SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

A maintainable hierarchical multi-label classification process for Big Data based on web reasoning

RAFAEL PINTO PEIXOTO

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

THÈSE présentée par

RAFAEL PINTO PEIXOTO

pour obtenir le

Grade de Docteur de
l'Université de Bourgogne

Spécialité : **Informatique**

A maintainable hierarchical multi-label classification process for Big Data based on web reasoning

Unité de Recherche :
Laboratoire Electronique, Informatique et Images (LE2I)

Soutenue publiquement le 9 Décembre 2016 devant le Jury composé de :

MARIA-ESTHER VIDAL	Rapporteur	Professeur à l'Université Simon Bolivar
SVEN GROPPE	Rapporteur	Maître de Conférence HDR à l'Université de Lübeck
JOÃO GAMA	Examinateur	Professeur à l'Université de Porto
CHRISTOPHE CRUZ	Directeur de thèse	Maître de Conférence HDR à l'UBFC
NUNO SILVA	Co-encadrant de thèse	Professeur à Polytechnique de Porto

ACKNOWLEDGEMENTS

Overall, I would like to thank everyone that in some way gave me their support and contribution during my entire work at GECAD (Knowledge Engineering and Decision Support Group, ISEP, Polytechnic of Porto) and at the LE2I (Le2i UMR6306, CNRS, Arts et Métiers, Univ. Bourgogne Franche-Comté).

First, I would like to thank both my advisers who patiently guided and actively helped me in my research. To Dr. Christophe Cruz, for giving me the unique opportunity of doing this thesis at the Univ. Bourgogne Franche-Comté, for his support during the elaboration of this work and all constructive discussions. To Professor Nuno Silva, for all his effort and time spent advising and constructively criticizing this work, but especially for his support in the last six years as scientific coordinator. Without his guidance and constant feedback, this PhD would not have been achievable.

Secondly, I would like to thank my thesis committee members: Professor Maria-Esther Vidal, Dr. Sven Groppe and Professor João Gama who have accepted to revise my thesis; for their time, interest, and helpful comments.

Thirdly, I would like to thank the opportunity of working in the AAL4ALL-Ambient Assisted Living (QREN 13852) project and their support in making this work possible.

Fourth, a special thanks to my girlfriend, my mother, my father and my brother, for their unconditional support. Last but not least, I would like to thank all my friends for all the good moments away from work.

TABLE OF CONTENTS

I

CONTEXT SPECIFICATION

1

INTRODUCTION

Retrieving valuable information over web data concerning a particular customer is a complex task addressed by both business intelligence and the data-mining field [Witten et al., 2005]. Data mining is not a trivial process and proper techniques for data analysis and representation are required. In the context of Big Data, data analysis is even more challenging, due to Big Data characteristics. An increasing number of V's has been used to characterize Big Data [Chen et al., 2014, Hitzler et al., 2013]: Volume, Velocity, Variety and Value. Volume concerns the large amount of data that is generated and stored through the years by social media, sensor data, etc.[Chen et al., 2014]. Velocity concerns both the production and the process to meet a demand because Big Data is not only a huge volume of data but it must also be processed quickly as new data is generated over time. Variety relates to the various types of data that constitute the Big Data. These types include semi-structured and unstructured data representing 90% of its content [Syed et al., 2013] such as audio, videos, webpages, and texts, as well as traditional structured data. Value measures how valuable information is to a Big Data consumer. Value is the most important feature of Big Data and is its "raison d'être", because the user expects to make profit out of valuable data [Sheth, 2014].

Big Data analysis can be deemed as the analysis technique for a special kind of data. Therefore, many traditional data analysis methods used in Data Mining (algorithms for classification, clustering, regression, among others) may still be utilized for Big Data Analysis [Chen et al., 2014]. This thesis focuses on a specific type of classification called Hierarchical Multi-Label Classification (HMC). It is a traditional Data Mining method that combines Multi-Label Classification with Hierarchical Classification [Bi et al., 2011]. In the classification field, a data item is anything that can be analysed (e.g. music, a video or a book). In HMC, the items can be assigned to different hierarchical paths and may simultaneously belong to different class labels in the same hierarchical level [Bi et al., 2011]. Werner et al. [Werner et al., 2014] proposed a method to semantically enrich an ontology used to describe the domain in a hierarchical way and to process the classification of text using the hierarchy of terms (HMC). This ontology aims to reduce the gap between the expert's perspective and the representation of classification rules. To enrich the ontology and classify the documents, an out-of-the-box Description Logics Web Reasoner like Pellet [Sirin et al., 2007], FaCT++ [Tsarkov et al., 2006], or Hermit [Shearer et al., 2009] is used. Most of these reasoners are sound and complete to high expressiveness such as the OWL2 SROIQ (D) expressiveness but on the other hand they are not highly scalable [Werner et al., 2014]. They are good enough for a proof of concept but when there's an increase in the number of documents, words and taxonomies, these reasoners cannot handle a large amount of data as is required in Big Data context [Werner et al., 2014]. How-

ever, as Semantic Web is growing, new high-performance "Web Scale Reasoning" methods have been proposed [Urbani, 2013]. These approaches, instead of using traditional Description Logics approaches like Tableau [Sirin et al., 2007, Tsarkov et al., 2006], Resolution [Motik et al., 2006] or Hypertableau [Shearer et al., 2009], use entailment rules for ontology reasoning [Urbani et al., 2012]. In fact, the rule-based reasoning approach allows the parallelization and distribution of work by large clusters of inexpensive machines by programming models to process large data sets [Urbani, 2010]. However, implementations of Web-Scale Reasoners such as WebPie [Urbani et al., 2012] are limited to low expressive ontologies like OWL-Horst fragment [ter Horst, 2005] due to the complexity of implementation and performance at web-scale.

In Big Data context, the issue is not only limited to a huge volume of data, but is also related to the velocity stream of data with big variety. The underlying process of generating a data stream can be characterized as stationary or non-stationary. In a stationary stream it is assumed that the data distribution will not change over time, while the probabilistic properties of the data in a non-stationary stream will change over time [Gama et al., 2014]. The learning algorithms that assume a stationary data distribution, in general, produce a classifier that does not change over time, hence being called a static classifier. In many real-world scenarios, however, the data is non-stationary [Ditzler et al., 2015]. In non-stationary data distribution, the performance of a non-adaptive classification model trained under a false stationary assumption may degrade classification performance [Tsymbal, 2004]. On the other hand, adaptive learning approaches update classification models online during their operation regarding non-stationary data streams [Gama et al., 2014]. Therefore, the research question addressed in this thesis is:

> Is a hierarchical multi-label classification process used to automatically classify data items based on web reasoning maintainable over a large amount of data with great variety in constant generation (i.e. in Big Data context)?

In this context, the maintainability of the classification process is defined by the easiness with which the classification process can be used to classify Big Data. To this end, this thesis presents a new classification process that classifies data items from a large amount of non-stationary data in constant generation according to a domain model (e.g. an ontology) using a web reasoner. The classification process is unsupervised meaning no labels, previously classified examples nor rules to connect both data items and labels exist. It uses simple but scalable approaches through a parallel and distributed architecture to reach the aims of Big Data analysis.

The remaining of this Chapter is structured in three sections. In the next section 1.1 the scope of research is defined by making some ground assumptions and more clearly defining the approach to solve the defined problem. In Section 1.2, the outline of this thesis is provided, briefly describing the content of each chapter to ease the understanding and navigation throughout this document. Finally, Section 1.3 reports the external collaborations on the developed work.

1.1/ SCOPE OF RESEARCH

Big Data analysis can be deemed as the analysis technique for a special kind of data. The analysis of Big Data is the final and most important phase in the value chain of Big Data, with the purpose of extracting values that provide suggestions or decisions [Chen et al., 2014]. Therefore, many traditional data analysis methods such as Data Mining Algorithms (Classification, clustering, regression, among others) may still be utilized for Big Data Analysis [Chen et al., 2014]. As stated in the research question, the aim of this thesis is to study the maintainability of a classification process that classifies data items according to an ontology-described classification model using a web reasoner in the context of Big Data. Therefore three main research areas define the scope of research for this thesis (Fig. 1.1):

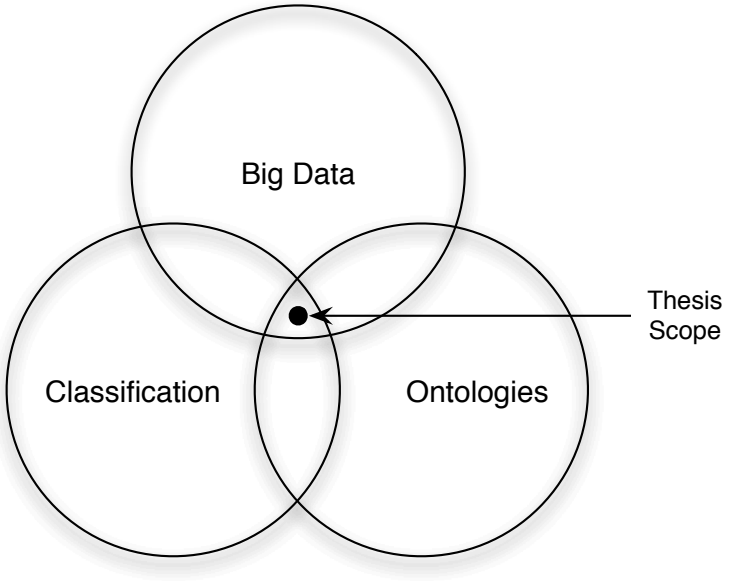


Figure 1.1 – Scope of research for this thesis

— **Big Data:** The term of Big Data is mainly used to describe enormous datasets characterized by an increasing number of V's (Volume, Velocity, Variety, Veracity, Value). Such amount of data requires new forms of processing to enable enhanced decision-making, insight discovery and process optimization. This thesis focuses on automatically classified data items in the context of Big Data.

— **Classification:** Classification is a machine-learning method used to predict different classes according to some constrains and to create a model to classify newly available data. In recent years, many approaches have been proposed to further improve classification performance by incorporating label correlations or exploiting label hierarchy. Among all existing types of classification, this thesis focuses on a specific type of classification that exploits connections in label hierarchy called Hierarchical Multi-Label Classification (HMC).

— **Ontologies:** Ontologies allow the definition of terms and meanings used to represent areas of knowledge. Ontologies are a good solution for intelligent computer

systems that operate close to a human concept level bridging the gap between human requirements and the computational requirements [Obrst, 2003]. From an ontology scope, this thesis aims to use ontologies to describe the classification model, ontology evolution to evolve the classification model according to data streams, and web reasoning to classify the items.

Since scalability is important in Big Data context, simple but highly scalable techniques are used in order to improve it compared to traditional classification approaches. The used techniques must allow the parallelization of the process and distribute it across several loosely coupled machines. The number of labels in many domains keeps growing during this, and even simple approaches can easily become computationally infeasible, not to mention the more sophisticated and computationally demanding approaches. To automatically analyse and describe data items in Big Data context, the number of labels can be even bigger.

To study the maintainability of the classification process, two main approaches of classification exist [Ditzler et al., 2015]:

— The first consists in learning the classification model from a static set of items (batch learning). In this case, a static classification model is created and all new documents are classified according to that classification model. Hence, no change to the classifications is made from one moment to another moment. E.g. once an item is classified with a set of labels, those labels will remain the same independently of the moment of the classification.

— The second consists in incrementally learning the classification model according to a stream of items. In this case, the classification model evolves with new documents used to train the classifier. Hence, the classification result for an item can change. E.g. an item classified in different moments can be classified with different labels.

Both approaches have advantages and disadvantages that make them suitable for different uses. The first approach is good for data whose statistical properties do not change significantly during time (stationary contexts). The main advantage of the first approach is that, once the classification model is created, it is used only once to classify each document. Once an item is classified, retrieving the classification for that item at any time is easy because no change occurs. The main disadvantage of this approach is that the process cannot learn with new documents (static training set). Therefore, the process cannot capture changes in non-stationary environments.

On the other hand, the main advantage of the second approach is the capacity to learn with new items and then capture the changes of non-stationary environments in the classification model. The main disadvantage is the necessary effort to update the classification model for each new document (or new group of documents) used to learn the classification model (dynamic training set). Also, in order to keep the classifications up-to-date, the items must be reclassified when a new document is included in the training collection at query-time, which can cause a process overload when querying huge amounts of data.

Furthermore, this work aims to use Web Reasoners to automatically classify items according to the learned ontology-described classification model. Current out-of-the-box reasoners can be used to classify small amount of data items but their scalability is limited and they cannot be used with large datasets as it is required in a Big Data context [Werner et al., 2014]. However, as Semantic Web is growing, new high-performance

"Web Scale Reasoning" approaches are emerging in literature (cf. Chapter 2 section 2.2) that are able to handle huge volumes of data by adopting rule-based reasoning instead of traditional description logics reasoning approaches such as the ones based on Tableau [Sirin et al., 2007, Tsarkov et al., 2006], Resolution [Motik et al., 2006] or Hypertableau [Shearer et al., 2009].

The classification approaches proposed in the following chapters address the identified problems and propose solutions to reach a maintainable classification process using rule-enriched ontologies and web reasoning in Big Data context.

1.2/ SUMMARY OF CHAPTERS

Nine chapters compose this thesis. The chapters are based on a collection of scientific papers that are either published or under review. This section outlines the overall structure of this thesis and provides a brief description of each chapter regarding the scientific papers.

The content of this thesis is divided in three main parts. The first part specifies the context of this thesis. Two chapters compose the first part:

— **Chapter 1**:The introduction describes context specification where the main objectives, contributions and research scope are defined.

— **Chapter 2**: In this chapter, the state of the art is described. The state of the art focuses on technologies that process Big Data, Ontologies, Web Reasoning and Classification - more specifically Hierarchical Multi-label Classification. However, the discussion of the related work for each specific contribution of this thesis is done in the beginning of the specific chapter.

The second part proposes a new hierarchical multi-label classification process to analyse large volumes of unstructured data in Big Data context. The process learns an ontology-described classification model from a static batch of data items and classifies the data items accordingly. Three chapters compose this second part. The first chapter proposes the hierarchical multi-label classification process that is described in detail in the following two chapters:

— **Chapter 3**: This chapter proposes a new hierarchical multi-label classification process called Semantic HMC to analyse huge volumes of unstructured data in Big Data context. The process is based on a non-supervised ontology-learning process using: (i) scalable Machine-Learning techniques to learn the ontology-described classification model and (ii) Rule-based reasoning to classify the items according to the learned classification model. The process is unsupervised, as no previously classified examples or rules exist to relate the data items with the labels. The ontologies play a key role in defining terms and meanings used to represent the knowledge, reducing the gap between the users and the HMC process. The content of this chapter is partially extracted from the following publication:

  - Hassan T, Peixoto R, Cruz C, Bertaux A, Silva N. Semantic HMC for Big Data analysis. In Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014, pages 26–28.

— **Chapter 4**: To perform a hierarchical multi-label classification in Big Data without overloading the users, this chapter proposes a new simple but highly scalable process to automatically learn the label hierarchy from huge sets of unstructured texts. The process is implemented using technologies for Big Data that distribute the process across several machines in order to reach high performance and scalability. The content of this chapter is largely extracted from the following publication:

  - Peixoto R, Hassan T, Cruz C, Bertaux A, Silva N. Semantic HMC : A Predictive Model using Multi-Label Classification For Big Data. In The 9th IEEE International Conference on Big Data Science and Engineering (IEEE BigDataSE-15).

— **Chapter 5**: This chapter proposes a new process to hierarchically multi-classify items from huge sets of unstructured texts using DL ontologies and Rule-based reasoning. The process is implemented using scalable approaches that distribute the process across several machines in order to reach the high performance and scalability required by Big Data. The process is evaluated and compared with some multi-label classification algorithms from the state of the art. The content of this chapter is extracted from the following publications:

  - Peixoto R, Hassan T, Cruz C, Bertaux A, Silva N. An unsupervised classification process for large datasets using web reasoning. In ACM, editor, SBD'16 : Semantic Big Data Proceedings, San Francisco (CA), USA.
  - Peixoto R, Hassan T, Cruz C, Bertaux A, Silva N. Hierarchical Multi-Label Classification Using Web Reasoning for Large Datasets. Open Journal of Semantic Web (OJSW) 2016

The third part focuses in maintaining a semantic hierarchical multi-label classification process like the Semantic HMC regarding streams of non-stationary data, such as those occurring in Big Data contexts. The first chapter proposes a new hierarchy maintenance process that is used in the new adaptive process proposed, instantiated and evaluated in the following chapters:

— **Chapter 6**: This chapter addresses the specific problem of maintaining automatically learned hierarchical relations. This is a complex problem and is studied alone in this chapter before proposing the adaptive process. To address this problem, this chapter proposes a process for maintaining hierarchical relations of an automatically learned ontology-described concept hierarchy regarding a stream of unstructured text documents in Big Data context. The hierarchical maintenance process is implemented using technologies for Big Data that distribute the process over several machines in order to reach high performance and scalability. The

content of this chapter is partially extracted from the following publication:

- Peixoto R, Cruz C, Silva N. Semantic HMC : Ontology-Described Hierarchy Maintenance in Big Data Context. In On the Move to Meaningful Internet Systems : OTM 2015, pages 492–501. Springer International Publishing - Lecture Notes in Computer Science.

— **Chapter 7**: This chapter proposes a new adaptive learning process to consistently adapt an ontology-described classification model used for a hierarchical multi-label classification regarding a non-stationary stream of unstructured text data in Big Data context. The process is instantiated for the specific problem of the Semantic HMC. The adaptive process is implemented using technologies to process Big Data that distribute stream processing over several machines to reach high scalability. The experiments performed with the adaptive classification model process are done by comparing the classification quality generated through a static classification model, and the one generated through the classification model adapted according to the proposed adaptive process. The content of this chapter is partially extracted from the following publication:

- Peixoto R, Cruz C, Silva N. Adaptive Learning Process for the Evolution of Ontology-described Classification Model in Big Data Context. In Proceedings of 2016 Science and Information Conference, SAI 2016, London, UK.

