International Journal of Innovative Computing, Information and Control
Volume **9**, Number **4**, April **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 1523–1537

AN EXPERT FRAMEWORK FOR EFFECTIVE DOCUMENT CLASSIFICATION USING SUPPORT VECTOR MACHINES

Muhammad Shahbaz[1], Qanita Ahmed[2] and Aziz Guergachi[3]

[1]Department of Computer Science and Engineering
[2]Al-Khawarzami Institute of Computer Science
University of Engineering and Technology
Lahore 54890, Pakistan
Muhammad.Shahbaz@gmail.com; M.Shahbaz@uet.edu.pk; qanita.ahmad@kics.edu.pk

[3]Ted Rogers School of Management – Information Technology Management
Ryerson University
Toronto, ON, Canada
a2guerga@ryerson.ca

Received February 2012; revised June 2012

Abstract. *The amount of textual information available on most topics has been increasing continuously in the public domain and in virtually all organizations. Whether we look at healthcare and its management systems, the environment, climate change and their challenges, or finance, investment banking and their reporting systems, the unstructured textual documents that are published count in the tens of thousands or more. Yet the ability of a human to understand and process it is limited. "Information Overload" arises when extensive and redundant information is available and it is difficult or sometimes even impossible for humans to understand and manage it. As a significant portion of data is not structured, unique analytical tools are required to assert intelligence through unsupervised learning and categorization methods. Though, the art, science, and tools of handling such data have advanced dramatically, there is still a need to structure learning methods by using the state of the art text classification and categorization techniques with the information retrieval tools such that the categories are also semantically associated with each other. In this paper, we present a framework that employs the superior classification capabilities of Support Vector Machines. We will extract useful, high quality information and classify them under various semantic features for the ease of knowledge users and information retrieval.*

Keywords: Text classification, Decision support system, Information retrieval, Support vector machines

1. **Introduction.** Classically, two main approaches to Text Categorization (TC) are presented in artificial intelligence (AI). The first approach is the *Knowledge Engineering* (KE) methods, where expert comprehension about the categories is either declaratively encoded into the system or as a set of procedural classification rules. Second approach to TC is the utilization of *Machine Learning* (ML) methods, where a set of pre-classified examples are used by the classifiers for the learning through a general inductive process. For the purpose of document management, the KE systems typically do better than the ML systems. *Knowledge Acquisition Bottleneck* (KAB) is the key shortcoming of the KE approach. KAB refers to the huge amount of highly skilled labor and expert knowledge required to create and maintain the knowledge-encoding rules [1]. In this paper, we have presented a framework to adhere to the positive aspects of the both approaches to manage the issues imposed by *Information Overload* [2].

In order to overcome the problems presented by *information overload*, we have employed a text classification framework based on Support Vector Machines (SVMs) [3,4] to harness the power of the omnipresent information in organizations. Such a framework is deemed to free the organization from the fatigue of organizing document-bases manually which can be very expensive when a large number of documents are involved and sometimes simply infeasible in the given time constraints of the application. The document management system in any organization has a dynamic nature, since new documents are constantly being created, destroyed, and reorganized. Using text classification for documents has two main concerns in this regard. First, the number of classes may change over time. Second, the contents associated with class labels are also liable to change. These concerns are especially present in the management of knowledge in areas that are critical for the long-term sustainability of our civilization, such as climate change, the environment, healthcare, finance, economic growth and investment banking.

We chose to employ different implementations of SVMs because the dimensionality of the feature space, in case of document classification, is often very large. The number of dimensions is, typically, defined by the number of unique indexed terms in the corpus. Since SVM techniques have proven their superior potential to manage high dimensional input spaces effectively compared with other classification techniques, the need for time consuming linguistic preprocessing (i.e., reduction of dimensions of the feature space) can be largely eliminated.

In this paper, we present a generic framework to investigate the following hypothesis: multiple implementations of support vector machines can be optimally employed to extract useful, high quality information and to classify them under various categories for the ease of knowledge users. We present an extensible framework to classify the documents of an organization into a set of semantic categories, so that the desired information can be retrieved efficiently based on such categories. Our classification model has been instantiated and extended in order to solve a set of diverse categorization tasks: multiple information source handling, feature and meta-feature extraction, intelligent conflict resolution, the acquisition of salient parameters and corresponding implementation of SVMs and, finally, information retrieval and decision support system mechanisms.

2. **Context and Background.** The amount of textual information available is increasing continuously specially in the last twenty years; however, the ability of a human to understand and process it is still limited. "Information Overload" arises when extensive and redundant information is available and it becomes difficult, or sometimes even impossible, for humans to understand and manage it. However, handling the problem of this 'Information Overload' is necessary for utilizing the information purposefully. As a result, ability to automatically organize and classify documents is of great significance. Simply stated, document classification is the task of assigning the unstructured digital data to one or more categories, based on its contents. Document categorization has been used to enhance information retrieval. Document Classification is closely related to many daily applications, including e-mail filtering, mail routing, spam filtering, news monitoring, selective dissemination of information to information consumers, automated indexing of scientific articles, automated population of hierarchical catalogues of Web resources, identification of document genre, authorship attribution, survey coding and so on [5]. Automated text categorization is attractive because manually organizing text document bases can be too expensive and unfeasible given the time constraints of the application or the number of documents involved.

Thus, a text classification framework has been presented in order to harness the power of the omnipresent information in organizations. Such framework is believed to release

the organization from the fatigue of organizing document bases manually. Manual classification of such documents is very expensive when a large number of documents are involved and sometimes simply infeasible in the given time constraints of the application. The document management system in any organization has a dynamic nature where new documents are constantly being created, destroyed, and reorganized.

We choose to employ different implementations of SVMs because in practice, in the case of document classification the dimensionality of the space of the feature space is often very large, as the number of dimensions is defined by the number of unique indexed terms in the corpus. Since SVM techniques have proven their superior potential to manage high dimensional input spaces effectively than other classification techniques, the need for time consuming linguistic preprocessing (i.e., reduction of dimensions of the feature space) can be largely eliminated.

