maintenance in big data context," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2015, pp. 492–501.

[3] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, M. Gori *et al.*, "Focused crawling using context graphs." in *VLDB*, 2000, pp. 527–534.

[4] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," *IEEE transactions on knowledge and data engineering*, vol. 18, no. 10, pp. 1411–1428, 2006.

[4] http://hadoop.apache.org/
[5] http://nutch.apache.org/
[6] http://lucene.apache.org/solr/
[7] https://tika.apache.org/
[8] http://stardog.com/

[5] F. McCown and M. L. Nelson, "Agreeing to disagree: search engines and their public interfaces," in *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2007, pp. 309–318.

[6] R. Kraft and R. Stata, "Finding buying guides with a web carnivore," in *Web Congress, 2003. Proceedings. First Latin American*. IEEE, 2003, pp. 84–92.

[7] G. Pant, K. Tsioutsiouliklis, J. Johnson, and C. L. Giles, "Panorama: extending digital libraries with topical crawlers," in *Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2004, pp. 142–150.

[8] S. Bao, R. Li, Y. Yu, and Y. Cao, "Competitor mining with the web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 10, pp. 1297–1310, 2008.

[9] P. De Bra, G.-J. Houben, Y. Kornatzky, and R. Post, "Information retrieval in distributed hypertexts," in *Intelligent Multimedia Information Retrieval Systems and Management-Volume 1*. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 1994, pp. 481–491.

[10] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[11] F. Menczer, G. Pant, and P. Srinivasan, "Topical web crawlers: Evaluating adaptive algorithms," *ACM Transactions on Internet Technology (TOIT)*, vol. 4, no. 4, pp. 378–419, 2004.

[12] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu, "Intelligent crawling on the world wide web with arbitrary predicates," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 96–105.

[13] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur, "The shark-search algorithm. an application: tailored web site mapping," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 317–326, 1998.

[14] G. Varelas, E. Voutsakis, P. Raftopoulou, E. G. Petrakis, and E. E. Milios, "Semantic similarity methods in wordnet and their application to information retrieval on the web," in *Proceedings of the 7th annual ACM international workshop on Web information and data management*. ACM, 2005, pp. 10–16.

[15] C. Corley and R. Mihalcea, "Measuring the semantic similarity of texts," in *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*. Association for Computational Linguistics, 2005, pp. 13–18.

[16] A. Hliaoutakis, G. Varelas, E. Voutsakis, E. G. Petrakis, and E. Milios, "Information retrieval by semantic similarity," *International journal on semantic Web and information systems (IJSWIS)*, vol. 2, no. 3, pp. 55–73, 2006.

[17] S. Chakrabarti, M. Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," *Computer Networks*, vol. 31, no. 11, pp. 1623–1640, 1999.

[18] M. Ehrig and A. Maedche, "Ontology-focused crawling of web documents," in *Proceedings of the 2003 ACM symposium on Applied computing*. ACM, 2003, pp. 1174–1178.

[19] J. Li, K. Furuse, and K. Yamaguchi, "Focused crawling by exploiting anchor text using decision tree," in *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM, 2005, pp. 1190–1191.

[20] Q. Xu and W. Zuo, "First-order focused crawling," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 1159–1160.

[21] G. Pant and P. Srinivasan, "Learning to crawl: Comparing classification schemes," *ACM Transactions on Information Systems (TOIS)*, vol. 23, no. 4, pp. 430–462, 2005.

[22] ——, "Link contexts in classifier-guided topical crawlers," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 107–122, 2006.

[23] D. Bergmark, C. Lagoze, and A. Sbityakov, "Focused crawls, tunneling, and digital libraries," in *International Conference on Theory and Practice of Digital Libraries*. Springer, 2002, pp. 91–106.

[24] H. Liu, J. Janssen, and E. Milios, "Using hmm to learn user browsing patterns for focused web crawling," *Data & Knowledge Engineering*, vol. 59, no. 2, pp. 270–291, 2006.

[25] H. Liu, E. Milios, and J. Janssen, "Probabilistic models for focused web crawling," in *Proceedings of the 6th annual ACM international workshop on Web information and data management*. ACM, 2004, pp. 16–22.

[26] S. Chakrabarti, K. Punera, and M. Subramanyam, "Accelerated focused crawling through online relevance feedback," in *Proceedings of the 11th international conference on World Wide Web*. ACM, 2002, pp. 148–159.

[27] D. Bergmark, "Collection synthesis," in *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2002, pp. 253–262.

[28] Y. Chen, "A novel hybrid focused crawling algorithm to build domain-specific collections," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2007.

[29] H. Dong and F. K. Hussain, "Self-adaptive semantic focused crawler for mining services information discovery," *IEEE Transactions On Industrial Informatics*, vol. 10, no. 2, pp. 1616–1626, 2014.

[30] H.-T. Zheng, B.-Y. Kang, and H.-G. Kim, "An ontology-based approach to learnable focused crawling," *Information Sciences*, vol. 178, no. 23, pp. 4512–4522, 2008.

[31] C. Su, Y. Gao, J. Yang, and B. Luo, "An efficient adaptive focused crawler based on ontology learning," in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*. IEEE, 2005, pp. 6–pp.

[32] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," pp. 51–53, 2007.

[33] D. Tsarkov and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description," in *Proceedings of the Third International Joint Conference (IJCAR)*, U. Furbach and N. Shankar, Eds., vol. 4130. Springer Berlin / Heidelberg, 2006, pp. 292–297.

[34] R. Shearer and I. Horrocks, "Hypertableau Reasoning for Description Logics," *Journal of Artificial Intelligence Research*, vol. 36, no. June 2008, pp. 165–228, 2009.

[35] J. Urbani, F. Van Harmelen, S. Schlobach, and H. Bal, "QueryPIE: Backward reasoning for OWL horst over very large knowledge bases," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, ser. ISWC'11, vol. 7031 LNCS, no. PART 1. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 730–745.