A single chapter composes the third and last part of this thesis which is Chapter 8. In this chapter the contribution presented in the previous chapters is summarized in some generic conclusions. Some future work is also proposed as well as some principles that can be used as a guideline for further research that addresses similar challenges.

1.3/ COLLABORATIONS

A large part of the content of this thesis is extracted from scientific publications written in collaboration with other researchers as is described in the previous section.

More in particular, the contribution presented in the second part of this thesis (Chapter4 and Chapter 5) is a work developed in conjunction with Thomas Hassan who has developed the Semantic HMC process in a distributed platform to process Big Data and also ran the experiments.

2

STATE OF THE ART

This chapter presents the existent approaches and solutions related to the problem. This state of the art chapter first focuses on describing Big Data and the main technologies used to process, store and analyse Big Data (Section 2.1). Later it presents the state of the art in Ontologies, Ontology Evolution and Web Reasoning (Section 2.2). Finally it describes the state of the art in Classification, more specifically Hierarchical Multi-label Classification, and the use of Ontologies in Classification (Section 2.3). A global discussion is also presented at the end of this chapter (Section2.4). However, the discussion of the related work for each specific contribution of this thesis is done in the beginning of its specific chapter.

2.1/ BIG DATA

The expression "Big Data" is mainly used to describe huge datasets characterized by an increased number of characteristics (Volume, Velocity, Variety, Veracity and Value among others). Big Data is also created by multiple types of sources. Among them, handheld devices, social networks, internet of things and multimedia are some of the most important sources. Such amount of data requires new forms of processing to enable enhanced decision making, insight discovery and process optimization. An increasing number of Vs has been used to characterize Big Data [Chen et al., 2014, Hitzler et al., 2013, Khan et al., 2014]. In literature, three Vs (Volume, Velocity, Variety) are currently used to characterize the essence of Big Data [Chen et al., 2014, Qin, 2014, Syed et al., 2013]:

— **Volume:** concerns the large amount of data that is generated and stored through the years by social media, sensor data, etc. [Chen et al., 2014]. Estimates [Troester, 2012] show that the amount of data in the world is doubling every two years. Should this trend continue, by 2020 there will be 50 times the amount of data there was in 2011.

— **Velocity:** concerns both the production and the process to meet a demand because Big Data is not only a huge volume of data but its generation is also fast. The proliferation of digital devices such as smartphones and sensors led to a huge rate of data creation. To handle such rate of data creation high velocity analytics (i.e. real-time analytics) are needed. E.g., even conventional retailers like Wal-Mart process more than one million transactions per hour [Cukier, 2010] which is just a fraction of transactions in Big Data context.

— **Variety:** refers to the various types of data composing the Big Data. These types include semi-structured and unstructured data representing 90% of its content [Syed et al., 2013]. Images, audio, video and text are examples of unstructured data, which lack the structural organization required for machine analysis. Semi-structured data lies between the fully-structured data (i.e. Relational Databases) and unstructured data. It has some structure but does not conform to strict standards (i.e. Extensible Markup Language (XML) documents contain user-defined data tags which make them machine-readable).

In addition to these, some have included additional characteristics (Vs) in Big Data [Hitzler et al., 2013, Syed et al., 2013, Khan et al., 2014]. One of the most recent and complete Big Data characterization uses 4 additional Vs (Khan et al., 2014):

— **Veracity:** concerns the truthfulness in data. In traditional data warehouses there was always the assumption that the data was certain, clean, and precise but in Big Data context, user-generated data can feature uncertain or imprecise data.

— **Validity:** concerns the correctness and accuracy of data regarding its intended usage [Khan et al., 2014]. Data may not have any veracity issues but may be valid for one application or usage and then invalid for another application or usage.

— **Volatility:** concerns the data retention period. The data retention period is the period in which the data will be stored. After this period expires, the data can be destroyed without negative consequences for the system. For example: an online e-commerce company may not want to keep a 1 year customer purchase history. After that period, the data is automatically destroyed [Khan et al., 2014].

— **Value:** measures how valuable the information is to a Big Data consumer. Value is the most important feature of Big Data and is the desired outcome of Big Data processing [Khan et al., 2014] because if data doesn't have value then it is useless.

How to determine relevancy among the large volume of data and create value from data is one of the main challenges of Big Data. Sheth [Sheth, 2014] proposes deriving Value by harnessing the challenges posed by Volume, Variety, and Velocity using Semantic Techniques and Technologies (Fig. 2.1). This requires organized ways to harness and overcome the challenges proposed by each V characteristic of Big Data through the use of metadata and employing semantics and intelligent processing. An IDC report [Gantz et al., 2011] also proposes value extraction from very large volumes of a wide variety of data by enabling high-velocity capture, discovery, and/or analysis.

2.1.1/ Big Data Analysis

The ultimate goal of Big Data is to provide business solutions that can help a company gain in business solutions. This reason itself makes the analysis of Big Data important. One of the main challenges is finding how to determine relevance among large volumes of data and create value from relevant data. Since Big Data has the particular characteristics of volume, velocity, variety, veracity and value it may change the standard statistical

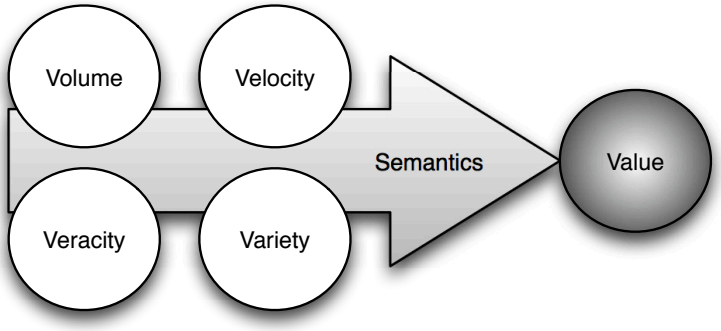


Figure 2.1 – Deriving value in Big Data

and data analysis approaches [Ma et al., 2014]. Therefore, several new issues for data analytics come up, such as privacy, security, storage, fault tolerance, and quality of data [Tsai et al., 2015].

According to Tsai et al. [Tsai et al., 2015] the limitations of the data analysis methods used in Big Data context can be described in three types:

— **Unscalability and centralization:** Traditional methods for data analysis are not designed to be scalable for huge volumes of data. The design is typically centralized and meant to be performed locally in a single machine containing all the data. Differently from traditional data analytics, in Big Data's case the data is ever-growing (e.g. wireless sensor network data analysis [Baraniuk, 2011]. This is because data producers like sensors can gather much data, but uploading such large data to traditional analytic approaches may create bottlenecks. To overcome this limitation, analytic approaches must scale, for example, by distributing the process across multiple machines. Another example is that in order to store huge volumes of data, traditional relational databases have been relaxed to more simple persistent models such as NoSQL.

— **Non-dynamic:** Traditional data analysis methods cannot be dynamically adjusted to different situations, meaning that they do not analyse the input data on the fly. For example, the classifiers are usually stationary and do not adapt the generated classification model to new data. The adaptive learning [Gama et al., 2014] is a promising research trend because it can dynamically adjust the classifiers according to streams of data.

— **Uniform data structure:** Most data mining problems assume that the format of the input data will be the same. Therefore, the traditional data mining algorithms may not be able to deal with the problem that the formats of different input data may be different and some of the data may be incomplete. Finding how to make the input data from different sources have the same format will be a possible solution to the Big Data variety problem.

Because the traditional data analysis methods are not designed for large-scale and complex data, they are almost unable to analyse Big Data. Redesigning how the data analysis methods are designed is a critical trend in Big Data analysis [Tsai et al., 2015].

Various solutions have been presented to process Big Data [Dean et al., 2008, Shvachko et al., 2010, Anil et al., 2010, Vora, 2011, Toshniwal et al., 2014]. Most of the studies on data analysis are focused on designing efficient approaches to find relevant information/knowledge/value from the data. However, in Big Data era, most existent computer systems are not able to process huge volumes of various types of data generated at high velocity at once. To handle the dimensions of Big Data, new technologies have been developed. Big Data technologies can be divided into three main topics:

— **Process:** Technologies that distribute the data process for a cluster of commodity hardware (not so expensive machines), (e.g. Hadoop/MapReduce [Dean et al., 2008] or Apache Storm [Toshniwal et al., 2014])

— **Storage:** Technologies meant to store very large files, running on clusters of commodity hardware (e.g. HDFS [Shvachko et al., 2010], Hbase [Vora, 2011]).

— **Analysis:** Technologies that provide scalable and intelligent data analysis for Big Data (e.g. Mahout [Anil et al., 2010]).

Cloud computing technologies are widely used to execute Big Data technologies in order to satisfy the large demands of computing power and storage.

The following subsections describe the technologies used to process, store and analyse Big Data in this thesis.

2.1.2/ TECHNOLOGIES TO PROCESS BIG DATA

In this subsection, some technologies from the state of the art regarding Big Data processing are described. One of the most emblematic technologies used to process Big Data is MapReduce. MapReduce is a programming model for processing and generating large data sets introduced by Google [Dean et al., 2008] and used in Hadoop. MapReduce's main contribution is to enable the parallelization and distribution of large datasets processing on large clusters of inexpensive machines. The programming model is based on two distinct phases (Fig. 2.2):

— The Map phase that executes a function that uses input data to generate a set of intermediate key/value pairs. $(k1, v1) -> list(k2, v2)$

— The Reduce phase merges the map output values that share the same key allowing all associated records to be processed together in the same node. The Reduce phase's output forms a possibly smaller set of values. $(k2, list(v2)) -> list(v3)$

The Map and Reduce functions are defined by the user. In other words, the user will define what the map and reductions functions do.

Different implementations of MapReduce are possible depending on the environment (shared-Memory, MultiProcessor, Network Clusters). One possible implementation is proposed by Dean et al. ([Dean et al., 2008]. This implementation targets large clusters connected by network depicted in Fig. 2.3.

The proposed MapReduce implementation in [Dean et al., 2008] has the following steps:

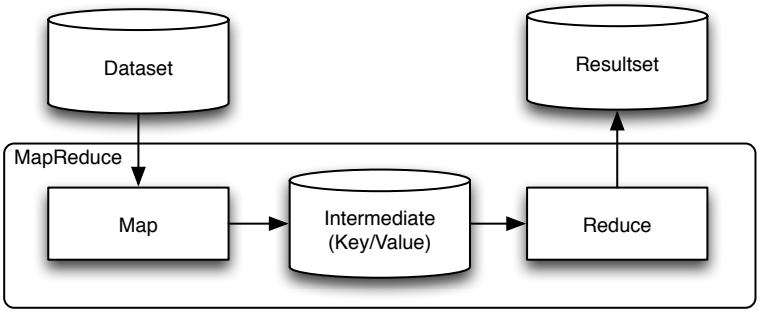


Figure 2.2 – MapReduce Overview

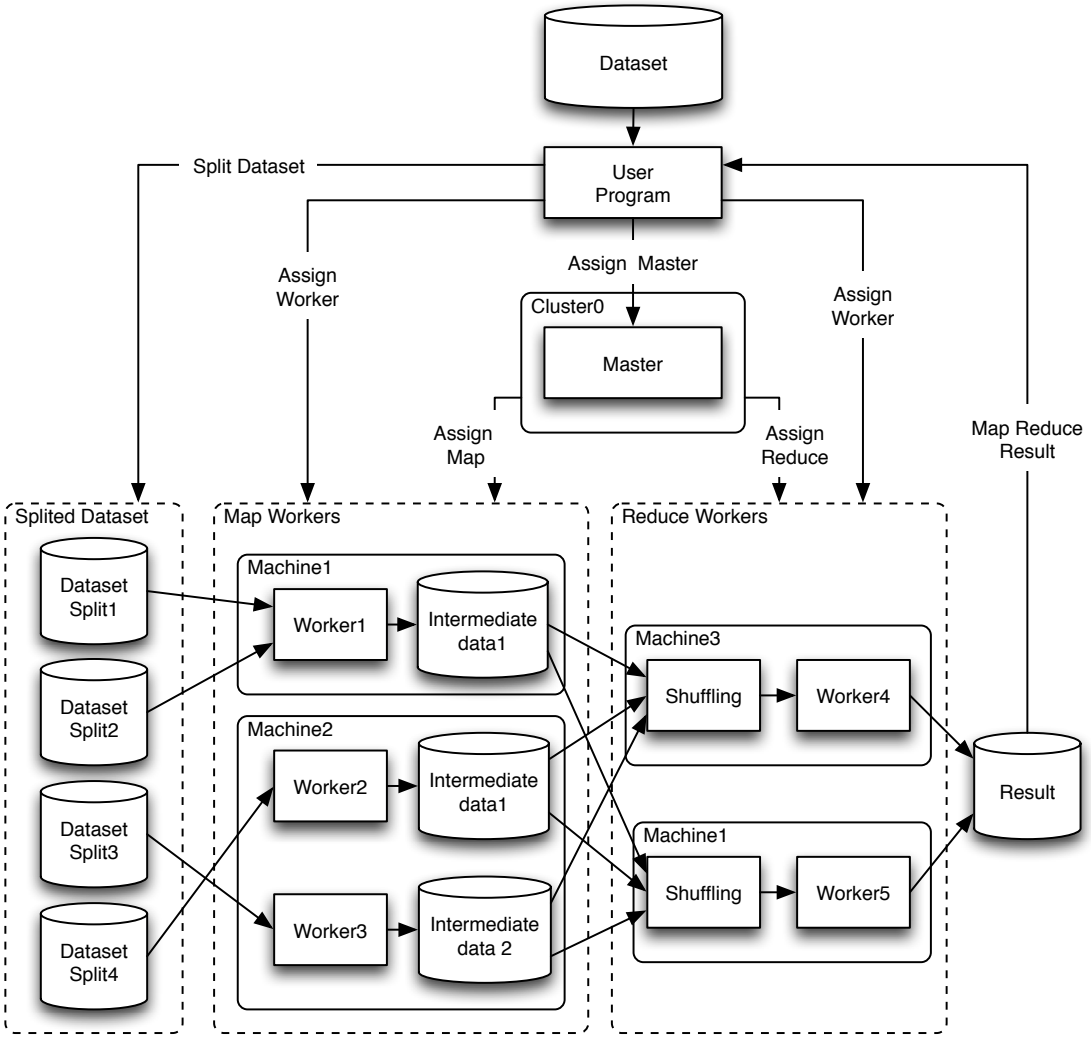


Figure 2.3 – MapReduce Implementation Model

1. Firstly the input Dataset is split by the user program into subsets.

2. The user program starts many copies of the MapReduce program in a cluster of machines. One of the copies of the program is defined as Master. The Master copy

then picks the idle worker and assigns a Map task or a Reduce task. The rest are workers that will be assigned by the Master as Map tasks and Reduce tasks.

3. A worker that gets a map task reads the contents of the corresponding split input. It parses key/value pairs out of the input data and passes each pair to the user-defined map function. The intermediate key/value pairs produced by the map function are buffered in memory or stored locally in the machine disk.

4. The location of Intermediate data is passed to the Master, which will forward them to the Reduce workers.

5. The reduce worker is notified by the Master about the locations and then uses remote procedures in order to read the intermediate data from the map worker. The reduce worker then sorts the intermediate data by key in order to group data that shares the same key.

6. Reduce worker then iterates over the data, and for each key, passes the key and the corresponding set of values to the Reduce function. The output is appended to the output file of this reduce partition.

7. When execution is completed, the master worker returns the handle to the program

A simple problem used by authors in [Dean et al., 2008] to explain how MapReduce works in practice consists in counting the occurrences of single words in documents. The input text is converted into a sequence of <key-value> tuples where all the words of the text are keys and values. For example, the sentence “Hello world” is converted into 2 tuples, each of them containing one word of the sentence as key and as value. The user-defined algorithm to count occurrences of single words within a text is presented in Algorithm 1. The map algorithm receives tuples in the form of <word-word> and transforms the original tuple in the form of <word, 1> where the key of this tuple is the word itself while 1 means one occurrence. The reduce function will receive a word (key) and a list with all similar words (values). Then, the reduce function counts the number of times that word appears in the text and returns the result in the form of <word, number of occurrences>

Listing 2.1– Map and Reduce Functions

```
1  map(key, value):
2    key: word
3    value: word
4    output.collect(key ,1)
5
6  reduce(key, iterator values):
7    count = 0
8    for (value in values)
9    count = count + 1
10   output.collect( key , count )
```

A more concrete example about counting occurrences of single words is presented in Fig. 2.4. The inputs are words about vehicles (Car, Bus, Truck, Bike). First, the two input texts are split by line and turned into the form of <word, word>. Then the map function is called for each <word, word> tuple. The map output will be in the format of <word, 1>. The map functions of each split line can be executed in distinct clusters. In order to group the mapping output by key, a Shuffling function is used. The reduce function will then be

called for each shuffled group. These functions can be executed in distinct clusters and will count the number of occurrences of each word.