3. **Proposed Solution.** The proposed system, as shown in Figure 1, demonstrates an expert system for document classification. Such a system can contribute significantly to overcome the common problem of high complexity of data and to relate the documents semantically. Therefore, it is helpful for the system to be capable to identify and capture the documents generation events and trigger appropriate action(s) for classifications. This process involves the following sub processes; each of these functions is described in more detail in subsequent sections:

1. Capture events and associated information.
2. Process actions associated with events (e.g., in the sense of an active database system).
3. Identify different parts like images, tables, text, paragraphs from a given document.
4. Extract Meta-feature vector for each part of document against the event trigger, and allow the user to enrich the data manually by adding meaning semantically. Identify and explore interesting patterns to mine valuable information from data sources.
5. Store the information in the knowledge base in an efficient manner such that it includes semantic context of data within.
6. Classify the information as included in the document by the appropriate implementation of SVM as suited by the nature of the enclosed information.
7. Recognize when a given document does not match any category, or when it falls between two categories and to identify new tags or keywords for it. This is because category boundaries are not always semantically well defined.
8. The system may query the data via associated applications and tools with context-sensitive information.

The proposed framework is able to handle any kind of textual material for intelligent classification and knowledge extraction. This framework not only identifies different forms of structured and unstructured text from a text corpus but is also able to build a rule-base for an efficient and effective decision support system. The decision support system is able to improve itself with the extension in the text-base. The proposed approach is unique in a way that it not only captures and preprocesses different kinds of text embedded together but is also able to select the right support vector machine algorithm for knowledge extraction hidden in the text to help build a robust decision support system.

4. **A Framework for Expert Document Classification Based on Multiple Implementations on SVMs.** In this section, we present a detailed formalization of each of the sub-processes mentioned in the last section. The architecture as presented in Figure 1, gives an abstract view of the general working of the system. All the documents from multiple information sources are handed over to the Event Gateway, which functions as

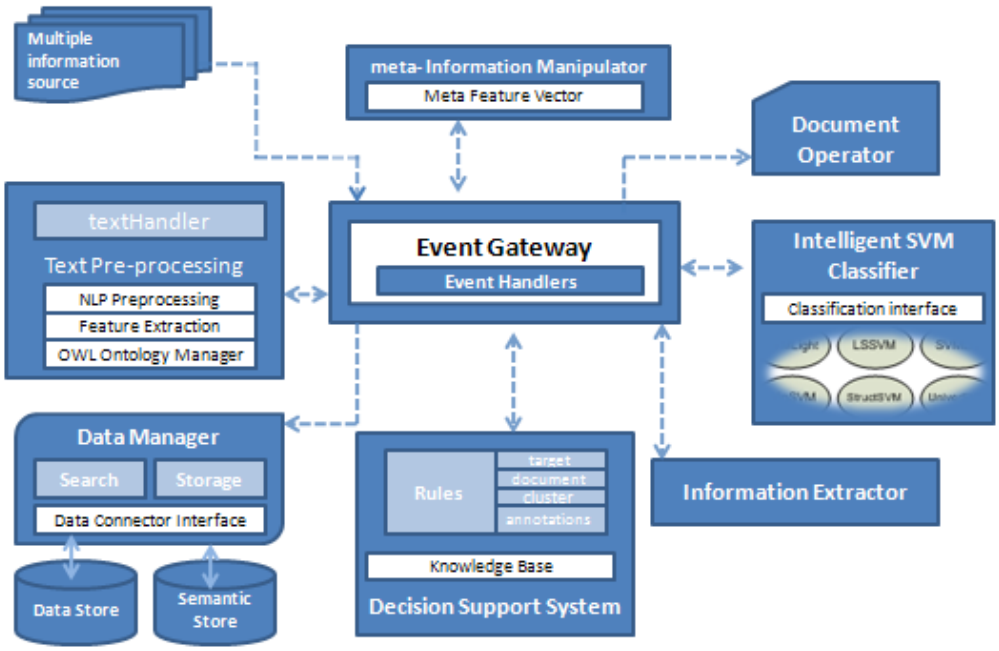

FIGURE 1. An expert document classification framework based on multiple implementations of SVMs

the main command & control centre of the whole model. The Event Gateway manages the fundamental services for the interplay of modules via an event-driven and standards-based messaging-engine. It orchestrates the sequence of invocation of the modules as per required basis. It is responsible for event-triggering and data flow between the modules. It does not take part in any document processing activity. Event Gateway (EG) along with the remaining modules; Document Operator (DO), Meta Information Manipulator (mIM), Text Pre-Processor (TPP), Intelligent SVM Classifier (iSVC), Data Manager (DM), Decision Support System (DSS), Information Extractor (IEr) are presented in this section.

4.1. **Event gateway (EG).** All the functionality of the proposed architecture is initiated and handled by the central event gateway. The events are normally asynchronous external activities that need to be handled by the framework for instance an event occurs, whenever a new document arrives in the system. Therefore, EG is an asynchronous callback subroutine to handle application-level information from the underlying framework. Event Gateway, thus, contains formal specification, in the form of rules (or ontology), for handling these synchronous or asynchronous events.

Whenever a new document arrives in the system, or any other trigger for classification arises, the event handler initiates a sequence of activities to update its current support vectors and finally the knowledge base. The main job of this event gateway is to supervise and overlook all the activities being performed within the scenario. It validates the exact sequence of activities to be carried out, i.e., document analyses, meta-feature extraction, classification mechanisms and launches the systematic instruction manual for the invocation of rest of components of the system to achieve the desired goals of document classification.

Input: $Ev_i \in$ **Ev**, where Ev_i can be any event
Output: sequence of activities
Events:
Event 1.1: Validates the sequence of activities
Activities:
Act 1.1: Maintain business processes

Enterprise Service Bus (ESB) [6], as shown in Figure 2, provides an abstraction layer on top of an implementation of an enterprise messaging system, which allows seamless integration of data and the modules. ESB works in close correspondence with a Business Process Execution Language BPEL [7] Engine to support implementation of complex functionality that requires synchronization of multiple modules (usually using BPEL). In this manner, service orchestration [8] enables coordination of multiple implementation modules.