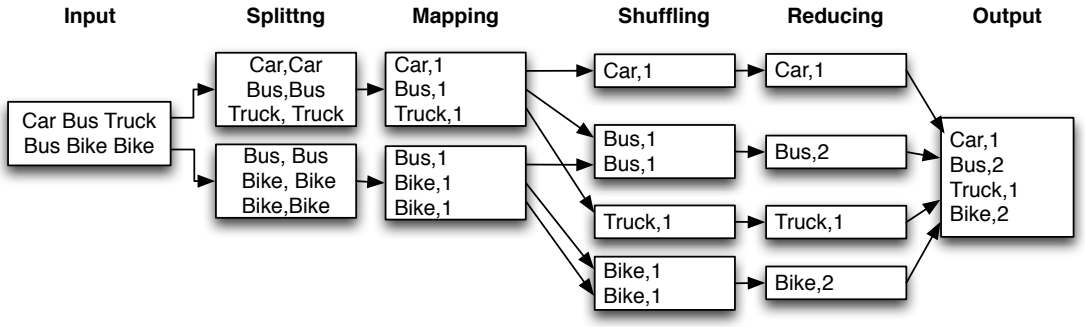


Figure 2.4 – Word count example using map-reduce

An open-source implementation of the MapReduce paradigm is provided by the Apache Hadoop framework [The Apache Software Foundation, 2012]. Hadoop uses MapReduce to process large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than relying on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, then delivering a highly available service on top of a cluster of computers, where each may be prone to failures.

The MapReduce paradigm implemented in a Hadoop cluster is good to process huge volumes of batch data; however, it was designed to process static data and cannot be efficiently applied to the process of data streaming [Jones, 2012]. Apache Storm[1] is a scalable, distributed and fault tolerant real time computation framework for processing large data streaming. In order to do stream computation on Storm, the user must create "topologies" (Fig. 2.5). A topology is a directed graph of computation components called "Bolts" that contain processing logic to process Tuples (Storm data item abstraction). Bolts process the Streams of Tuples (unbound list of Tuples) from Spouts (source of a Stream) so as to create new Streams.

The Storm cluster is similar to the Hadoop cluster, except the storm cluster runs different topologies for different storm tasks, while the Hadoop platform runs MapReduce jobs for corresponding applications. The main difference between a MapReduce job and a topology is that the MapReduce job eventually ends, whereas a topology keeps processing messages until a user terminates it [Toshniwal et al., 2014].

The Storm cluster is partitioned and distributed to a number of worker processes where each worker implements a part of the topology. Two types of nodes compose a storm cluster (Fig. 2.6): a nimbus node (master) and slave nodes. Nimbus is in charge of distributing code across the storm cluster, scheduling and assigning tasks to slave nodes, and monitoring the whole system. The slave node is composed by a supervisor and multiple workers. The supervisor distributes the process by multiple workers. The supervisor is started or stopped by the nimbus node as necessary to comply with the assigned tasks. All coordination between nimbus and the supervisors is done through an Apache

[1] http://storm.apache.org/

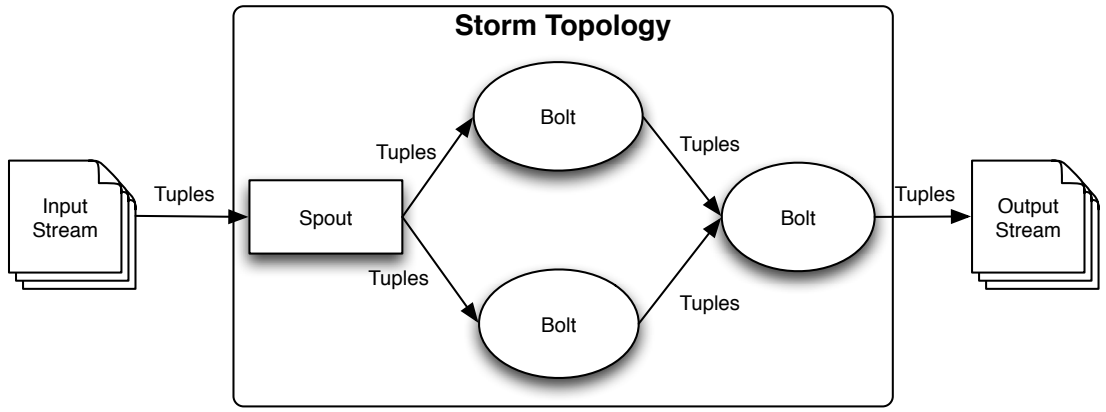


Figure 2.5 – Storm Topology

ZooKeeper [2] server. Apache Zookeepter is an open-source server which enables highly reliable coordination for distributed applications.

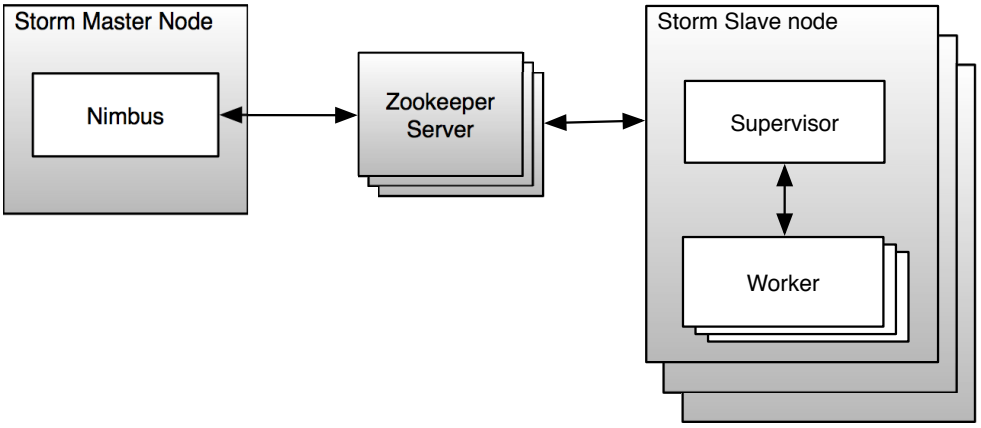


Figure 2.6 – Storm Architecture

Other technologies recently emerged, such as Spark[3], which has the general purpose of processing huge batches of data. In Spark, the workflows are defined in a similar style to MapReduce but it claims to be more efficient in some situations than the implementation of MapReduce is in the Hadoop framework. Apache Spark also has a Streaming API that allows continuous data processing via short interval batches. Differently from Storm, which has the ability to perform data computations in real time, Spark Streaming treats streaming computations as a series of deterministic batch computations on small time intervals (micro-batching). On one hand, Spark Streaming has advantage over Storm regarding the processing of stream data items with a delay between data item generation and processing. This is done by using user-defined time intervals (micro batch of items). One the other hand, Storm has the advantage of processing each incoming data item in real time.

[2] http://zookeeper.apache.org
[3] http://spark.apache.org

2.1.3/ TECHNOLOGIES TO STORE BIG DATA

In this subsection, some state-of-the-art technologies to store Big Data are described. When a dataset grows beyond the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. Hadoop Distributed File System (HDFS) [Shvachko et al., 2010] is the file system used in the Hadoop framework. It is "*a distributed file system that provides high-throughput access to application data*" [The Apache Software Foundation, 2014]. HDFS works in the same way as MapReduce by using a master-worker pattern. HDFS has three types of nodes as depicted in Fig. 2.7 [Shvachko et al., 2010]:

— **Namenode:** is a master node that manages the file system namespace, maintaining the file system tree and the metadata for all files and directories.

— **Datanodes:** (workers) store and retrieve blocks and periodically report the list of blocks that they are storing back to the namenode.

— **Clientnode:** refers to the client that accesses the filesystem on behalf of the user by communicating with the namenode and datanodes.

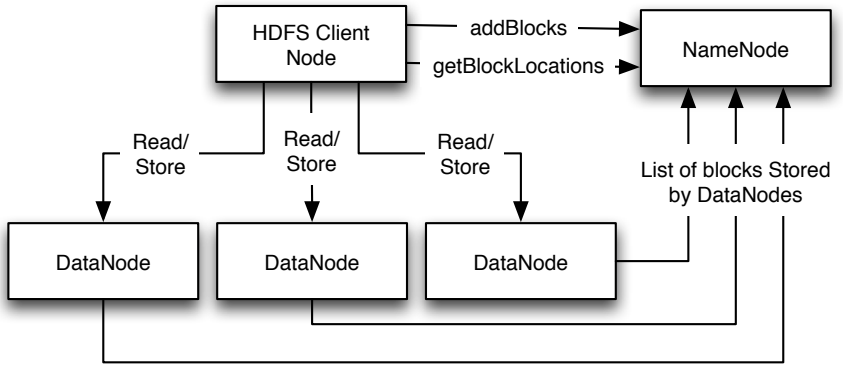


Figure 2.7 – HDFS Nodes

HBase[4] is an open source, distributed, fault-tolerant, non-relational (NoSQL) database that runs on top of HDFS. It provides real-time read/write access to large datasets with linear scalability to billions of rows and millions of columns. It is natively integrated with the Hadoop server and can conveniently be used to backup MapReduce jobs.

Hbase has two main components coordinated by a Zookeeper: HMaster servers and Region Servers. ZooKeeper HMaster servers make information about the cluster topology available to clients. HBase requires that all tables must have a primary key. The table is divided into sequential blocks that are then allotted to a region. Region Servers have one or more regions to spread the load uniformly across the cluster. Clients connect to ZooKeeper to download the list of RegionServers, their regions and the table key hosted by the regions. Thus the clients can directly access the RegionServers in order to access the data without any central coordination (Fig. 2.8).

[4] http://hbase.apache.org

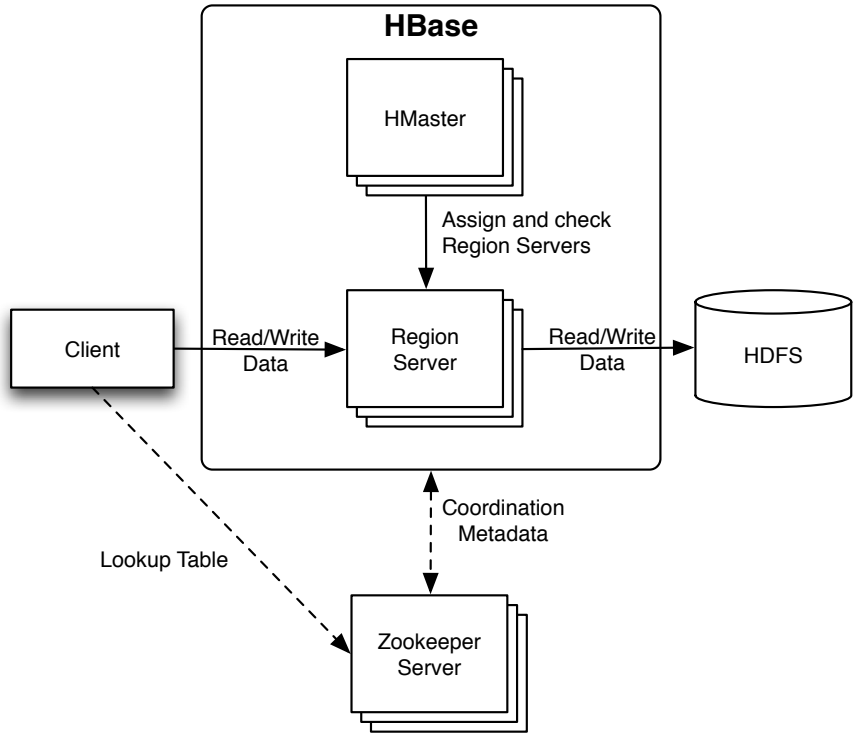


Figure 2.8 – HBase Architecture

Apache Kafka[5] is a distributed messaging system that stores feeds of messages in topics. Messages are simple byte arrays which allow storing any object in any format (e.g. String, JSON, etc.). A topic is a category name where messages are stored. For each topic, the Kafka cluster maintains a partitioned log sorted from the oldest messages to the most recent. A sequential id number called the offset is assigned to each message, uniquely identifying each message within the partition. This offset is controlled by the consumer and is normally advanced linearly as the consumer reads messages. However, if the consumer is restarted, the offset is lost. All published messages are stored for a configurable period of time whether or not they have been consumed. Producers write (push) data to topics and consumers read (pull) from topics (Fig. 2.9). Since Kafka is a distributed system, topics are partitioned and replicated across multiple nodes called brokers.

The partitions are distributed over the brokers in the Kafka cluster. Each broker handles data and requests for a share of partitions. Each partition is replicated across a configurable number of brokers to allow fault tolerance. In the case of replication, for each partition one broker acts as the "leader" and the others act as the "followers". The leader handles all requests for the partition while the Kafka cluster can be used, for example, in conjunction with Apache Storm so as to distributively store and process streams of data. In this case, Storm Spouts are used to read from the Kafka topics (as consumers) and bolts are used to write to the Kafka topics (as producers).

[5]http://kafka.apache.org

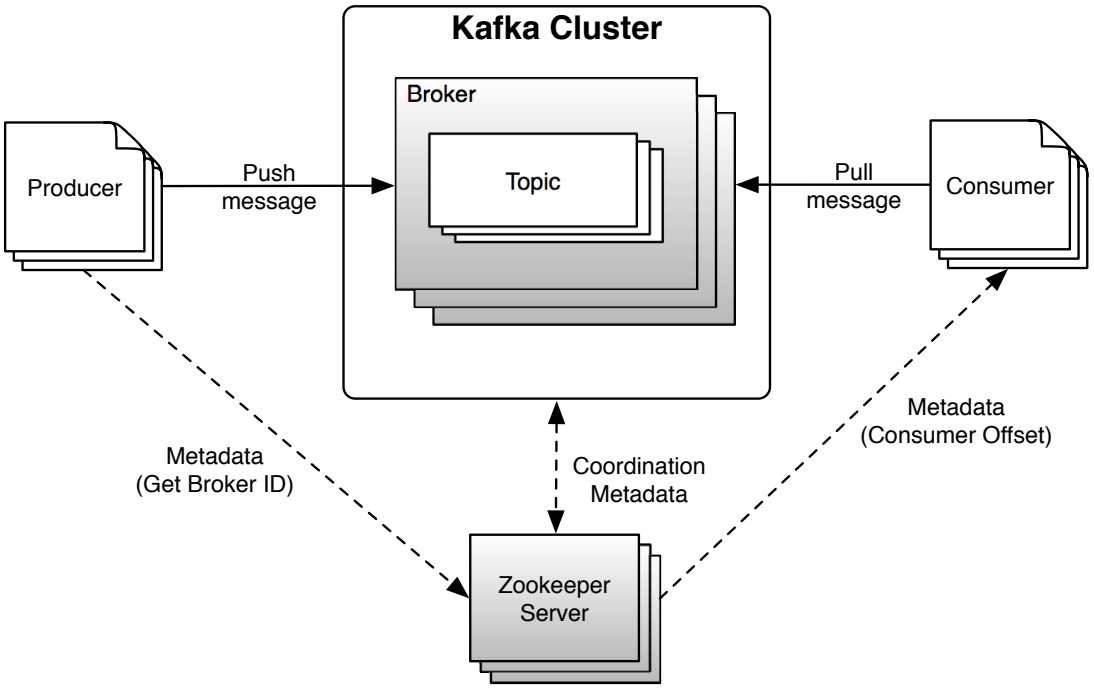


Figure 2.9 – Kafka Cluster Architecture

2.1.4/ TECHNOLOGIES FOR BIG DATA ANALYSIS

Some state-of-the-art technologies used to analyse Big Data are described in this section. Mahout is an open source machine-learning library from Apache [Anil et al., 2010]. Apache Mahout aims to provide scalable and commercial machine learning techniques for large scale and intelligent data analysis applications. The algorithms it implements fall under the broad umbrella of machine learning or collective intelligence. While this might have multiple interpretations, in this context it means that Mahout proposes algorithms mainly for collaborative filtering, clustering and classification.

Mahout aims to be the machine learning tool of choice whenever the collection of data to be processed is very large, perhaps far too large for a single machine. In its current incarnation, these scalable machine learning implementations in Mahout are written in Java, while some portions are built upon Apache's Hadoop distributed computation project and others over Apache Spark.

Mahout can be used on a wide range of classification projects, but the advantage of Mahout over other approaches becomes striking as the number of training examples gets extremely large. What "large" means can vary enormously. When up to about 100,000 examples, other classification systems can be efficient and accurate. But generally, as the input exceeds 1 to 10 million training examples, Mahout exceeds all other traditional methods in performance [Anil et al., 2010].

The core algorithms of Mahout include clustering, classification, pattern mining, regression, dimensionality reduction, evolutionary algorithms and batch-based collaborative filtering. These are running on top of Hadoop and most recently Spark platforms.

In Big Data context, MapReduce is not suitable to express streaming algorithms, and

traditional sequential online algorithms are limited by the memory and bandwidth of a single machine. To run data mining and machine learning algorithms on a distributed stream processing engine, A. Biffet [Bifet et al., 2014] has proposed SAMOA (Scalable Advanced Massive Online Analysis), an open-source platform for Big Data streams mining. It provides a collection of distributed streaming algorithms for the most common data mining and machine learning tasks such as classification, clustering, and regression. It uses distributed stream processing engines to express parallel computation on streams, and combines the scalability of distributed processing with the efficiency of streaming algorithms.

2.2/ Ontologies and Web Reasoning

Ontologies [Gruber, 1993] are the most accepted way to represent semantic information in Semantic Web. The Semantic Web [Berners-Lee et al., 2001] is an extension of the World Wide Web where semantic information can be interpreted by machines. Semantic Web is growing and in order to handle this increasing amount of semantic web data, high-performance reasoning methods are required [Urbani, 2013]. With the Semantic Web, reasoning over large and complex ontologies (Large-scale reasoning) became even bigger and can be compared with the entire size of the Semantic Web, being recently called “Web-scale Reasoning” [Urbani, 2013]. This section describes and systematizes basic concepts of Ontology, Ontology Evolution and reasoning focusing in Large Scale Reasoning.

2.2.1/ Ontologies

The word “Ontology” has different meanings in different communities. In computer science, there are many definitions of the word ontology. The most popular definition is by [Gruber, 1993] who defines Ontology as “an explicit specification of a conceptualization”. This definition is further extended to “Ontologies are a formal, explicit specification of a shared conceptualization” by [Studer et al., 1998]. Ontology allows the definition of terms and meanings used to represent areas of knowledge. Ontologies are extremely important in the interaction between systems that constantly exchange knowledge between them. The proper communication of these systems will only be achieved when both systems receive the same interpretation of the implicit knowledge of the documents exchanged. Ontology is generally designed to:

— Enable the use of a semantic knowledge and application;

— Facilitate the knowledge sharing process between computers;

— Allow the correct semantic interpretation.

To achieve every previous feature, ontology defines terms and concepts in order to describe and represent specific knowledge domains. The terms and concepts follow a conceptual modeling usually composed by classes, properties and their hierarchical relationships. As conceptual models become more restrict and rules become more complex, the

Description Logic (DL) concept gained importance in providing a logical formalism for Ontologies [Baader et al., 2001, Nardi et al., 2003]. Description logics (DLs) are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way [Baader et al., 2009].

Description Logics (DLs) are a family of knowledge representation formalisms that represent the knowledge of an application domain by first defining the relevant concepts of the domain and then using these concepts to specify properties of objects and individuals occurring in the domain [Baader, 2003]. Description Logics assume the Open World Assumption which states that the absence of knowledge in the Knowledge Base does not imply that it is false. This assumption allows the system to differentiate between incomplete information and information that is known to be false. A DL knowledge base consists in [Baader et al., 2009] :

— a Terminological Box (TBox) that contains the definitions of the conceptual terms.

— a Assertional Box (ABox) that contains a set of membership instances and role assertions.

Attributive Language (AL) is a minimal Description Logic language where negation can be applied only to atomic concepts and only the top concept can be used over the existential quantification [Baader, 2003]. Attributive Language with Complements (ALC) includes the AL language but negation and existential quantification are not limited. In ALC, concepts are constructed using ($\sqcap$, $\sqcup$, $\neg$), existential value restriction ($\exists$), and universal restriction ($\forall$)[Baader, 2003]. Due to the suitability of DLs as ontology languages, several ontology languages have been created based on DL, notably the Web Ontology Language (OWL) [Antoniou et al., 2009].

The Resource Description Framework (RDF) [Brickley et al., 2004] is an infrastructure that enables and promotes the encoding, exchange, and reuse of structured metadata. Each RDF statement is made of three different terms (common referred as triples) [Manola et al., 2004]:

— Subject identifies the thing that the statement describes,

— Predicate is the relationship between the subject and the object,

— Object identifies the property value.

RDF can describe structured information but cannot specify the meaning of that information. To support semantic information, RDF has been extended as RDF Schema (RDFS) [Brickley et al., 2004]. RDFS provides a way to develop shared vocabularies describing resource classes and properties allowing classes and properties in hierarchies to be arranged. RDFS Inference is possible through Entailment Rules (Hayes, 2004). Entailment is the relationship between two sentences where the truth of one (A) requires the truth of the other (B).

RDFS defines relations between the class hierarchy and the properties or their domain and range, but the community needed a language that could be used for more complex ontologies. The Web Ontology Language (OWL) [Antoniou et al., 2009] extends the

RDFS vocabulary with additional resources that can be used to build more expressive ontologies for the web. The OWL language has two specifications: the original OWL and the OWL 2 specification [W3C OWL Working Group et al., 2009]. Both the original OWL specification and OWL 2 provide sub-languages that are characterized by their expressivity. The original OWL recommendation consists of three sub-languages with increasing expressivity [Antoniou et al., 2009]:

— **OWL-Full:** The entire language is called OWL Full, and uses all OWL language primitives. As a disadvantage due to its lack of constricts, OWL-Full is non-decidable, giving no guarantees on complete and efficient reasoning.

— **OWL-DL:** Is a sub-language of OWL-Full, which restricts the way in which constructs can be used, allowing decidability of the ontology and efficient reasoning support.

— **OWL-Lite:** An even further restriction limits OWL DL to a subset of the language constructors. For example, OWL Lite excludes enumerated classes, disjointed statements and arbitrary cardinality (among others). The advantage of this is a language that is both easier to grasp and easier to implement. The disadvantage is the restricted expressivity.

The OWL 2 specification, like the original OWL language, aims to produce a subset of OWL with different expressivity having in mind the user communities and implementation technologies [Shearer et al., 2009]. The three OWL 2 profiles are:

— **OWL EL:** provides expressive features of OWL required by large-scale ontologies while also eliminating unnecessary features, providing only existential quantification. This profile is designed to provide polynomial time computation for determining the consistency of an ontology and mapping individuals to classes.

— **OWL QL:** profile addresses the response of conjunctive queries in log-space regarding the number of assertions in the knowledge base that is being queried. This profile provides polynomial time computation for determining ontology consistency and mapping individuals to classes.

— **OWL RL:** aims at scalable reasoning without sacrificing too much expressive power. Using rules and a rule processing system can do this profile implementation. Part of the design of the OWL RL is that it only requires the rule processing system to support conjunctive rules.

OWL-Horst [ter Horst, 2005] is an OWL fragment with expressivity $pD*$ that defines semantic extensions of RDFS that involve data types and a subset of the OWL vocabulary that includes the property-related vocabulary, comparisons and value restrictions.

2.2.2/ Ontology Evolution

Like other software models notably database schemas, ontologies inevitably change over time [Noy et al., 2004]. *An ontological change is an action performed over an ontology*

whose result is an ontology different from the original version [Noy et al., 2004]. Nowadays ontologies are increasingly used in mainstream applications that change ontologies during their lifetime. These applications require a new research focus in ontology engineering that supports the complete lifecycle of ontologies, beyond the initial steps of acquisition, development and deployment [Zablith et al., 2013].

Ontology evolution is not *an atomic, well-defined and self-contained notion* (Zablith et al., 2013) and can be defined in various ways:

— [Noy et al., 2004] combines ontology evolution and versioning into a single concept defined as "*the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology*".

— [Haase et al., 2005] sees ontology evolution as the process to "*adapt and change the ontology in a timely and consistent manner*".

— [Flouris et al., 2007] defines ontology evolution as a process aiming to "*respond to a change in the domain or its conceptualization*" by implementing a set of change operators over a source ontology.

— The more recently compiled, the NeOn Glossary of ontology engineering tasks states that ontology evolution is "*the activity of facilitating the modification of an ontology by preserving its consistency*" [Zablith et al., 2013].

An ontological change is *an action performed over an ontology whose result is an ontology different from the original version* [Klein et al., 2003, Noy et al., 2004]. Based on the most common literature, [Flouris et al., 2007] identified and studied ten subfields of ontology change.

The ontology evolution process proposed by [Stojanovic et al., 2002] is a six-phase cyclic process (Fig. 2.10).

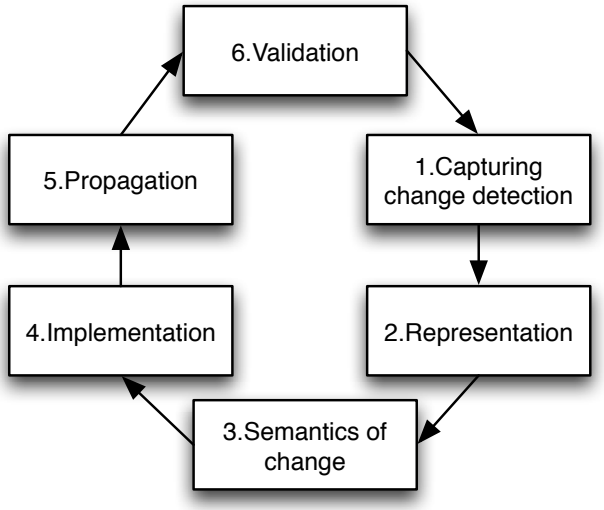


Figure 2.10 – Ontology Evolution Process By [Stojanovic et al., 2002]

1. Capturing changes to be applied to the ontology. Three types of changes are identified: (a) Usage-driven change discovery (i.e., derived from user behaviour), (b) Data-driven discovery (i.e., derived from changes to the ontology instances) and (c) Structure-driven change discovery where changes are derived from the analysis on the ontology structure.
2. Representation process providing a way to represent the changes in a specific model called “evolution ontology”.
3. Semantics of change process addressing syntactic and semantic inconsistencies that could arise as a result of the changes.
4. Implementation process coupled with user interaction to approve or cancel changes.
5. Propagation process allowing the update of outdated instances and reflecting the impact in referenced ontologies.
6. Validation to check that the performed changes led to the desirable result allowing the user to accept or reject changes.

A framework is presented by [Klein et al., 2003] to support users when the ontology evolves from one version to another through the following ontology evolution tasks:

1. Data transformation, where data in the old ontology version is transformed into a format that is compatible with the new ontology version;
2. Data Access retrieves all data that was accessible via queries on the old ontology and transforms these queries in the new ontology’s terms. Instances of concepts in the old ontology should be instances of equivalent concepts in the new ontology.
3. Ontology update adapts a remote ontology to specific local needs, and the remote ontology changes, having to propagate the changes in the remote ontology to the adapted local ontology;
4. Consistent reasoning analyses the changes to determine whether the axioms valid in the old ontology are also valid in the new ontology in order to keep the ontology consistent;
5. Verification and approval where ontology developers verify and accept or reject the ontology changes.

More recently a systematization of the Ontology evolution cycle was presented by [Zablith et al., 2013] where they intend to abstract the cycle from the specificity of each previous framework. The Evolution Cycle (Fig. 2.11) has five main stages:

1. Detecting the Need for Evolution stage starts the ontology evolution process by detecting a need for change derived from user behavior or data sources. This stage corresponds to the Change Capturing step in the [Stojanovic et al., 2002] approach.
2. TheSuggesting Changes stage suggests and represents changes to the ontology. This is the Representation step of [Stojanovic et al., 2002] and the Data Transformation step of [Klein et al., 2003]. These changes can be represented by structured (i.e. ontologies) or unstructured sources (i.e. text).

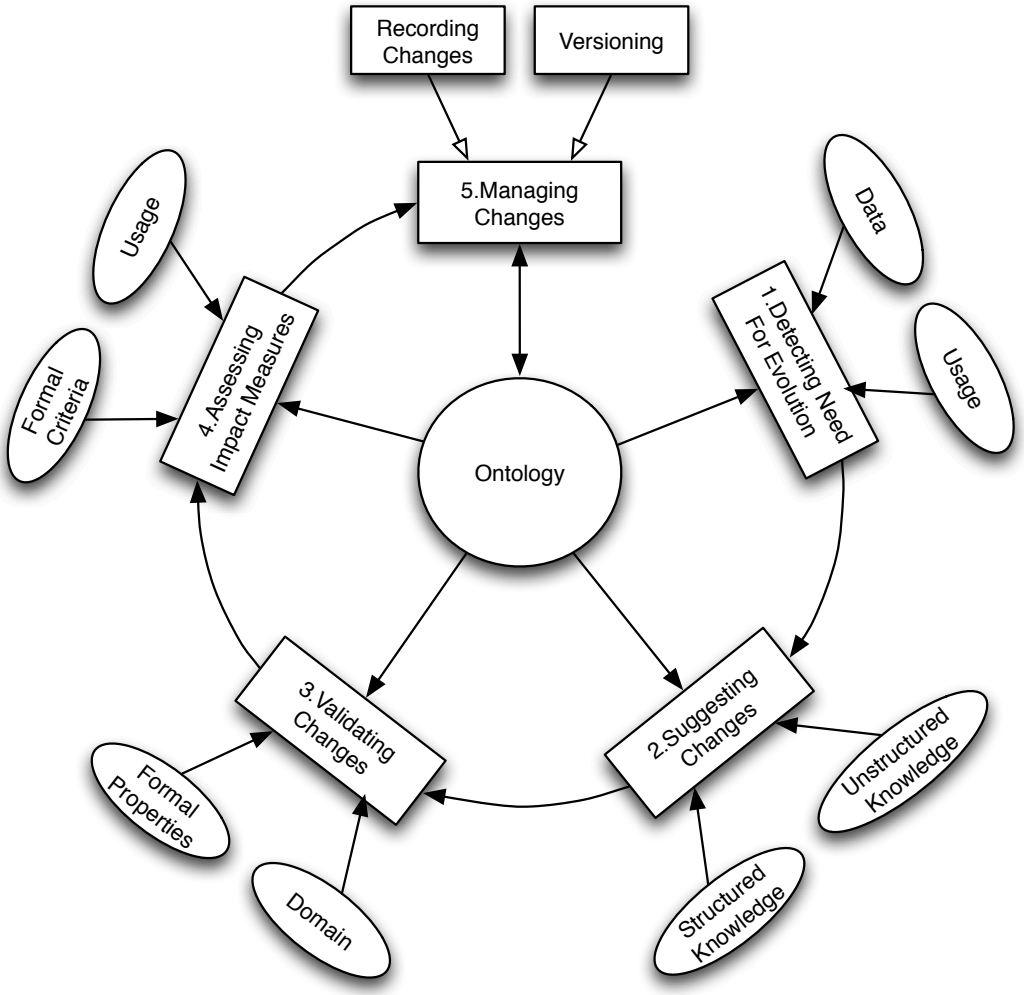


Figure 2.11 – Abstract systematization of Ontology Evolution Cycle [Zablith et al., 2013]

3. The Validating Changes stage filters the changes that lead the ontology to an inconsistent state or don't satisfy domain or application-specific constraints. This is equivalent to the Semantics of change and Implementation steps of Stojanovic et al. [Stojanovic et al., 2002] ontology evolution process.

4. The Assessing Impact stage measures the impact on external artefacts that are dependent on the ontology. The impact measure is based on the cost involved in adding a suggested change to the ontology, or the effect of such a change at the application level. This stage is similar to the Propagation step in Stojanovic et al. [Stojanovic et al., 2002] ontology evolution process.

5. The Managing Changes stage applies and records changes and keeps track of the various versions of the ontology. This stage is divided into two sub-stages: (a) Recording the changes realised on the ontology and (b) Versioning the ontology.

Current work done by Liu et al. [Liu et al., 2014] focuses on the phase of change representation; more specifically, modeling elementary changes via a novel mathematical

calculus named SetPi that is an extension of the mature mathematical tool called Pi-Calculus [Milner, 1999].

Another problem related to Ontology evolution occurs when migrating ontologies to their last versions [Tzitzikas et al., 2013]. It can generate gaps that must be re-examined and others where no re-examination is justified. In [Tzitzikas et al., 2013] some principles, techniques and tools to manage uncertainties in ontology evolution are proposed, specifically for:

— *Automatically identifying descriptions that are candidates for specialization*,

— *Recommending possible specializations*,

— *Updating the available descriptions (and their candidate specializations), after the user (curator of the repository) accepts/rejects such recommendations*.

2.2.3/ Ontology Reasoning

Reasoning is used during ontology development or maintenance time as well as at the time ontologies are used to solve application problems [Moller et al., 2009]. Core reasoning services of Description Logic reasoners include [Sirin et al., 2007]:

— Consistency checking to ensure that the ontology does not contain any contradictory or inconsistent facts.

— Concept satisfiability to check if it is possible for a class to have any instances. If class is unsatisfiable, then defining an instance of the class will cause the whole ontology to be inconsistent;

— Classification to compute the subclass relations to create the complete class hierarchy.

— Realization to find the most specific classes that an individual belongs to.

Description Logics (DL) is a subset of First-Order predicate logics that combines expressivity with a well-understood logical framework. First Order logic (FOL) [Smullyan, 1995] is a formal logical system which includes a domain of discourse over which the quantifiers range. There are many subsystems for first-order logic [Ben-Ari, 2012b], including Hilbert Calculus, Deduction Systems, Gentzen Systems, Tableau-based Algorithms and Resolution-based Algorithms.

A calculus can be classified according to several properties, two of them are [Ben-Ari, 2012b]:

— Soundness - All theorems that can be proved in the logical system are true.

— Completeness - All true statements can be proved in the logical system.

From the First Order Logic algorithms, the most used to reason over ontologies are the Tableau and Resolution algorithms.

Tableau procedures aim to construct a model that satisfies all axioms of the given knowledge-base [Rudolph, 2011]. Tableau algorithms are usually used to test concept satisfiability (consistency). A tableau algorithm contains the following main elements [Bao et al., 2011]:

— A tableau that represents a model of the DL language, typically a “tree model”;

— Tableau expansion rules to construct and complete a consistent completion graph;

— Blocking rules to detect infinite cyclic models and ensure termination;

— Clash conditions to detect logic contradictions.

The tableau base expansion rules are [Baader et al., 2001]: disjunction ($\sqcap$), conjunction ($\sqcup$), value restriction ($\forall$) and existential restriction ($\exists$). Tableau-based methods are successfully implemented in ontology reasoning systems such as:

— Pellet [Sirin et al., 2007], a free open-source Java-based reasoner for SROIQ description logic. It provides a tableau-based decision procedure for TBox and ABox reasoning;

— FaCT++ [Tsarkov et al., 2006] is a free open-source reasoner implemented in C++ with SHOIQ description logic expressivity. It provides a tableau-based decision procedure for general TBox and partial ABox reasoning.

— RacerPro is a commercial tableau-based reasoner for $\mathcal{SHIQ}$ description logic supporting TBox and ABox reasoning.

Resolution is a sound and complete algorithm for propositional logic [Ben-Ari, 2012a] and a very popular reasoning method for First Order Logic [Bachmair et al., 2001]. At the core of reasoning via resolution is the resolution rule [Rudolph, 2011]. Because DL is a subset of First Order Logics, a resolution method can be applied to DL ontologies. In Resolution-based reasoning for DL, the DL ontology is transformed into a set of first-order clauses. While tableau algorithms need sophisticated blocking techniques to ensure termination, resolution-based algorithms automatically terminate as a side effect of the resolution calculus. One implementation of a Resolution-based algorithm is the KAON2 reasoner [Motik et al., 2006], a free Java-based reasoner with *SHIQ* DL expressivity.