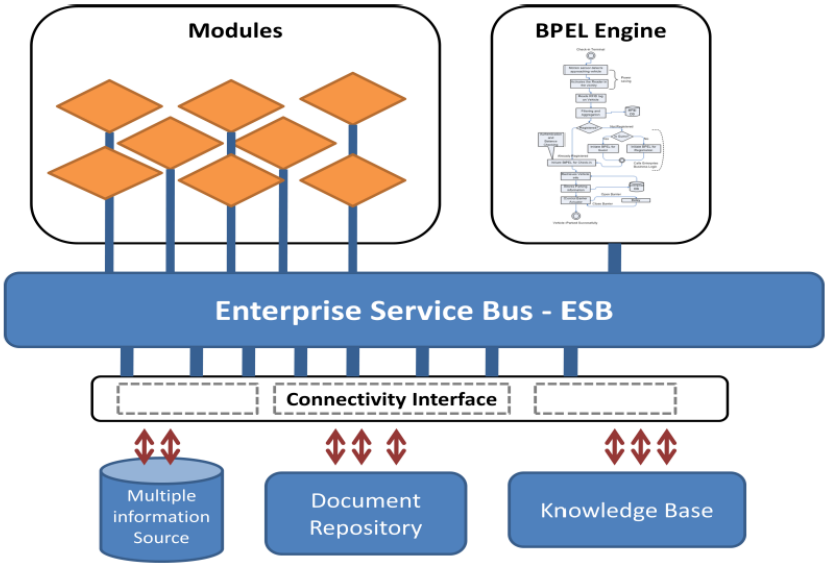

FIGURE 2. A high-level working of event gateway in our system

4.2. **Document operator (DO).** Since we are talking of an open environment, the file types under consideration are not limited or restricted to some specific universally accepted file types. However, at this point, we believe that we should limit the scope of the input document to various file types like html, PDF, xml, image, excel, email, etc.

In a completely automated environment, the input documents must be converted machine understandable text material in order to facilitate the text categorization process. The appropriate elements within the document may be; **text** to store flat information, **tables** for storing statistical information, **context free diagrams** for highest-level view of a system and **images** that may contain the relevant information and other intricate details attached as meta-data captions with the relevant picture.

This module thus analyses the logical structure of a document as described in Figure 3 in order to redistribute the document according to its elements. The process is carried out in three phases: creation of logical structure based on the document architecture model, description of the logical structure based on document grammar and finally generation of an SGML/ XML document based on the elements. The main principle is to identify the individual elements, to analyze the logical structure of these elements as defined in a document architecture model. Whenever a certain element is encountered, its corresponding start and end tags are added directly to the resultant XML document.

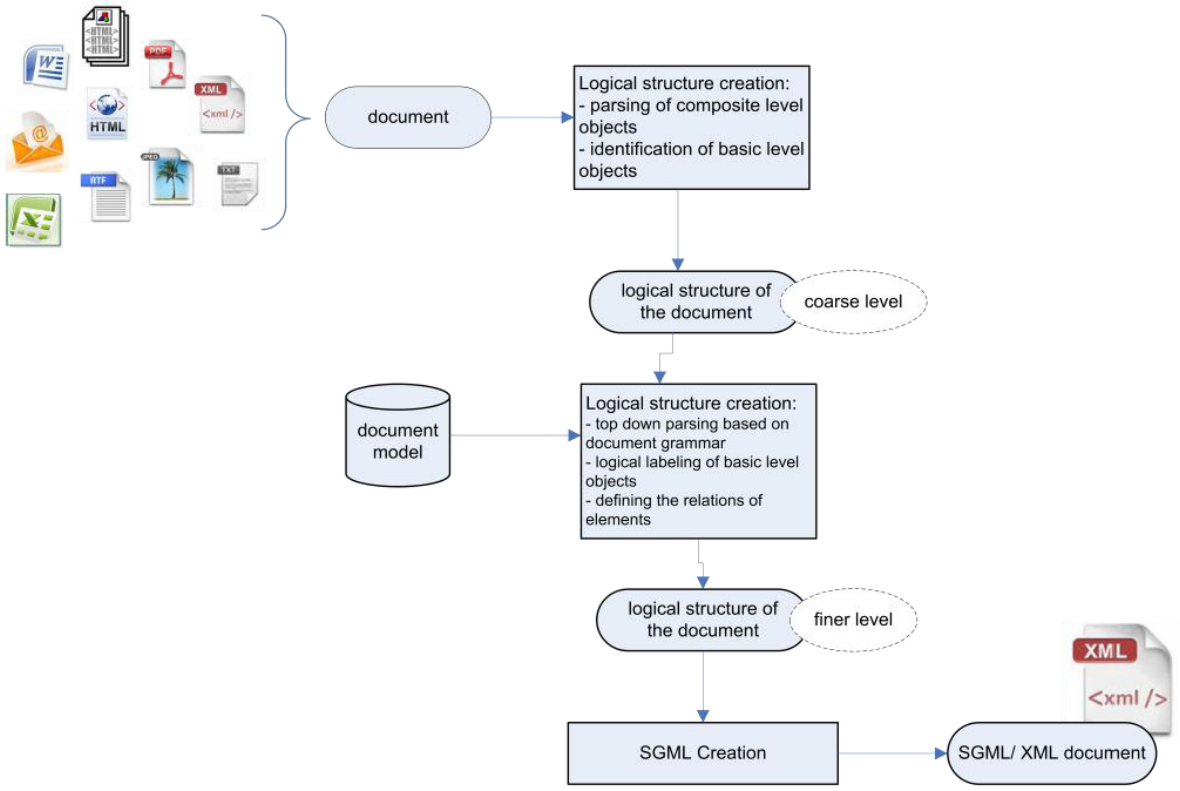

FIGURE 3. A flow diagram for conversion of input document to XML format

```
setDocumentLocator(locator);
  startDocument()
    while(!endofDocumentObjects){                    // Logical Structure Creation
      doStartTag();                          //Top down parsing based on document grammar
        setTagAttributes(tagAttributes);
      startElement(URI, localName, attributes);    //Logical labeling of basic level objects
          setElementAttributes(tagAttributes);
          ∴ Define the relations of the elements
      endElement(URI, localName, attributes);
    }                                                // End of Document Objects
  endDocument()
```