Another example is the algorithm used in Hermit [Shearer et al., 2009]. It is a hybrid approach of the resolution and tableau approaches. The base of the Hermit reasoning algorithm is a Hyper-resolution [Robinson, 1965] and Hyper-Tableau [Baumgartner et al., 1996] calculus extended with a blocking condition to ensure termination.

2.2.4/ Large and Web-scale Reasoning

In the Semantic Web it is possible to associate semantic annotations to resources. To handle the large amount of semantic web data, high-performance reasoning methods

are required. This subsection describes some reasoning approaches that contribute to solving the large and web-scale reasoning problem [Urbani, 2013]. The first approaches described parallelize and distribute tableau and resolution-based reasoning approaches, and the last subsection describes approaches that use rules to handle reasoning.

Deslog is a parallel tableau-based description logic reasoner for *ALC*. A first empirical evaluation to classify TBox is presented in [Wu et al., 2012].

Bao et al. [Bao et al., 2011] propose the use of a Tableau-based reasoning algorithm for the description logic *ALC*. The proposed approach takes advantage of several properties of an *ALC* tableau that can be used in a parallel implementation as the union property. They propose algorithm realization using the MapReduce framework by representing the tableau as key-value pairs where tableau nodes are used as keys. The authors Schlicht and Stuckenschmidt [Schlicht et al., 2008] also propose a distributed, complete and terminating algorithm that decides satisfiability of terminologies in *ALC*. The algorithm is based on recent results [Schlicht et al., 2008] on applying resolution to description logics.

On the other hand, rule-based reasoning consists in exhaustively applying a set of rules to a set of triples in order to infer conclusions [Urbani, 2010]. These rules can be applied in a forward way referred to as forward-chaining (or materialization) or in a backward way referred to as backward-chaining. Some authors adopted MapReduce to WebScale reason upon OWL Horst (pD*) ontologies [Mutharaju et al., 2010] [Urbani et al., 2012] or $EL+$ ontologies [Liu et al., 2012a]. These algorithms transform ontology reasoning rules into Map Reduce instructions in order to be processed in a parallel and distributed environment such as Hadoop.

In forward-chaining (or also called materialization reasoning), rules are applied over the entire Knowledge Base until all possible triples are derived [Urbani et al., 2011]. After the closure has been calculated, the query-answer is very simple and efficient. On the other hand, the main disadvantage of these methods is that the closure needs to be updated at every change in the KB, leading to system overload in large and frequently updated knowledge-bases. Adressing the necessity to solve the web-scale reasoning problem, Urbani's [Urbani, 2013] contribution starts by proposing a rule-based forward reasoning technique for materialising the closure of an RDF graph based on MapReduce [Urbani et al., 2009]. In [Urbani, 2010], the author extends the proposed approach to the use of MapReduce to reason over RDFS and over OWL Horst. The authors research culminates in the WebPIE inference engine [Urbani et al., 2010] that is developed on Hadoop. WebPIE outperforms all other published approaches in an inference test over 100 bilion triples [Urbani et al., 2012]. In [Urbani, 2013], the author extracted, by experience, three important principles about web-scale Reasoning:

— ***Treat schema triples differently*:** In this principle, authors state that reasoning methods should make a distinction between schema and other triples when designing reasoning algorithms.

— ***Data skew dominates the data distribution*:** Huge volumes of data can't be replicated in all machines across the cluster and must be distributed across the cluster. Current Web data is highly skewed which makes it hard to choose a balanced partition criterion. The impact of data skewness is important for the reasoned overall performance and should not be underestimated.

— ***Certain problems only appear at a very large scale*:** In web-scale reasoning this is not only a theoretical research question but also an important engineering

problem since they might radically change the evaluation and therefore mislead the judgement over the quality of the proposed method.

The reasons for choosing OWL Horst fragment in WebPIE are [Urbani et al., 2012]:

— *It is a de facto standard for scalable OWL reasoning, implemented by industrial strength triple stores such as OWLIM*;

— *It can be expressed by a set of rules*;

— *It strikes a balance between the computationally infeasible OWL Full and the limited expressivity of RDFS.*

The future work upon the WebPIE [Urbani et al., 2012] reasoner lies in reasoning over user-supplied rule sets, where the system would choose the correct implementation for each rule and the most efficient execution order, depending on the input. Leading to fuzzy scalable reasoning, [Liu et al., 2012a] propose a reasoning approach comparable to WebPIE but capable of leading with pD^* fuzzy semantics. The prototype system is called FuzzyPD and is based on the Hadoop implementation of map-reduce. As for future work, they propose to consider other fuzzing semantics.

Also, a MapReduce algorithm for reasoning over ontologies with EL^+ expressivity [Baader et al., 2006] is proposed by the authors in [Mutharaju et al., 2010]. EL^+ is a fragment of the OWL 2 EL profile. For future work, authors propose the investigating of usage of MapReduce for reasoning over more expressive languages such as EL^{++} or even OWL 2 DL.

Backward-chaining reasoning is focused on query answering because the rules are applied only over the strictly necessary data that leads to the derivation of ground triples of the input query. Updating the knowledge base does not require large resources because the reasoning is only performed for each query. On the other hand, the system has to perform specific computations for every query. The purpose of a Backward-chaining algorithm is to pick from a given database J and from a Datalog program P all possible ground atoms $R(a) \in P(J)$ that are answers to a given atom Q [Urbani et al., 2013]. In [Urbani et al., 2011] WebPIE's author presents an algorithm to perform efficient backward-chaining reasoning on large datasets called QueryPIE (Query Parallel Inference Engine). This algorithm was initially made for the OWL Horst ($pD*$) fragment datasets but recent work shows that it is possible to do efficient backward-chaining over large expressive triple stores [Urbani et al., 2013] using OWL RL expressivity. In [Urbani et al., 2013], the prototype performs the inference by pre-materializing part of the inferences upfront instead of during query time, increasing the performance in OWL RL expressivity.

In [Wu et al., 2013], authors describe a kind of semantic web rule execution mechanism using the MapReduce programming model, which not only can handle RDFS and OWL Horst semantic rules, but also can also be used in SWRL reasoning.

2.3/ CLASSIFICATION

Classification is a type of knowledge discovery. Knowledge discovery in databases refers to the overall process of discovering useful (valuable) knowledge from data. Data mining is a particular step in the knowledge discovery process. Data mining is the application of specific algorithms for extracting patterns from data [Fayyad et al., 1996]. Terminology of the Data Mining research filed is synthetized in the following Fig. 2.12 [Maimon et al., 2010].

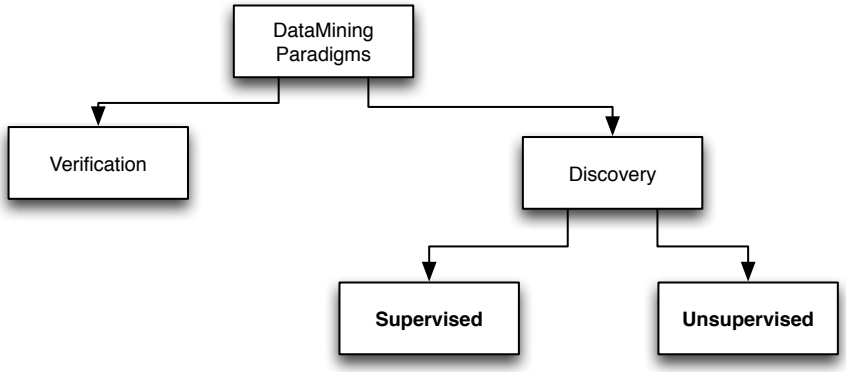


Figure 2.12 – Data Mining terminology

This terminology separates the Data Mining paradigms into two Data Mining main Types [Maimon et al., 2010, binti Oseman et al., 2010, Mashat et al., 2012]:

— Discovery: the system finds new rules and patterns autonomously.

— Verification: the system verifies the user's hypothesis.

A common terminology adopted in the Machine Learning literature divides Data Mining Discovery methods into Supervised and Unsupervised learning methods [Maimon et al., 2010, Kotsiantis, 2007, Amandeep Kaur Mann, 2013, Ricci et al., 2010]:

— Supervised learning methods are methods that attempt to discover the relationship(s) between input attributes (sometimes called independent variables) and target attributes (sometimes called dependent variables) [Ricci et al., 2010]. The relationship found is represented in a structure referred to as a model [Rokach, 2007].

— Unsupervised learning methods are methods that can group instances without a pre-specified, dependent attribute [Ricci et al., 2010].

Another terminology in literature describes Data Mining Discovery methods as Prediction-oriented and Description-oriented [Rokach, 2007, Maimon et al., 2010]:

— **Prediction-oriented methods:** are considered supervised methods.

— **Description-oriented methods:** focus on understanding how the underlying data operates. These methods include the unsupervised learning methods (Fig. 2.12).

The goal of the Supervised Machine Learning Methods is to build a concise model of the distribution of labels used to describe data items by discovering the relationship between the input attributes (features) and the target attribute (labels) [Maimon et al., 2010, Kotsiantis, 2007] based on the training instances (training model). The resulting classifier is used to assign labels to the testing instances where input values are known but the labels for each test instance are unknown. In Supervised Machine Learning Methods, the training dataset used to build the model must include the inputs (features) and the desired results (labels) [Amandeep Kaur Mann, 2013]. Unsupervised learning methods try to suitably organise the elements in classes based on statistical properties without any training instances [Ricci et al., 2010, Amandeep Kaur Mann, 2013]. Clustering is one of the most used unsupervised methods in literature. It automatically arranges data items in groups (clusters) with similar characteristics. Clustering algorithms can be categorized into [Amandeep Kaur Mann, 2013]: partition-based algorithms, hierarchical-based algorithms, density-based algorithms and grid-based algorithms. On the other hand, in semi-supervised learning, features are learned from an unlabelled dataset, which are then employed to improve performance in a supervised setting with labelled data.

Classification is a Machine Learning Method that can predict different classes according to some constrains, and create a model to classify newly available data [Kumar et al., 2012, AL-Nabi et al., 2013]. Classification can be divided into four main types [Tsoumakas et al., 2007, Tsoumakas et al., 2008, Maimon et al., 2010, Cherman et al., 2011] :

— Single Label Classification classifies an item based on one label $|Y| = 0, 1$ from a set of disjointed labels $|L| = 1$;

— Binary Classification is based on classifiers that classify items with only one $|Y| = 1$ of two mutual exclusive labels $|L| = 2$, (e.g. Yes or No);

— Multi-Class is the problem of classifying items into only one label $|Y| = 0, 1$ from a set of available labels $|L| > 2$.

— Multi-Label classifies a item with a set of labels $|Y| >= 0$ from a set of labels $|L| > 1$[Tsoumakas et al., 2007].

Fig. 2.13 depicts the four main types of classification and some examples of each type.

From the following subsections, the first one describes the Multi-Label Classification approach. The second subsection describes the combination of Multi-label and Hierarchical classifications, called Hierarchical Multi-Label Classification (HMC). The third subsection describes Adaptive Learning in Classification context. The fourth subsection describes some related work that uses ontologies in the classification process. The last subsection describes some of the most used methods to evaluate the quality of multi-label classifiers.

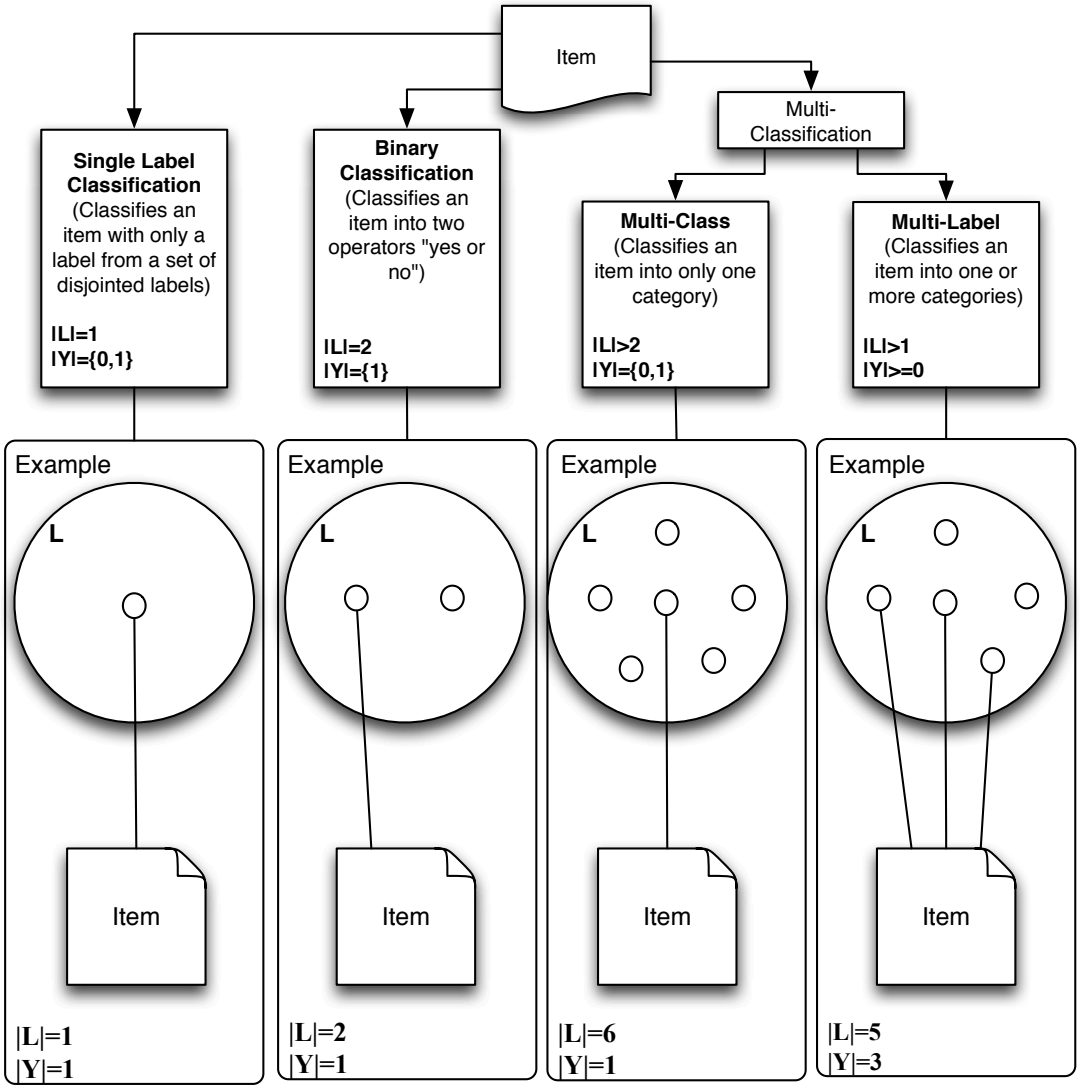


Figure 2.13 – Classification types

2.3.1/ Multi-Label Classification

Multi-label Classification (MLC) methods classify an item with one or more labels from a subset of labels Y where $Y \subseteq L$ and L is the complete set of labels. Textual data, such as documents and web pages, are frequently annotated with more than a single label. MLC is concerned with learning a model that outputs a bipartition of the set of labels as relevant or irrelevant respecting a query instance; In the Multi-Label learning domain, beyond MLC, two more major approaches exist [Maimon et al., 2010]:

— Label Ranking (LR) is concerned with learning a model that outputs an ordering of the class labels according to their relevance to a query instance;

— Multi-label Ranking (MLR) is the combination of MLC and LR, which enables both, an ordering and a bipartition of the set of labels.

Multi-Label learning (MLC, LR and MLR) can be classified in two groups [Tsoumakas et al., 2007, Santos et al., 2009, Maimon et al., 2010, Cherman et al., 2011]: (1) Problem transformation methods and (2) Algorithm adaptation methods. Problem Transformation methods are algorithm-independent because they transform the learning task into one or more single-label classification tasks. There are many problem transformation methods and among them the most popular are:

— Label Power Set, which maps the multi-label problem into one single-label problem [Tsoumakas et al., 2007, Tsoumakas et al., 2010];

— Binary Relevance (BR), which maps the multi-label problem into several single-label problems [Tsoumakas et al., 2007, Tsoumakas et al., 2010, Cherman et al., 2011].

Table 2.1 presents a Multi-label example composed by four labels and four items.

Table 2.1 – Multi-label Problem Transformation example

Item	Label1	Label2	Label3	Label4
Item1	x		x	
Item2				x
Item3		x	x	
Item4	x			x

Applying the Label Power Set method to this example, Labels 1 and 3, 2 and 3, 1 and 4 are combined into only one problem as presented in Table 2.2. In the Item2 case, it only has Label4. Having only one label (Label4), that label appears alone in the Label Power Set result table.

Table 2.2 – Label Power Set Example result

Item	Label1&3	Label4	Label2&3	Label1&4
Item1	x			
Item2		x		
Item3			x	
Item4				x

Applying the Binary Relevance method to the example of Table 2.1, a binary problem is obtained for each label as presented in Table 2.3.

On the other hand, algorithm adaptation methods extend specific learning algorithms in order to handle multi-label data directly. Among these algorithm adaptation methods are the C4.5, the AdaBoost.MH and AdaBoost.MR, the Multi-class Multi-label Perceptron (MMP) and many others [Tsoumakas et al., 2007, Maimon et al., 2010].

Some of the Multi-label Classification algorithms most used in the literature are:

— Classifier chaining (CC) method [Read et al., 2011] uses binary classifiers as in the BR approach. A number of Q different classifiers are linked along a chain, where

Table 2.3 – Label Power Set Example result

Item	Label1	¬Label1
Item1	x	
Item2		x
Item3		x
Item4	x	

Item	Label2	¬Label2
Item1		x
Item2		x
Item3	x	
Item4		x

Item	Label3	¬Label3
Item1	x	
Item2		x
Item3	x	
Item4		x

Item	Label4	¬Label4
Item1		x
Item2	x	
Item3		x
Item4	x	

the $i - th$ classifier deals with the binary relevance problem associated with label $\lambda_i \in L$ where $(1 \leq i \leq Q)$. The feature space of each link between classifiers in the chain is extended with the 0/1 label associations of all previous links. Prediction and ranking of the relevant labels is determined in the same way as in the binary relevance method.