Input: $d_i \in \mathbf{d}$ where d_i is a raw document of any type
$d_i \in$ {html, text, jpg, pdf, email, xsl, ... , xml}

Output: $D_i \in \mathbf{D}$ where D_i is a document of XML type $d_i|_k \overset{filering}{\Longrightarrow} D_i|_{XML}$
where k is any type of document including html, PDF, xml, image, excel, email, etc.

Events:
Event 2.1: Conversion of a document to XML format
Event 2.2: Validation of XML format

Activities:
Act 2.1: Parsing of composite level object for a document
Act 2.2: Identification of basic level object or elements in a document
Act 2.3: Generation of document model
Act 2.4: SGML/XML Creation

4.3. **Meta information manipulator (m-IM).** As the document is pre-processed and each element is individually identified, meta-information about different parts of documents can be extracted to form the schema of that document. The XML schema (W3C XML, 2000) describes the XML document, in terms of restraints on the structure and content of documents of a certain element type. Similarly, during the meta-information feature extraction phase, the elements are mapped on the meta-features contained within the documents. The meta-feature vector, generated at the conclusion of this phase, contains the constituent elements of the document (as recognized in the last module), hence providing a relatively high-level abstract view of the input document. Analogous to XML Schema, which presents a view of the document type at a comparatively higher level of abstraction, structure and field names as well as specific values are described in a self-documenting format. The meta-feature vector MF produced will also contain the elements and token names as contained in the document. The MF is stored in the central data repository managed by Data Manager.

From the implementation perspective, an *XHandler* is provided for each meta-feature in MF as illustrated in Figure 4. *XHandler* is passed back to Event Gateway for the invocation of respective modules as specified in the form of rules. For instance, the *textHandler* is solely responsible for the further processing of textual data. For the scope of current research and development purpose, only *textHandler* is made functional, nonetheless we have foreseen the implementation of the rest of handlers as well.

Input: D_{xml} where D_{xml} is a document of XML type

Output: $MF = \bigcup_{j=0}^{n-1} \varepsilon_j$ where ε_j is the meta-features that belong to Meta-Feature Vector MF

Events:

Event 3.1: Generation of Meta-Feature Vector MF

Event 3.2: Store MF in Data Manager

Activities:

Act 3.1: Generate detailed list of the meta-features contained within the documents

Act 3.2: Incorporation of meta-features in the meta-data of XML document

Act 3.3: Pass MF to the Event Gateway to store it to knowledge repository

4.4. **Text pre-processor (TPP).** As the *textHandler* is initiated through the Meta Information Manipulator (m-IM) module, an event is generated in the Event Gateway to pass the *textHandler* along with the attached element of the document to the Text Pre-Processor (TPP) module. This step is scalable, and the relevant *XHandler* will later be

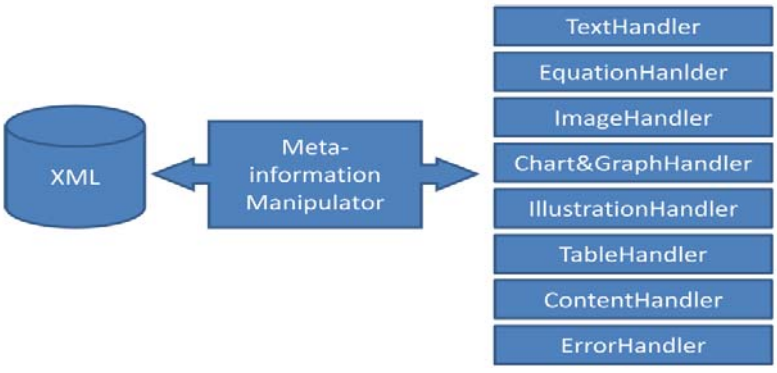

FIGURE 4. Decomposition of an XML document to its constituent meta-features

extended to initiate specific modules for the further processing of the specified elements of the document as described in the previous module.

The standard text pre-processing routine includes: Tokenization, Dimensionality reduction, Stemming, POS Tagging, Feature extraction, TF-IDF, Latent Semantic analysis, Vector space modeling, etc. [1]. As the Text Pre-processor (TPP) is activated, TPP performs the basic steps like tokenization, POS tagging, NP Chunking, stemming for the purpose of feature extraction. For the purpose of this research, we have employed "A General Architecture for Text Engineering" Gate [9], a widely used open source tools for natural language processing, for utilizing its text pre processing functionalities. We briefly describe the core functionality of Gate.

Using Gate for Text Preprocessing: For the purpose of pre-processing, the *TextHandler* ensures the following subtasks as described in Gate [9] are performed:

- To employ Sentence Splitter to generate sentence annotation, in order to identify the sentences from the document under process.
- The key responsibility of tokeniser is to split the text into simple tokens. By default, Gate provides these tokens: Words (upperInitial, allCaps, lowerCase, mixedCaps), Number, Symbol (Currency Symbol, Symbol), Punctuation, SpaceToken.
- For Part-of-Speech (POS) tagging, ANNIE Tagger which is a modified form of Brill Tagger is utilized. The tagger produces a POS tag as an annotation on each word or symbol. The tagger uses a default lexicon and ruleset provided by the Gate Implementation. Some of the papermenters required by the ANNIE Part-of-Speech tagger are: encoding, lexiconURL, rulesURL, document, inputASName, outputASName baseTokenAnnotationType, baseSentenceAnnotationType, and failOnMissingInputAnnotations.
- The English Tokeniser is comprised of a standard tokeniser and a JAPE transducer. The JAPE transducer adjusts the standard output of the tokeniser to the specific requirements of the English part-of-speech tagger.
- Typically the named entities are generated with Person, Organization and Location annotation by the use of Named Entity Transducer and the OrthoMatcher. The gazetteer is to used to recognize entity names in the text based on lists such as names of cities, organisations, days of the week.

The term frequency in TPP module is identified by TF-IDF, and the features are then modeled into the vector space. The Vector Space F is saved in the Data Manager and Event Gateway is notified.

Input: $D_{table}, D_{text} \subseteq D_{XML} \ \because D_{XML} \supset \{D_{table}, D_{text}, D_{image}, \ldots\}$

Output: $F = \bigcup_{j=0}^{n-1} \phi_j$ where ϕ_j is the features that belong to Vector Space F

Events:

Event 4.1: Identification of keywords in the documents

Activities:

Act 4.1: Tokenization
Act 4.2: Stemming
Act 4.3: POS Tagging
Act 4.4: Feature extraction
Act 4.5: TF-IDF
Act 4.6: Latent Semantic analysis
Act 4.7: Vector space modeling
Act 4.8: Dimensionality reduction

Act 4.9: Managing relationships between the categories expressed in Web Ontology Language (OWL).

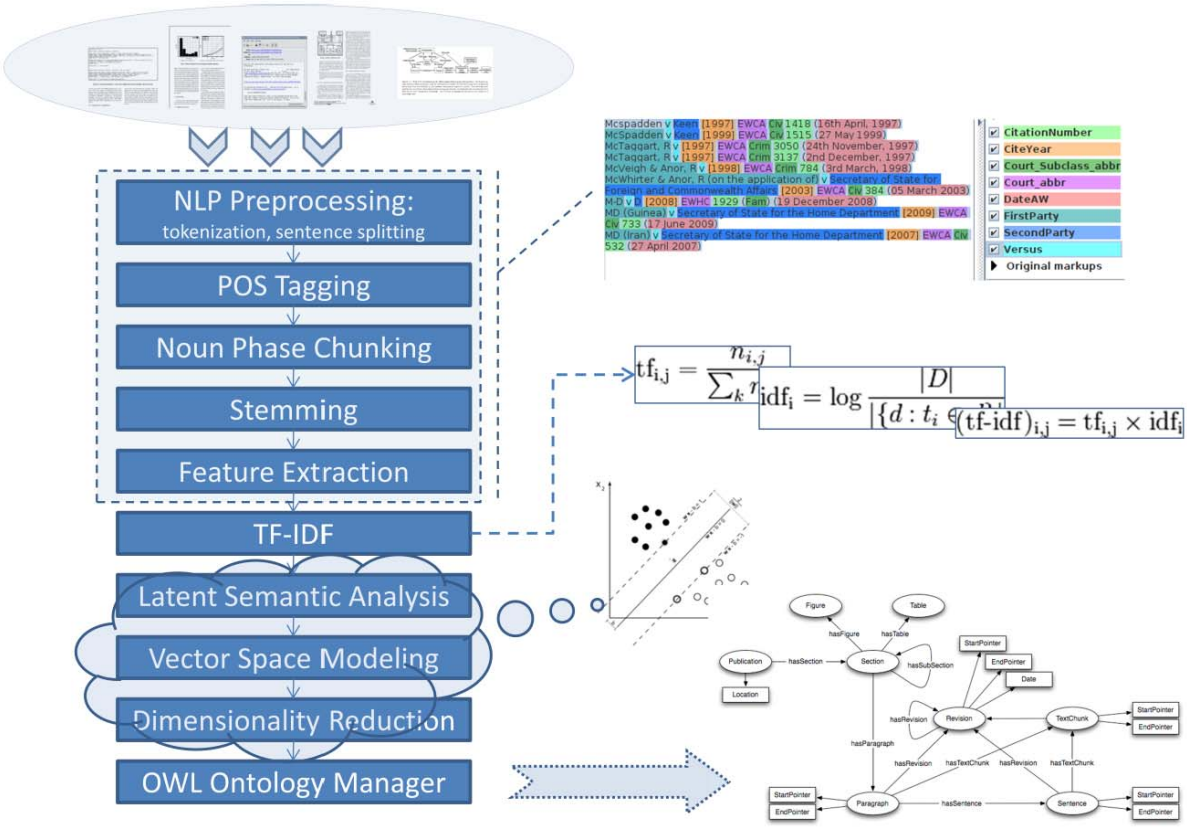

FIGURE 5. An illustration of steps covered in Text-Preprocessing module [9]

4.5. **Intelligent SVM classifier (iSVC).** Once the TPP completes it functionality, the preprocessed documents are fed into the SVM classifier for text categorization. A classifier interface provides point of interaction for the documents and the underlying classifiers. Based on the meta-features of the document, the classifier interface presents the optimal SVM implementations from a pool of available implementations. This enumeration is handed over to decision support module (discussed later), in order to suggest best possible classifier and hand the list of recommended classifiers to the SVM Classifier unit, which then will be held responsible for carrying out the classification process [3].

The ultimate objective of SVM based classifier is to discover a decision surface to divide the training data samples into the target classes. The decisions hence made are based on the support vectors that are chosen as the effective elements in the given training set. SVM makes decision based on the globally optimized separating hyper-plane for text classification [10].

The SVM classifier contains various implementations of SVMs like SVMlight, BSVM, WinSVM, LIBSVM, and LSSVM which are the specific implementations of the support vector machines. The documents are mutually exclusively classified for the classifier as recommended earlier. After classification, some conflicting support vectors may arise. These conflicting support vectors may be simply defined, as the support vector will be passed on to Decision Support System to resolve the conflict.

SVMlight: SVMlight is a C language implementation of Support Vector Machines (SVMs). It employs fast optimization algorithm to solve classification and regression problems.

SVMstruct: It presents SVM learning for multivariate and structured outputs like sets, sequences, and trees.