— TNBCC and Path-BCC are two approaches derived from Chain Classifiers [Sucar et al., 2014]. For each label L_i in the chain, a naïve Bayes classifier is defined. In Path-BCC, all the subsuming nodes in the chain from the root label to the label L_i are considered as attributes of the label L_i, while TNBCC only includes the parent label. Hence, Path-BCC uses more contextual information.

— Hierarchy Of Multi-label Classifiers (HOMER) [Tsoumakas et al., 2008] is an algorithm for multi-label learning, designed for large multi-label datasets. HOMER constructs a hierarchy of multi-label classifiers, where each classifier is dedicated to a small set of labels. This allows the process to scale by evenly distributing the items among the classifiers.

— Multi-label k-nearest neighbors (ML-kNN) [Zhang et al., 2007] is an extension of the k-nearest neighbours (kNN) algorithm. First, for each $item_i$, its k-nearest neighbours in the training set are identified, based on its features. Then, the maximum a posteriori principle is used to determine the label set for the $item_i$, based on statistical information extracted from the label sets of its neighbours, i.e., the number of neighbouring examples belonging to each possible label.

— Random Forest of Predictive Clustering Trees (RF-PCT) and Random forest of ML-C4.5 (RFML-C4.5) [Kocev et al., 2007] are ensemble methods that respectively use PCTs and ML-C4.5 trees as base classifiers. Several base classifiers are used to determine the relevance of labels. The predictions made by the classifiers are then combined using voting schemes such as the majority or probability distribution vote.

— Latent Dirichlet Allocation (LDA) is a generative probabilistic model for discovering the underlying structure of discrete data. The Collapsed Gibbs Sampling (CGS) al-

gorithm [Griffiths et al., 2004] approximates the LDA parameters of interest through iterative sampling in a Markov Chain Monte Carlo (MCMC) procedure. Unlike CGS, the final predictions made in the CGS_p[Papanikolaou et al., 2015] approach are based on a Collapsed Variational Bayesian inference update procedure, i.e. CVB0 [Asuncion et al., 2009].

2.3.2/ HIERARCHICAL MULTI-LABEL CLASSIFICATION

In Multi-label Classification, labels are simply treated separately but in many real-world classification problems, one or more class labels can be divided in subclasses or grouped in super-classes [Tsoumakas et al., 2010, Bi et al., 2011, Cerri et al., 2014]. If class labels do not have any structure they are called flat labels. The organization in classes and super-classes forms a hierarchical structure that can usually be two things [Cerri et al., 2014]: (1) Tree-structured hierarchies and (2) Directed Acyclic Graph (DAG) in which a class can have multiple parents. An item is only associated with a certain label if it is associated with the parent labels in hierarchy. Fig. 2.14 depicts the organization of classification classes as described before and some examples of each one.

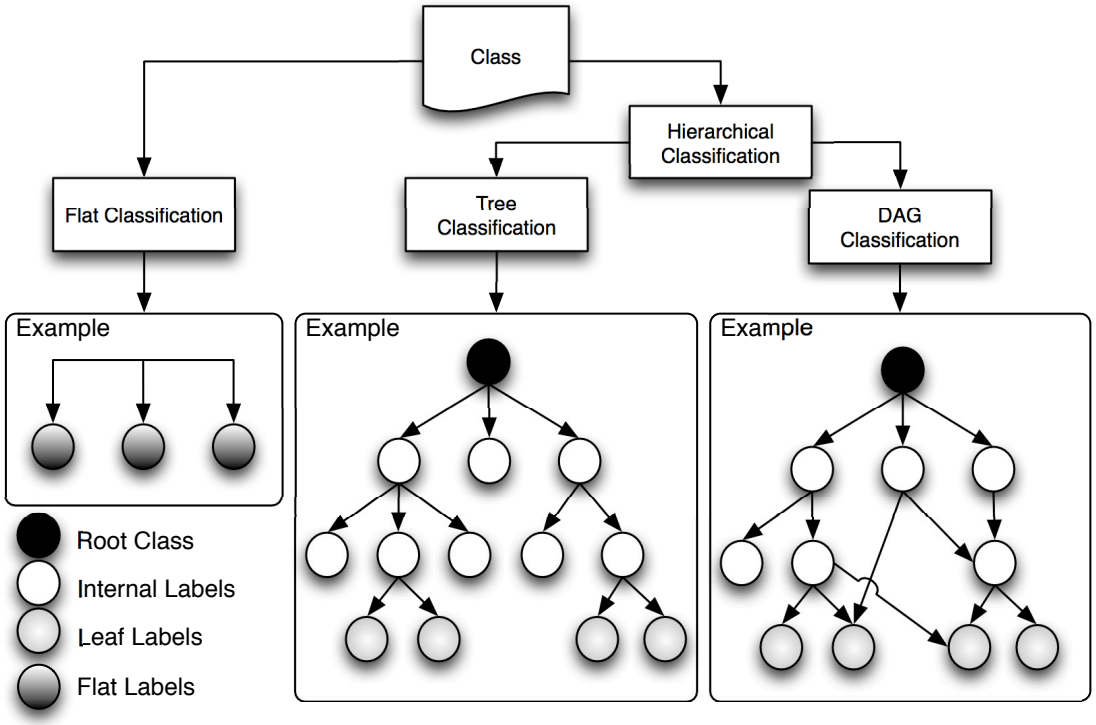


Figure 2.14 – Flat and Hierarchical Classification

Hierarchical Multi-label Classification (HMC) is the combination of Multi-Label classification and Hierarchical Classification [Santos et al., 2009, Bi et al., 2011, Cerri et al., 2014]. Thus, in HMC, items can be assigned to different hierarchical paths and may simultaneously belong to different class labels in the same hierarchical level. Fig. 2.15 depicts an HMC example with class labels organized as Tree and a DAG respectively.

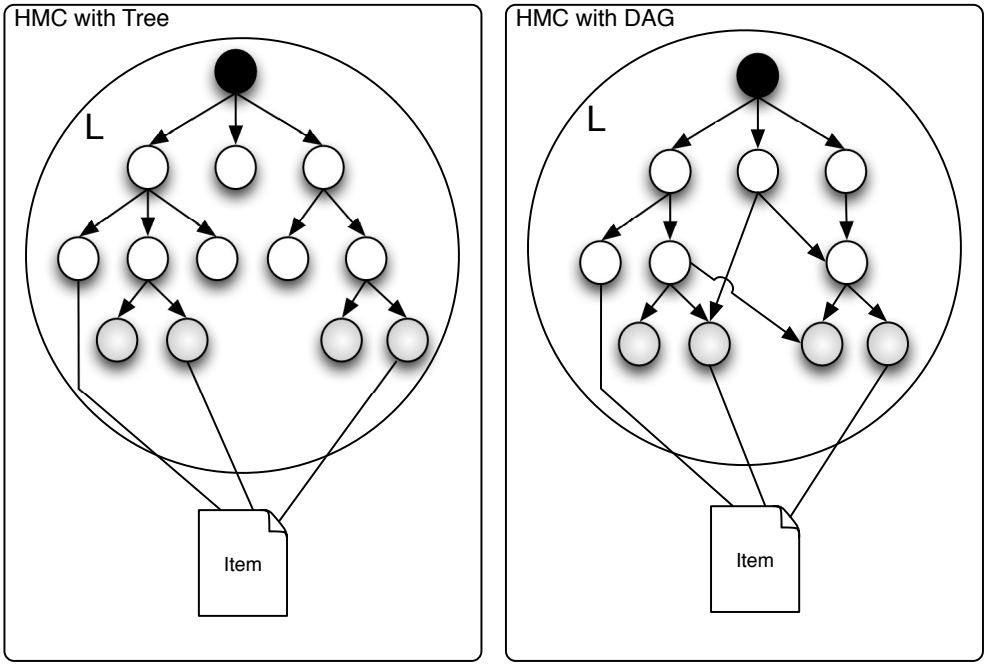


Figure 2.15 – HMC with Tree and DAG Hierarchical structures

Authors in [Santos et al., 2009] and in [Cerri et al., 2014] refer two main approaches to solve HMC problems:

— A local hierarchical approach where a top-down hierarchy of algorithms to classify unlabelled items is produced. The information used during the training process of each classifier is only the local information about the class label hierarchy (direct nodes). In local hierarchical approach, conventional algorithms such as one Local Classifier per Node (LCN), one Local Classifier per Parent Node (LCPN), and one Local Classifier per Level (LCL) [Cerri et al., 2014] can be directly used.

— A global hierarchical approach that uses all the class label hierarchy at once while using only one classifier to discriminate all these classes [Santos et al., 2009, Cerri et al., 2014]. In global hierarchical, one approach can't directly use conventional algorithms unless these are adapted to consider the label class hierarchy.

2.3.3/ Adaptive Learning

The data stream classification problem has been widely studied in the literature [Hulten et al., 2001, Wang et al., 2003, Katakis et al., 2006, Wenerstrom et al., 2006, Wang, 2006, Gao et al., 2007, Spinosa et al., 2008, Masud et al., 2010]. The underlying process generating a data stream can be characterized as stationary or nonstationary [Gama et al., 2014]. In nonstationary, the probabilistic data properties change over time. Theses changes in data distribution can induce changes in the target concept, which is generally known in literature as concept-drift [Widmer et al., 1996]. In nonstationary data distribution, the performance of a non-adaptive classification model trained under the false stationary assumption may degrade classification performance [Tsymbal, 2004].

Two of the most challenging and well-studied characteristics of data streams are its infinite-length and concept-drift [Gama et al., 2014]. Most of the existing data stream classification techniques are designed to handle these two aspects of the classification process [Hulten et al., 2001, Wang et al., 2003, Katakis et al., 2006, Wenerstrom et al., 2006, Wang, 2006, Gao et al., 2007, Spinosa et al., 2008, Masud et al., 2010]. The most used approach to handle these two problems is incremental learning, classified as either:

— Single-model incremental approach, where the classification model is adapted regarding new data [Hulten et al., 2001, Wang, 2006].

— Hybrid batch-incremental approach, in which the model is re-built using a batch learning technique and older models are replaced by newer models [Wang et al., 2003, Gao et al., 2007]

In addition to infinite-length and concept-drift characteristics, some have also studied concept-evolution and feature-evolution [Masud et al., 2010, Spinosa et al., 2008, Katakis et al., 2006, Wenerstrom et al., 2006]. Concept-evolution occurs when new classes (labels in Semantic HMC) used to classify the items emerge in the underlying stream of text items [Masud et al., 2010], [Spinosa et al., 2008]. For example, Spinosa et al. [Spinosa et al., 2008] applied a cluster-based technique to detect novel classes in data streams. The approach builds a normal model that is a static model representing the initial model and remaining as a reference to the initial learning process. The normal model is composed of hyperspheres and is continuously updated with stream progression. If any cluster that satisfies a certain density constraint is formed outside this hypersphere, a novel class is declared. However, this approach focuses on only one class and it is not directly applicable to a multiclass data stream classification.

In the literature, the feature space is a vocabulary, usually composed of a high number of words, used to describe data items. In data stream, new words that can better describe the items must be introduced in the feature space. Feature-evolution occurs when new words (features) emerge from the data stream and old features fade away [Masud et al., 2010], [Katakis et al., 2006], [Wenerstrom et al., 2006]. Katakis et al. [Katakis et al., 2006] propose a feature-evolution technique for text streaming where a dynamic feature space is adapted by an incremental feature selection. In the proposed technique, when a new text document arrives, it is checked whether there are any new words in the document. If there is a new word, it is added to the vocabulary and then the statistics are updated. Experimental results showed that the proposed approach offers better predictive accuracy compared to classical incremental learning.

2.3.4/ Ontologies in Classification

Ontology allows the definition of terms and meanings used to represent areas of knowledge. Two typical scenarios have been identified in the use of ontologies in classification[Fang et al., 2010, Elberrichi et al., 2012, Aparicio et al., 2013, Costa et al., 2013, Galinina et al., 2013, Ben-David et al., 2010, Werner et al., 2014]:

— Capturing and representing classification rules and domain knowledge.

— Ontology reasoning. Beyond capturing and representing knowledge, inference engines use ontologies in order to help the classification process.

Some approaches from the literature that use ontologies to represent the domain knowledge for classification are:

— In [Galinina et al., 2013] authors propose two ontologies to represent a classification system: (1) Domain ontology that is independent of any classification method; (2) Method ontology devoted to decision tree classification. Authors present the complete construction of the Method Ontology but no reference or description about Domain ontology has been found in the paper.

— In [Aparicio et al., 2013] the authors propose the use of ontologies in order to assist the semi-supervised classification. Ontologies are used to provide the expected words to find in documents of a particular class. By using this information, they guide the direction of the use of unlabelled data respecting the particular method rules.

— In [Costa et al., 2013] authors introduce a new conceptual framework for representation of knowledge sources (whether they are web pages or documents), where each knowledge source is semantically represented (within its domain of use) by a Semantic Vector (SV), which is enriched by using the classical Vector Space Model approach extended with ontological support, employing ontology concepts and their relations in the enrichment process.

— In [Elberrichi et al., 2012] authors present a method for improving the classification of medical documents using domain ontologies (MeSH - Medical Subject Headings). Two steps compose the method:

  1. First, mapping the terms into concepts, having chosen a matching and disambiguation strategy for an initial enrichment of the representation vector. The words are mapped into their related concepts using the ontology. For example, the words "appendicitis" and "appendiceal" are mapped in the "appendicitis" concept and the term frequencies of these two words are added in the concept frequency.

  2. Then, there is a second step that enriches the ontology by adding hyperonyms to the representation vector.

— In [Johnson et al., 2010] the authors propose an iterative and interactive (between AI methods and domain experts) approach to achieve prediction and description (“which are usually hard to fulfill”), considering “domain expert knowledge and feedback”. The authors do not aim to automatically multi-classify the items but only to improve the ontology, which means that the resulting ontology is not used for an automatic classification of items.

— In [Vogrincic et al., 2011] the authors are concerned with automatically creating an ontology from the text documents without any prior knowledge about their content.

Some approaches from the literature that use ontologies to represent not only the domain knowledge but also the inference capacities of web reasoners in order to help in the classification process are:

— In [Fang et al., 2010] authors present a document classification method that uses ontology reasoning and similarity measures. Four steps compose the classification method:

1. Documents and Categories: A unique identifier and a set of weighted terms represent the documents and categories are represented using an ontology.
2. Ontology reasoning is used to compute all the more specific concepts in the ontology.
3. Classification: A Boolean value T,F is assigned to each pair of <document, category> in order to represent the document aptitude in the category.
4. Calculation of Semantic Similarity: Normalized Google Distance [Cilibrasi et al., 2007] is used to calculate similarity scores between the ontology and documents.

— In [Ben-David et al., 2010] authors introduce a generic, automatic classification method that uses Semantic Web technologies to: define classification requirements, perform the classification and represent the results. This method allows data elements from diverse sources and of different formats and types to be classified using a universal classification scheme. The proposed generic classifier is based on an ontology, which gives a description of the entities that need to be discovered, the classes mapping these entities, and information on how they can be discovered. Classification is performed using data mining techniques and the inference is used only to improve the output results.

— In [Werner et al., 2014] a method proposes to semantically enrich the ontology used to describe and process the classification. This ontology aims to reduce the gap between the experts' perspective and the representation of classification rules. To enrich the ontology and classify the documents, the classification process uses a DL out-of-the-box Ontology Reasoner like Pellet [Sirin et al., 2007], FaCT++ [Tsarkov et al., 2006], or HermiT [Shearer et al., 2009]. Most of these reasoners are sound and complete to high expressivity such as the OWL2 SROIQ (D) expressivity. However, they are not highly scalable.

2.3.5/ Classification Quality Evaluation

Several studies in the literature evaluate the quality of the classification process[Bi et al., 2011, Vens et al., 2008, Cerri et al., 2014]. Precision and recall are two standard measures widely used in text categorization literature to evaluate the generalization performance of the learning system on a given category. For single-label classification problems, precision and recall are defined as:

$$Precision = \frac{TP}{(TP + FP)} \tag{2.1}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{2.2}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives.

However, performance evaluation on multi-label learning algorithms is not trivial as each item is associated with multiple labels simultaneously, whereas traditional single-label criteria such as accuracy, precision or recall cannot be directly applied. Specific evaluation metrics on multi-label learning are proposed in literature and generally categorized into two groups [Zhang et al., 2014] :

— Example-based metrics evaluate the learning system's performance on each test example (labelled item) separately, and then return the mean value across the test set.

— Label-based metrics learn the system's performance on each class label separately, then returning the macro/micro-averaged value across all class labels.