SVMperf: SVMperf offers new training algorithm for linear classification SVMs that is designed to be faster than SVMlight for large datasets.

TinySVM: TinySVM put forward solution for the problem of pattern recognition based on latest findings in statistical learning theory. It has been implemented to large number of real-world applications, such as hand-written character recognition, text categorization.

SimpleSVM Toolbox: The SimpleSVM Toolbox is a Matlab toolbox (fully implemented in Matlab) that implements the SimpleSVM algorithm. It is fast, easy to modify and makes many extension possible (for instance invariance treatment).

Input: F, **SVM**, $\wp|_{\delta_k} = \bigcup_{j=0}^{n-1} \rho_{jk}$ where ρ_{jk} is the parameter set that defines a certain implementation of SVM δ_k.

Output: $\phi_j'' \subset \phi_j' \subset \bigcup_{j=0}^{n-1} \phi_j$, where ϕ_j' are the Support Vectors, ϕ_j'' are the conflicting support vectors

Events:

Event 5.1: Identifies the parameters of given implementation of SVMs.

Event 5.2: Classify the given test data to target classes

Event 5.3: Conflicting support vectors have arisen

Activities:

Act 5.1: Decide the target class based on the globally optimized separating hyper-plane.

Act 5.2: Identify support vectors.

Act 5.3: Discover a decision surface to divide the training data samples into the target classes.

Act 5.4: Globally optimized separating hyper-plane for text classification.

Act 5.5: Resolve the problem of conflict support vectors.

4.6. **Data manager (DM).** The principle of the data store is to store and get documents. Data Store is a document repository server to store, query and fetch XML based documents. It has been designed for practical needs to allow the storage of semi-structured documents and un-structured documents. The documents to be classified are stored in conventional relational database (MySQL, Postgresql, SAP DB, IBM DB2) [11].

Ontology Manager is a lightweight tool for managing relationships between the categories expressed in Web Ontology Language [12]. With this manager, one can browse, search, and submit ontologies to an ontology repository. Users can discover relationships between several categories without having to go through all the relevant documents. Thus saves the time and effort [13].

The categories lying under conflicting support vectors are modified according to the ontological relationships as specified in the Ontology Manager to resolve the state of divergence. SVM Classifier can reclassify the documents with revised information.

Input: $\mathrm{d_i}|_k$, $\mathrm{D_i}|_{XML}$, MF, F, $\wp$, δ_k, **SVM**

Output: Data Store, Semantic Store, Ontology Manager

Events:

Event 6.1: Data needs to be stored or retrieved.

Activities:

Act 6.1: Fetch XML based documents.

Act 6.2: Query XML based documents.
Act 6.3: Store and retrieve meta-feature vector
Act 6.4: Store and retrieve feature vector
Act 6.5: Browse, search, and submit ontologies to an ontology repository.
Act 6.6: Mark the specific feature as support vector and conflicting support vectors
Act 6.7: Keep the record of all the implementation of SVMs available and their respective parameters

4.7. **Decision support system (DSS).** The meta-feature vector is fed into the decision support systems (DSS) which are simple but intelligent information extraction systems that are including knowledge base systems for all the classifiers to support decision making activities. Based on the meta-features of the document, the DSS suggests the optimal SVM implementation from various implementations available. The main job of decision support module is to suggest the classifiers and hand the list of recommended classifiers to the SVM Classifier unit, which then will be held responsible for carrying out the classification details. The components of this DSS will be: (1) Operation Rule Filter, to describe the operation logic, i.e., the condition and action; (2) Objective Events Filter, to discriminate the events with anticipated properties; (3) Rules Mapper to match the condition logic as described in operation rules and map them to specific implementation of support vector machines.

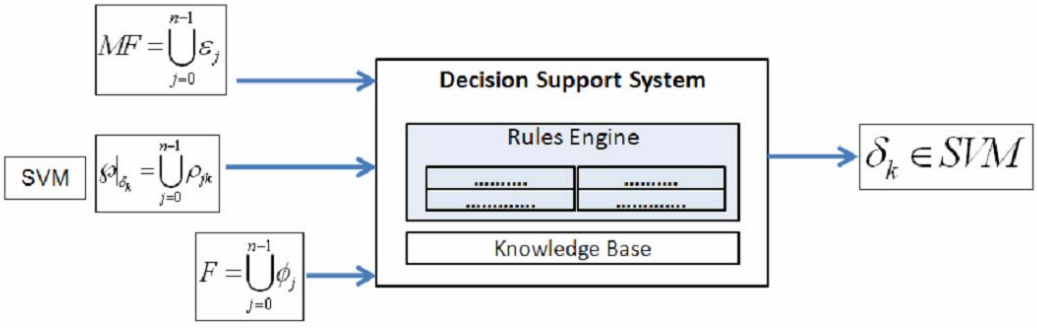

FIGURE 6. Decision support system to suggest the optimal implementation of SVM

For the purpose of our framework, we need these methods for constructing and operating autonomous Information Retrieval (IR) systems:

1. Define effectiveness measure that computes a score, indicating how good those decisions were, when they are applied to a set of decisions made by the system.
2. Tune the system for the best possible effectiveness of its decisions; this can be carried out simply by assigning the weights to the most pertinent SVM implementation. For a more sophisticated and optimized performance, attributes of the training new data can be crosschecked with the available implementations of SVM to predict most suitable classifier.

The system estimates the effectiveness of its decisions in an on-going fashion, and invokes the appropriate module when these estimates indicate some problem with handling new data.

Input: F, MF, $\wp$
Output: $\delta_k \in$ **SVM**
Events:
Event 7.1: Suggest the optimal implementation for classification

Activities:
Act 7.1: Determine effectiveness measure for the decision
Act 7.2: Operation Rule Filter
Act 7.3: Objective Events Filter
Act 7.4: Rules Mapper
Act 7.5: Determine suitable implementation of SVM according to the given parameters and meta-feature vector space

4.8. **Information extractor (IEr).** As the ontology manager in the previous phase stores the relationships between the categories in the form of ontologies. The ontologies are queried for the purpose of finding out the relationships between the different categories. The information extracted in this manner allows the system to adjust the support vectors to resolve the conflicts of classification. The accuracy in the discovery of inherent information of the document is improved if ontologies are described at a finer level [14].

TABLE 1. Comparison of different implementations for SVMs

	Author	Platform	Comments
SVM^{light} [10]	Joachims	Linux, Windows, Cygwin, and Solaris	• SVM classification and regression package • fast optimization algorithm • can be applied to very large datasets
SVM^{struct} [24]	Joachims	Linux	• model complex (multivariate) output such as trees, sequences, or sets • can be applied to natural language parsing, and part-of-speech tagging
SVM^{multiclass}	Joachims	Linux, Windows	• for multi-class classification
SVM^{cfg}	Joachims	Linux, Windows	• learns a weighted context free grammar from examples
SVM^{align}	Joachims	Linux, Windows	• learns to align protein sequences from training alignments
SVM^{hmm}	Joachims	Linux, Cygwin	• Learns a Markov model from example
Latent SVM^{struct}	Joachims	Linux, Windows, Cygwin,	• Training of structural SVM predictions rules for unobserved dependency structure in NP-coref, motif finding, ranking with weak ordering
SVM^{map}	Yue, Finley	Linux	• Learns rankings that optimize Mean Average Precision (MAP) as the performance metric.
SVM^{div}	Yisong Yue	Linux, Cygwin, Windows	• Learns to predict diversified rankings and sets for Information Retrieval.
SVM^{perf} [22]	Joachims	Linux Cygwin	• binary classification rule that directly optimizes ROC-Area, F1-Score, or the Precision/Recall Break-Even Point
SVM^{rank}	Joachims	Linux Cygwin, Windows	• Learns a rule for predicting rankings as typically used in search engines and other retrieval systems
mySVM	Stefan Rüping	C++ source code and Windows binaries	• SVM classification and regression
LIBSVM	Chang and Lin	interfaces for Python, R, Splus, MATLAB, Perl, Ruby, and LabVIEW	• C-classification, ν-classification, ε-regression, and ν-regression • also supports multi-class classification, weighted SVM for unbalanced data, cross-validation and automatic model selection
BSVM [11]	Hsu and Lin	Linux and Windows	• provides two implementations of multi-class classification and SVM regression
TinySVM [23]	Vapnik	Linux /Windows	• C-classification and C-regression, uses sparse vector representation • can handle several ten-thousands of training examples, and hundred-thousands of feature dimensions
R-SVM	Zhang and Wong	Linux	• specially designed for the classification of microarray gene expression data
MATLAB SVM Toolbox		Windows , linux	• implements SVM classification and regression

However, it is important not to identify loads of rules in order to avoid the problem of "over accurate", because in such cases the results may not be useful. If ontologies are considered as information chains, knowledge gained from one domain can be extended to another domain through similarity over components in order to improvise the classification of documents, when a large and heterogeneous collection is covered.

Input: $\phi_j'' \subset \phi_j'$

Output: $\phi_j'' \subset \phi_j' \subset \bigcup_{j=0}^{n-1} \phi_j$

Events:

Event 8.1: The conflicting support vectors need to be resolved

Activities:

Act 8.1: Query the stored ontologies to find out the relationships between the different categories

Act 8.2: Adjust the support vectors to resolve the conflicts of classification

5. **Validation and Verification.** In order to assert that an algorithm is *functionally correct*, we have to show that for all input data that satisfies some condition, which is called the *pre-condition* of the algorithm, the output data satisfy a certain pre-defined condition, which is called the *post-condition* of the algorithm [15].

As illustrated in Figure 7, the respective pre-conditions and post-conditions are shown for each module and the overall data processing in the system. The Document Operator DO receives documents $\mathrm{d_i}|_k$ of any extension from multiple information sources and converts it into a universal format of XML document$\mathrm{D_i}|_{XML}$. The XML document is then presented as input for meta-Information Manipulator m-IM to analyze the elements of the XML document D_{XML}. Each element is labeled with respective meta-features $MF = \bigcup_{j=0}^{n-1} \varepsilon_j$ and respective handlers for the features are instantiated. In order to limit the scope of the implementation in the initial phases, only *textHandler* is tackled so far. The Text Pre-processing module, extracts the features $F = \bigcup_{j=0}^{n-1} \phi_j$ from the input text provided.

For the choice of optimal implementation of SVM $\delta_k \in SVM$, Decision Support System DSS is invoked. DSS takes $MF, F, SVM, \wp|_{\delta_k} = \bigcup_{j=0}^{n-1} \rho_{jk}$. The intelligent SVM classifier contains several implementations of SVMs, once the DSS suggests a certain implementation $\delta_k \in SVM$ along with the feature vector $F = \bigcup_{j=0}^{n-1} \phi_j$, to be classified, are presented for classification. If successfully classified at this point, support vectors $\phi_j' \subset \bigcup_{j=0}^{n-1} \phi_j$ are stored to the database and the classification operation is finished. If, however, some conflicting support vectors $\phi_j'' \subset \phi_j'$ arise at this point, they are fed into Information Extractor (IEr). IEr queries upon the ontologies stored in the TPP phase and then is relocated to the conflicting support vectors $\phi_j'' \subset \phi_j' \subset \bigcup_{j=0}^{n-1} \phi_j$.

As discussed in the previous section, the Event Gateway (EG) manages the fundamental service for the interplay of modules. It orchestrates the sequence of invocation of the modules as per required basis. It does not take part in any document or data processing activity. Similarly, the core functionality of Data Manager (DM) is to store and retrieve

the parameters like $\mathrm{d_i}|_k$, $\mathrm{D_i}|_{XML}$, $MF = \bigcup_{j=0}^{n-1} \varepsilon_j$, $F = \bigcup_{j=0}^{n-1} \phi_j, \delta_k \in SVM$, $\wp|_{\delta_k} = \bigcup_{j=0}^{n-1} \rho_{jk}$, $\phi_j'' \subset \phi_j' \subset \bigcup_{j=0}^{n-1} \phi_j$ and does not involve in the core processing of the system. Hence, Event Gateway (EG) and Data Manager (DM) are not formally specified in Figure 7.