In order to evaluate the classification performance, a label-based metric with macro and macro averaged precision, recall and $F1$ measure. The macro-averaged precision, recall and $F1$ are calculated as:

$$Precision_{macro} = \frac{1}{n}\sum_{i=0}^{n}\left(\frac{TP_i}{TP_i + FP_i}\right) \tag{2.3}$$

$$Recall_{macro} = \frac{1}{n}\sum_{i=0}^{n}\left(\frac{TP_i}{TP_i + FN_i}\right) \tag{2.4}$$

$$F1_{macro} = \frac{2 * (Precision_{macro} * Recall_{macro})}{(Precision_{macro} + Recall_{macro})} \tag{2.5}$$

where TP_i is the number of true positives, FP_i the number of false positives and FN_i the number of false negatives for Label i. The micro-averaged precision, recall and $F1$ are calculated as:

$$Precision_{micro} = \frac{\sum_{i=0}^{n} TP_i}{\sum_{i=0}^{n} TP_i + \sum_{i=0}^{n} FP_i} \tag{2.6}$$

$$Recall_{micro} = \frac{\sum_{i=0}^{n} TP_i}{\sum_{i=0}^{n} TP_i + \sum_{i=0}^{n} FN_i} \tag{2.7}$$

$$F1_{micro} = \frac{2 * (Precision_{micro} * Recall_{micro})}{(Precision_{micro} + Recall_{micro})} \tag{2.8}$$

Typically, multi-label classification systems rely on a threshold to limit how many labels will be predicted for each item [Vens et al., 2008, Bi et al., 2011, Cerri et al., 2014]. Low threshold values lead to a high percentage of items being classified with many labels, resulting in high recall and low precision. On the other hand, larger threshold values lead to less items being classified, resulting in high precision and low recall. To deal with this problem, Precision–Recall curves (PR-Curves) that plot the precision of a classification system as a function of its recall, are currently used in literature [Vens et al., 2008, Bi et al., 2011, Cerri et al., 2014]. PR-Curves can be obtained by

ranging the threshold between [0,1] where each threshold value represents a point in the PR-Space. In order to summarize the classification quality by a single performance score, the Area Under the PR Curve (AUPRC) is currently being used. In this case, the larger the AUPRC, the better the classification quality is. An example of a AUPRC is depicted in Fig. 2.16.

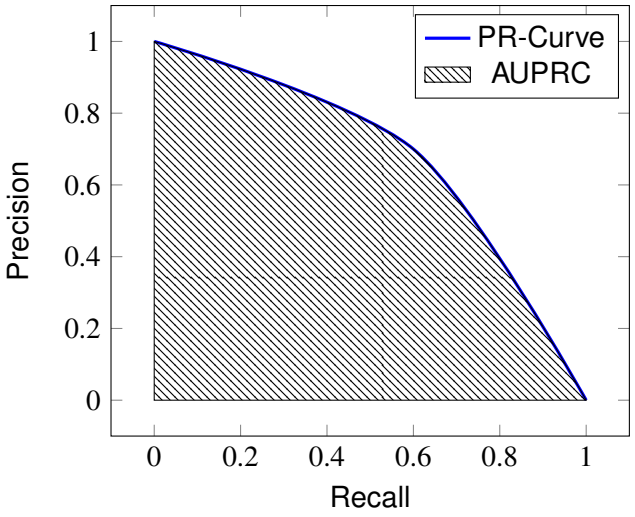


Figure 2.16 – Area Under the PR-Curve (example)

2.4/ SUMMARY

From the results of recent studies, Big Data analytics is still at an early development stage, emphasized by the fact that most of the approaches attempted to apply traditional solutions to the new problems/platforms/ environments.

On one hand, most of the literature in Classification field focuses on describing or improving the classification processes using ontologies, but do not take advantage of the reasoning capabilities of web reasoning to automatically multi-classify the items. In [Werner et al., 2014] authors use out-of-the-box reasoners to classify economy-related documents but their scalability is limited and cannot be used with large datasets as it is required in a Big Data context. However, as Semantic Web is growing new high-performance "Web-scale Reasoning" methods have been proposed [Urbani, 2013]. Rule-based reasoning approaches allow the reasoning implementation in distributed and parallelized technologies such as the ones used to process Big data (e.g. MapReduce) [Dean et al., 2008].

Web-scale Reasoners, [Urbani, 2013] however, instead of using traditional Description logics approaches like Tableau [Sirin et al., 2007, Tsarkov et al., 2006], Resolution [Motik et al., 2006] or Hypertableau [Shearer et al., 2009], use entailment rules for reasoning over ontologies. Web-Scale Reasoners based on MapReduce programming models like WebPIE [Urbani et al., 2010] outperform all other published approaches in an inference test over 100 billion triples [Urbani et al., 2012]. However, recent implementations of web-scale Reasoners like WebPIE are limited to low expressive ontologies such as the OWL-Horst fragment [ter Horst, 2005], due to the complexity of implementation and performance at a web scale. In [Wu et al., 2013] authors describe a kind of seman-

tic web rule execution mechanism using MapReduce which can be used with the more expressive OWL-Horst and with SWRL rules.

On the other hand technologies to analyse Big Data as Apache Mahout or SAMOA are based on highly scalable technologies to process Big Data that distribute the process across several machines. However, neither method uses ontologies to describe the classification model in their algorithms, nor uses Web Reasoning to classify the data items.

To the extent of our knowledge, a classification process to automatically classify text documents in a Big Data context by taking advantage of ontologies and rule-based reasoning to perform the classification is novel. The discussion of the related work for each specific contribution of this thesis is done at the beginning of the specific chapter when needed.

II

Semantic Hierarchichal Multi-label Classification Process

3

SEMANTIC HMC FOR BIG DATA ANALYSIS

Data analysis is the use of proper statistical methods to: *"analyze massive data, to concentrate, extract, and refine useful data hidden in a batch of chaotic datasets, and to identify the inherent law of the subject matter, so as to maximize the value of data"* [Chen et al., 2014]. Big Data analysis can be deemed as the data analysis technique for Big Data. The analysis of Big Data is the final and most important phase in the value chain of Big Data with the purpose of providing either suggestions or decisions [Chen et al., 2014]. Therefore, many traditional data analysis methods such as Data Mining algorithms (Classification, clustering, regression, among others) may still be utilized for Big Data Analysis [Chen et al., 2014].

In order to analyse huge volumes of unstructured data in Big Data context, this chapter proposes a new hierarchical multi-label classification process called Semantic HMC. The process learns the classification model from a static set of items (batch learning). In this case, a static classification model is created and all new documents are classified according to that classification model. The process is based on an unsupervised ontology learning process using scalable machine-learning techniques and rule-based reasoning. Ontologies are used to describe the classification model in which they play a key role defining terms and meanings used to represent knowledge, reducing the gap between the users and the HMC process. The classification model is composed of labels (classes) and classification rules.

In this chapter, the Semantic HMC process is proposed on a high abstraction level and will be detailed in Chapter 4 and Chapter 5.

The remaining of this chapter is structured in two sections. Section 3.1 proposes and overviews the Semantic HMC process to automatically and hierarchically classify data items from large sets of data. Section 3.2 draws conclusions and suggests further research.

3.1/ OVERVIEW OF THE SEMANTIC HMC

The Semantic HMC is based on an unsupervised ontology learning process using scalable Machine-Learning techniques and Rule-based reasoning. The Semantic HMC process is generic for a large Variety of unstructured data items (e.g. text, images) and

scalable for a large Volume of data. The process is unsupervised so that no previously labeled/classified examples or rules exist that relate the data items to the labels.

The classification model is represented by an ontology (Abox+Tbox). To infer the most specific concepts for each data item and all subsuming concepts, rule-based reasoning is used. The rules are applied to a set of triples (items) to infer conclusions [Urbani et al., 2011], i.e. the item classifications. This rule-based reasoning approach allows the parallelization and distribution of work by large clusters of inexpensive machines through Big Data technologies like map-reduce [Dean et al., 2008]. Web-Scale Reasoners [Urbani, 2013] currently use rule-based reasoning to reach high scalability by load parallelization and distribution, thus addressing the Volume and Velocity dimensions of Big Data.

The Semantic HMC is composed of five individually scalable steps (Fig.3.1) to reach the aims of Big Data analytics:

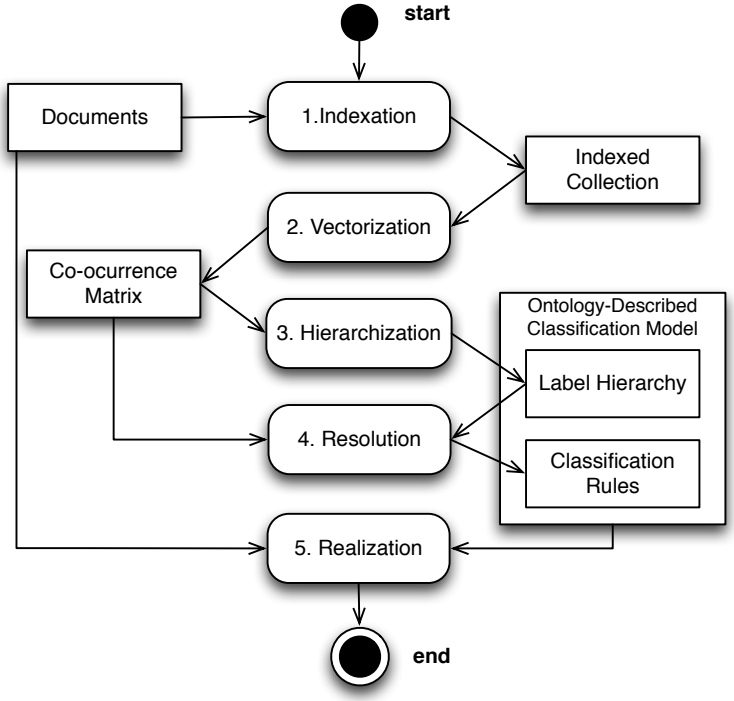


Figure 3.1 – The Semantic HMC process

— **Indexation** extracts terms from data items and creates an index of data items.

— **Vectorization** calculates the term-frequency vectors of the indexed items.

— **Hierarchization** creates the label taxonomy (i.e. subsumption hierarchy) using term-frequency vectors.

— **Resolution** describes taxonomy concepts using relevant terms and creates the reasoning rules to classify data items with labels based on term-frequency vectors.

— **Realization** populates the ontology with items and then determines, for each item, the most specific label and all its subsuming labels.

The rest of this section describes the classification model represented by an ontology and then describes the five steps that compose the Semantic HMC process.

3.1.1/ Ontology-described Classification Model

The Semantic HMC's classification model is captured by a DL ontology with $ALCI$ expressivity, as described in Table 3.1.

Table 3.1 – 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

The *Item* class represents data items to be labeled/classified and is populated with data items (e.g. text document) at the assertional level (Abox).

The *Term* class defines the extracted terms from data items and is populated at the assertion level (Abox) with terms (e.g. words extracted from text documents, symbols representing subjects/objects in photos). The *Label* class defines the terms that are considered to classify the items.

The *Broader* and *Narrower* relations define the subsumption hierarchy between labels. The relation *hasTerm* links the asserted *Items* to the asserted *Terms*. As proposed in [Werner et al., 2014] two types of relations (Alpha and Beta) are used to relate the features (*Term*) and the labels in order to create the classification rules for each *Label*. The relation's *hasAlpha* and *hasBeta* represent the relations of the type Alpha and Beta respectively.

The relation *isClassified* represents the categorization of an *Item*. Predicting relationships between *Item* and *Label* is the final goal of the process.

3.1.2/ Indexation

The indexation step extracts terms from data items and creates an index of data items. In the example of indexing text documents, the extracted terms are relevant words that describe an item. The term "extraction" includes treatments

such as spelling correction and stop-word and synonym detection [Hearst, 1992, Toutanova et al., 2000, Cimiano et al., 2003] as well as calculation of composed terms by collocation [Smadja et al., 1990]. A collocation is a sequence of words (n-grams), which co-occur more often than it would be expected by chance. An association measure algorithm evaluates whether the co-occurrence is purely by chance or statistically significant. The inverted index allows efficient retrieval of documents by term.

3.1.3/ VECTORIZATION

Vectorization calculates the term-frequency vectors of the indexed items by calculating the term frequency of each term in the collection of documents (i.e. Document Frequency and TF-IDF). Furthermore, the term co-occurrence frequency matrix is created to represent the co-occurrence of any pair of terms in a document. The calculated matrix is exploited in the hierarchization step to create subsumption relations and in the Resolution step's case to create the classification rules. The vectors have lately been used in the realization step to describe the data items with relevant terms.

3.1.4/ HIERARCHIZATION

Hierarchization creates the taxonomy label (i.e. subsumption hierarchy) by exploiting the co-occurrence matrix. As it is an unsupervised process where no labels are previously defined, the most relevant terms are designated as labels. In order to calculate the relevancy of a term, a ranking method based on information retrieval is used. Thus, a subsumption algorithm is used to automatically calculate the hierarchical relations between labels. The result of this step is a subsumption hierarchy of labels, and more specifically a Directed Acyclic Graph (DAG).

3.1.5/ RESOLUTION

Resolution describes the taxonomy concepts (labels) using their related terms and creates the classification rules used to classify data items with labels, i.e. it establishes the conditions for an $item1$ to be classified in $label1$. The created classification rules define the necessary and sufficient terms for an item to be classified with a label. The rules are serialized in a horn clause language. The main interest in using horn clause rules instead of translating the rules into logical constraints of an ontology captured in Description Logic is to reduce the reasoning effort, thus improving the scalability and performance of the system.

3.1.6/ REALIZATION

The realization step populates the ontology and performs the multi-label hierarchical classification of the items. To that end, the ontology is first populated with the items and most relevant terms to describe each item at an assertion level (Abox). Then, a rule-based inference engine is used to infer the most specific labels as well as all the broader concepts for each item according to the label hierarchy. This leads to a multi-label classification of the items based on a hierarchical label structure (Hierarchical Multi-label Classification).

3.2/ SUMMARY

This section proposes an ontology-based hierarchical multi-label classification process called Semantic HMC that automatically multi-classifies data items. Five individually and scalable steps (Indexation, Vectorization, Hierarchization, Resolution and Realization) compose the Semantic HMC process to reach the aims of Big Data analytics and then extract Value from Big Data. The next two chapters describe in detail the steps of the proposed Semantic HMC process:

— **Chapter 4:** describes in detail the three first steps of the Semantic HMC process (Indexation, Vectorization and Hierarchization). It automatically learns the hierarchy of labels from huge sets of data. The steps are implemented using technologies for Big Data that distribute the process by several machines in order to reach high performance and scalability.

— **(Chapter 5):** focuses on the two last steps of the Semantic HMC process (Resolution and Realization). It proposes a new process to hierarchically multi-classify items from huge sets of unstructured texts using DL ontologies and Rule-based reasoning. The process is implemented using scalable approaches that distribute the process by several machines in order to reach high performance and scalability required by Big Data. The process is evaluated and compared with some multi-label classification algorithms from the state of the art.

4

Automatic Hierarchy learning for Hierarchical Multi-label Classification

Chapter 3 described the Semantic HMC process that proposes five individually scalable steps to classify data items in the context of Big Data: (i) Indexation, (ii) Vectorization, (iii) Hierarchization, (iv) Resolution and (v) Realization. This chapter focuses on the first three steps of the Semantic HMC process. The process is implemented using technologies for Big Data that distribute the process by several machines in order to reach high performance and scalability, addressing the Volume and Velocity of the source data. The results show that the label hierarchy is automatically learned from large datasets of unstructured text data in a scalable way.

The remaining of this chapter is structured in seven sections. Section 4.1 presents the specific background knowledge and related work in automatic extraction of concept hierarchies from unstructured text to a Hierarchical Multi-Label Classification. Section 4.2 introduces the Indexation step where terms are extracted from data items and an inverted index of data items is created. Section 4.3 describes the Vectorization step where term-frequency vectors and a co-ocurrence frequency matrix are calculated from indexed data items. Section 4.4 systematizes the Hierarchization step where the label hierarchy is created using the co-occurrence frequency matrix. Section 4.5 show the implementation of the proposed process to automatically learn a hierarchy of concepts in a scalable and distributed platform to process Big Data. Section 4.6 discusses the results obtained. Finally, the last section 4.7 summarizes this chapter.

4.1/ Specific Background and Related Work

This section introduces the background knowledge and discusses the current related work in automatic hierarchy extraction from unstructured text corpus for a hierarchical multi-label classification of unstructured texts.

In an automatic taxonomy learning process, the labels, i.e. nodes that compose the taxonomy, and their hierarchical organization form are learned from text analysis. Two main different aspects of label hierarchy extraction from texts can be distinguished: (1) label extraction and (2) hierarchy creation.

Several methods exist in literature for extracting terms from a set of documents. These methods can be categorized into two different approaches: (i) linguistic and (ii) statistical approaches. Linguistic approaches use natural language processing (NLP) for term extraction [Hearst, 1992, Toutanova et al., 2000, Cimiano et al., 2003, Cambria et al., 2014, Sidorov et al., 2014]. A common linguistic approach is the part-of-speech tagging that labels the part-of-speech (e.g. noun, adjective, verb) in a text and often selects the nouns and adjectives as terms [Toutanova et al., 2000]. The statistical approaches use probabilistic techniques to extract the terms from text, such as those described in [Salton et al., 1988, Maedche et al., 2001, Wu et al., 2004, Lomotey et al., 2013]. These methods have a good performance in evaluating the term relevancy but lack the advanced grammatical analysis of linguistic methods that can capture the function and sense of words in a sentence, i.e. semantics.

To automatically organize the extracted labels in a hierarchy, unsupervised learning methods are used in literature [Lance, 1967, Sanderson et al., 1999, De Knijff et al., 2013, Meijer et al., 2014]. They usually try to suitably organize the labels based on statistical properties only [Ricci et al., 2010, Amandeep Kaur Mann, 2013]. Several unsupervised learning methods exist to create hierarchical relations between concepts, including [Lance, 1967, Sanderson et al., 1999, De Knijff et al., 2013, Meijer et al., 2014]:

— Hierarchical clustering methods that create isolated clusters for each concept, and progressively merge clusters that are close, e.g. [Lance, 1967, De Knijff et al., 2013, Meijer et al., 2014].

— Subsumption methods that construct the broader-narrower relations between concepts based on their co-occurrence, e.g. [Sanderson et al., 1999, De Knijff et al., 2013, Meijer et al., 2014].