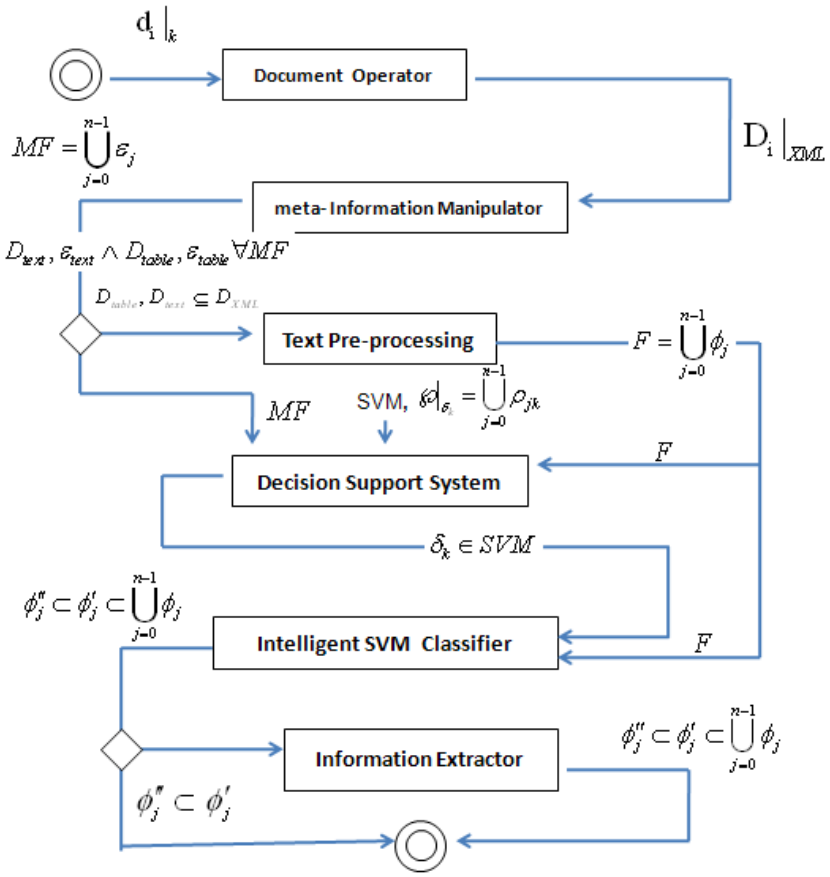

FIGURE 7. Formal verification of the proposed system

6. **Conclusion.** As the usage of computers and electronics has grown in industrial systems, an enormous increase in document generation has resulted. As the behavior of industrial systems becomes more complex, the common activities are more dependent on the availability of knowledge and information. To tackle the information challenges in the industrial environment, the automated text classification for unstructured information has flourished for some time now. In this paper, we have presented a framework for expert document classification for industrial informatics based on multiple implementations on SVMs, to cater to diverse information needs. Industrial informatics focuses on different methodologies of information technology. To enhance industrial fabrication, intelligence and manufacturing processes, we have combined a document classification framework with the semantic features of information retrieval, for the benefit of both diverse methodologies.

Acknowledgment. This work was supported in part by NSERC (Natural Science and Engineering Research of Council of Canada) and CFI (Canadian Foundation for Innovation).

REFERENCES

[1] R. Feldman and J. Sanger, *The Text Mining Handbook, Advanced Approaches in Analyzing Unstructured Data*, Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, 2007.

[2] A. F. Farhoomand and D. H. Drury, Managerial information overload, *Commun. ACM*, vol.45, no.10, pp.127-131, 2002.

[3] T. Joachims, Transductive inference for text classification using support vector machines, *ICML*, pp.200-209, 1999.

[4] T. Joachism, *Learning to Classify Text using Support Vector Machines: Methods, Theory and Algorithms*, Kluwer Academic Publishers, 2002.

[5] G. T. Raju, P. S. Satyanarayana and L. M. Patnaik, Knowledge discovery from web usage data: Extraction and applications of sequential and clustering patterns – A survey, *International Journal of Innovative Computing, Information and Control*, vol.4, no.2, pp.381-389, 2008.

[6] D. Chappell, *Enterprise Service Bus: Theory in Practice*, O'Reilly Media Published, 2004.

[7] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems, *Business Process Execution Language for Web Services Version 1.1*, http://www.ibm.com/developerworks/library/specification/ws-bpel/, 2007.

[8] C. Peltz, *Web Services Orchestration and Choreography. Computer 36, 10 (Oct. 2003)*, IEEE Computer Society Press, Los Alamitos, CA, USA, 2003.

[9] H. Cunningham, D. Maynard and K. Bontcheva, *General Architecture for Text Engineering (GATE): A Full-Lifecycle Open Source Solution for Text Processing*, The University of Sheffield, http://gate.ac.uk/overview.html, 2010.

[10] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor and J. Vandewalle, *Least Squares Support Vector Machines*, World Scientific, Singapore, http://www.esat.kuleuven.be/sista/lssvmlab/, 2002.

[11] S. Csaba and A. Attila, *Data Store Version 0.7a*, http://spacemapper.sourceforge.net/DataStore/index.html Revision:1.13, 2002.

[12] World Wide Web Consortium W3C, *Extensible Markup Language (XML) 1.0*, http://www.w3c.org/TR/REC-xml, 2000.

[13] D. Rekesh and K. Mittal, A Web-based system for managing Web Ontology Language (OWL) ontologies, *IBM Web Ontology Manager*, 2006.

[14] G. Amati, F. Crestani and F. Ubaldini, A learning system for selective dissemination of information, *Proc. of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, 2nd Edition, *Introduction to Algorithms*, The MIT Press, 2001.