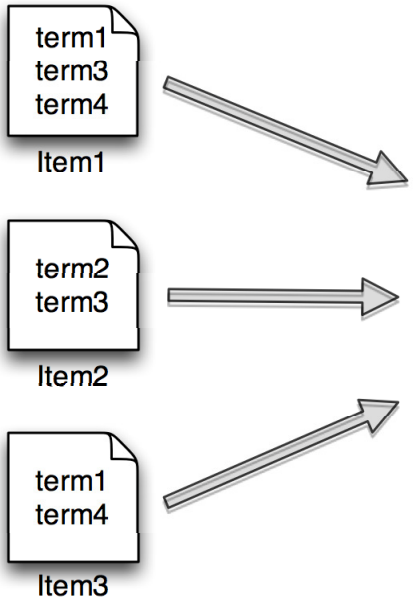
The main advantage of the subsumption method compared to hierarchical clustering is the balance between the processing speed and the ability to provide good broader-narrower relations. The advantages and drawbacks of each method are deeply studied in [De Knijff et al., 2013].

Based on the proven advantages and disadvantages of each approach, this chapter describes the proposal of a process to learn the hierarchy using: (i) a hybrid approach between statistical and lexical approaches to learn the labels and (ii) a highly scalable subsumption method to learn hierarchical relations between labels. This process allows automatically learning a label hierarchy from huge volumes of unstructured data. This label hierarchy is then used to perform Hierarchical Multi-Label Classification (in steps 4 and 5 of the proposed Semantic HMC process).

4.2/ INDEXATION

The indexation step extracts terms from a collection of items and creates an index of items. Each item type has its specific parser to efficiently get useful information for the other steps, reducing the Limited Context Analysis problem. The Limited Content Analysis problem [Lops et al., 2011] [Bobadilla et al., 2013] is defined by the difficulty to extract reliable automated information from various content (e.g. text, images, sound), which can greatly reduce the quality of the classification. By reducing the Limited Content Analysis,

the Semantic HMC's capability to handle more Variety of data is improved as more types of items can be processed. In the example of indexing text documents, the extracted terms are relevant words to describe an item. The result is an inverted index (Fig. 4.1), represented by a set of vectors in the form: $vector_t < i_1, i_{2,} \ldots, i_n >, \forall i \in C$, where t is a term, i_j are the data items where the term t occurs, C a collection of items, and $n \leq |C|$ the number of items in the collection C. The inverted index allows efficient retrieval of items by a giving term.



Inverted Index

Term	Items
term1	(item1, item3)
term2	(item2)
term3	(item1,item2)
term4	(item1,item3)

Figure 4.1 – Example of an Inverted Index

4.3/ VECTORIZATION

In the Vectorization step, the identified terms give rise to two types of vectors [Salton et al., 1988]

— Item-frequency (document frequency), $df_C = \{(t, df_{t,C})\}$ where t is the term, C is the collection of items and $df_{t,C}$ is the number of items from the collection C where the term t is observed.

— Term-frequency in each item, $\mathcal{V}^i_{tfidf} = \{(t, tfidf_{i,t,C})\}$, where t is the term, i is the item, C is the collection of items and $tfidf_{i,t,C}$ is the TF-IDF value [Salton et al., 1988] of term t in item i regarding the collection of items C.

Furthermore, a term co-occurrence frequency matrix cfm is created to represent the co-occurrence of any pair of terms $(term_i, term_j)$ in the collection of items C, such that:

$$cfm\left(term_i, term_j\right) = \left|\left\{item_{i,j} \in C | item_{i,j} \in vector_{term_i} \wedge item_{i,j} \in vector_{term_j}\right\}\right| \tag{4.1}$$

where $item_{i,j}$ is an item, $vector_{term_i}$ is the vector of the inverted index for $term_i$ and $vector_{term_j}$ is the vector of the inverted index for term $term_j$. If n denotes the number of different terms in a collection of C items, the term co-occurrence matrix for the collection C is a $n \times n$ symmetric matrix. The main diagonal of the co-occurrence matrix $cfm(term_i, term_i)$ denotes the occurrence of $term_i$ in all items of C (i.e. the document frequency). Therefore, the main diagonal $cfm(term_i, term_i)$ is the maximum value for the line and column of $term_i$ of the co-occurrence table. Table 4.1 depicts an example of a 3×3 term co-occurrence frequency matrix, where $cfm\,(Term_1, Term_2) = cfm\,(Term_2, Term_1) = 70$

Table 4.1 – Term co-occurrence frequency matrix

	$Term_1$	$Term_2$	$Term_3$
$Term_1$	95	70	80
$Term_2$	70	80	60
$Term_3$	80	60	90

4.4/ HIERARCHIZATION

The hierarchization step learns the labels and their subsumption relations. The most relevant terms are designated as labels. To evaluate term relevance, the information retrieval method described in [Feldman et al., 1998] is used. The method exploits the item-frequency vectors as to calculate, for each $term_j$, the proportion $P_C(term_j)$ of items in a C collection in which the $term_j$ appears:

$$P_C\left(term_j\right) = \frac{df_C(term_j)}{|C|} \tag{4.2}$$

where $df_C\left(term_j\right)$ is the document frequency of the $term_j$ in the collection of items C and $|C|$ is the cardinality of the collection C.

Based on the terms that have a higher proportion $P_C\left(term_j\right)$ than a label threshold (lT) such that $P_C(term_j) \geq lT$ where $term_j \in Term$, the hierarchization process outputs the set of labels ω_{lT}:

$$\omega_{lT} = \left\{term_j \in Term | P_C\left(term_j\right) \geq lT\right\} \tag{4.3}$$

The terms in this set are classified as $Label$ such that $Label \subseteq Term$.

To build the subsumption hierarchy, the subsumption method is adopted because of its proven performance. A scalable method based on [Sanderson et al., 1999] is used, that exploits the co-occurrence matrix. Label x potentially subsumes label y, *i.e.* $x \prec y$ if (Fig. 4.2(A)):

$$(P_C\,(x|y) = 1) \wedge (P_C\,(y|x) \prec 1) \tag{4.4}$$

where:

— $P_C(x|y)$ is the conditional proportion (number) of the items from collection C common to x and y, in respect to the number of items in y such that:

$$P_C\,(x|y) = \frac{x \cap y}{y} = \frac{cfm(x,y)}{df_y} \tag{4.5}$$

— $P_C(y|x)$ is the conditional proportion (number) of the items from collection C common to x and y, in respect to the number of items in x such that :

$$P_C(y|x) = \frac{x \cap y}{x} = \frac{cfm(x,y)}{df_x} \tag{4.6}$$

In [Sanderson et al., 1999] the authors noticed that many subsumed concepts are failing to be included because a few occurrences of the subsumed term y do not co-occur with the term x . By relaxing the first condition $P_C(x|y) = 1$ it is possible to capture these failing concepts (Fig. 4.2 (B)). The subsumption with the first condition relaxed is redefined as:

$$(P_C(x|y) \geq st_1 \wedge (P_C(y|x) < 1) \tag{4.7}$$

where $st_1 \in [0,1]$ is the subsumption co-occurrence threshold for the first condition. In [De Knijff et al., 2013] the authors also propose to relax the second condition $P_C(y|x) < 1$ allowing to define how larger the set of items containing the subsuming term must be in relation to the subsumed term y (Fig. 4.2 (C)). With both conditions relaxed, the subsumption is thus redefined as:

$$(P_C(x|y) \geq st_1) \wedge (P_C(y|x) < st_2) \tag{4.8}$$

where $st_1 \in [0,1]$ is the subsumption co-occurrence threshold for the first condition, $st_2 \in [0,1]$ is the subsumption co-occurrence threshold for the second condition and $st_1 \geq st_2$. If x appears in at least proportion st_1 of the documents in which y also appears, and y appears in less than proportion st_2 in which x appears, then x potentially subsumes y, i.e. $x \prec y$. By applying this method to all labels, the result is a hierarchy of labels, and more specifically a Directed Acyclic Graph (DAG).

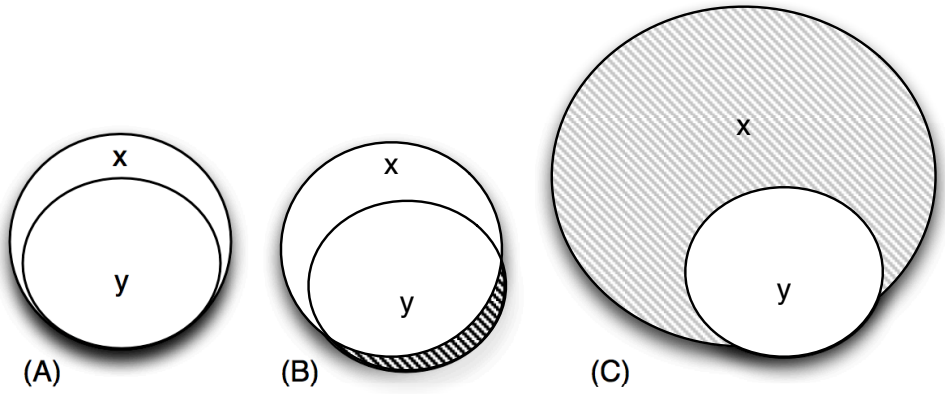


Figure 4.2 – Euler diagrams of the subsumption method

4.5/ IMPLEMENTATION

This section describes the implementation of the first three steps of the proposed Hierarchical Multi-label Classification process described in the previous section. The process is implemented as a Java application, as many Java libraries support the proposed methods.

In the first three steps (indexation, vectorization and hierarchization) of the Semantic HMC process, Big Data processing techniques such as MapReduce are used. Each algorithm described in the previous section is implemented in MapReduce, and deployed in a Hadoop server. The Hadoop implementation of MapReduce was chosen because of its open-source nature and its ease of integration with the tools used in the process. The vectors, the co-occurrence matrix and the hierarchy are stored in HDFS (Hadoop Distributed File System), to be used in the resolution and realization steps.

The next subsections describe the implementation details of each step of the Semantic HMC process.

4.5.1/ INDEXATION

The indexation step extracts terms from a collection of text documents and creates an index of text documents.

In this implementation, a parser and a tokenizer are used to extract terms from text documents. Term extraction includes stop-word and synonym detection, lemmatization, and calculation of composed terms. The relevance of composed terms is calculated by collocation. An association measure algorithm evaluates whether the co-occurrence is statistically significant or not. Statistically based methods are used to identify potentially relevant combinations of words, and the Log-Likelihood [Dunning, 1993] association measure is used to calculate the relevance.

The Apache Solr server is used as parser, tokenizer and indexer. Collocation's computation is performed using Mahout library [The Apache Software Foundation, 2013]. Mahout is a scalable open-source machine-learning library especially addressed to process collections of data that are too large for a single machine. Solr and Mahout are deployed on a Hadoop cluster. Figure 4.3 depicts the logical component diagram for the indexation step (Hadoop not represented) where the indexer specific coordination is described trough a dispatcher[1].

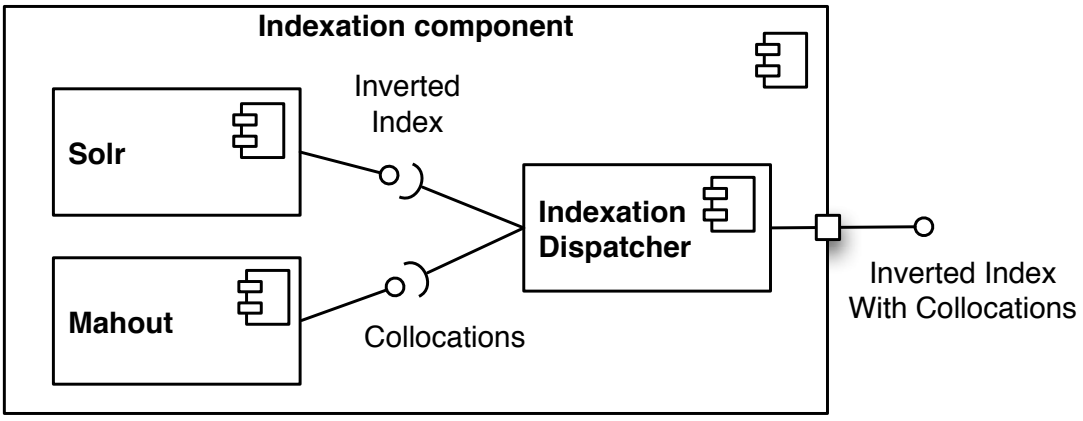


Figure 4.3 – Indexation Component Diagram (in UML notation)

The output is an inverse index of the resulting terms stored in HDFS (Hadoop Distributed

[1] Named upon the Message Dispatcher pattern [Hohpe et al., 2004]

File System), which is used for the vectorization step.

4.5.2/ VECTORIZATION

In the vectorization step, term-frequency vectors and the co-occurrence matrix are calculated.

The calculation of term-frequency vectors includes the calculation of document frequency V_{df} and term frequency in each text document $(item)V^{item}_{tfidf}$. Calculating the term-frequency vectors in a large collection of text documents is a very intensive task, as is calculating the co-occurrence. The term co-occurrence matrix cfm is created to represent the co-occurrence of any pair of terms in the items $cfm(term_i, term_j)$. To handle such an intensive task, the distribution of this process over several machines using the MapReduce model is proposed. A MapReduce algorithm to create the co-occurrence matrix was developed. Two methods are commonly used to compute a co-occurrence matrix from text: the pair method and the stripes method. These algorithms are adapted for the MapReduce paradigm [Lin, 2013] and both use the same statistical principle to find co-occurrences in text.

In the MapReduce map function, for all given items, all pairs of terms ($term_i$, $term_j$) within a window of neighborhood are counted as a co-occurrence. In the pair algorithm each co-occurrence is outputted, while in the stripes approach, each output consists of a term and the list of its co-occurring terms. As the pair algorithm is a more intensive disk input/output task, the stripes approach is adopted.

The co-occurrences are generated and passed to the reduce function for counting. The $(key_i, value_i)$ pairs generated are defined as:

— key_i is a term ($term_i \in Term$)

— $value_i$ is the list of co-occurring terms of size n: $(term_0, term_1, ..., term_n)$

Where n is the number of terms in the current item, while the neighborhood window used to define a co-occurrence is the whole item.

According to the MapReduce paradigm, the pairs are shuffled by key_i and the reduce function is executed for each set of pairs with the same key_i. The reduce function aggregates the terms from the $value_i$ of all pairs, counting each occurrence. The output of the reduce function is a list of pairs in the $(key_o, value_o)$ format where the key_o is composed of each combination of key_i and each term in $value_i$ of all input pairs, and the counted term occurrence is $value_o$.

Term-frequency vectors calculation is handled by the Mahout library [The Apache Software Foundation, 2013] deployed on a Hadoop Cluster. Also, the Hadoop cluster handles the MapReduce algorithm and stores the vectors and the matrix in HDFS. Figure 4.4 depicts the logical component diagram for the vectorization step (Hadoop not represented).

The resulting matrix is used in the hierarchization step to create a set of labels and their subsumption relations.

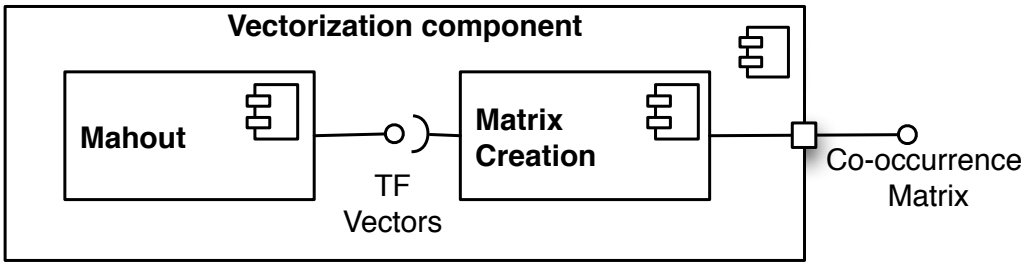


Figure 4.4 – Vectorization Component Diagram (in UML notation)

4.5.3/ HIERARCHIZATION

The hierarchization step learns the labels and their subsumption relations automatically from text documents based on a statistical approach. The labels are calculated via an information retrieval approach based on the proportion $P_C(term)$ of a $term \in Term$ and collection C.

Detecting subsumption relations between labels is a very intensive task. To distribute the process across several machines, the previously described subsumption algorithm was adapted for the MapReduce paradigm, by turning the process into a set of MapReduce operations. Considering the Item (document) frequencies of $term_i$ and $term_j$ in collection C, the co-occurrence matrix cfm can be viewed as a set of pairs $((term_i, term_j), P(x|y))$, where $P(x|y)$ is the conditional proportion of the items common to x and y, in respect to the number of items in y.

The set of pairs $< (term_i, term_j), P(x|y) >$ is used as the input of the map function. The $(key, value)$ pairs are defined as:

— key is a tuple $(term_i, term_j)$ where both $term_i$ and $term_j$ are terms identified in the Vectorization step.

— $value$ is the proportion $P(x|y)$

In the map phase, the lT threshold is applied to each pair. If both terms are included in the set ω_{lT}, the map function generates the couple $((label_i, label_j), P(x|y))$ where $(label_i, label_j)$, are terms in the set ω_{lT}.

The reduce function groups output by $label_i$, which allows computing the subsumption potential parent of each couple $(label_i, label_j)$ according to the relation defined above. The output of the reduce function is the set of couples of type $(label_i, label_j)$, where $label_j$ is a potential parent of $label_i$.

The subsumption algorithm is deployed on a Hadoop Cluster and the hierarchy of Labels is stored in HDFS. The output is a Label hierarchy that is used to classify the items. This label hierarchy is stored in the ontology-described knowledge base using the OWL-API library. Figure 4.5 depicts the logical component diagram for Hierarchization step (Hadoop not represented) where the hierarchization specific coordination is described trough a dispatcher.

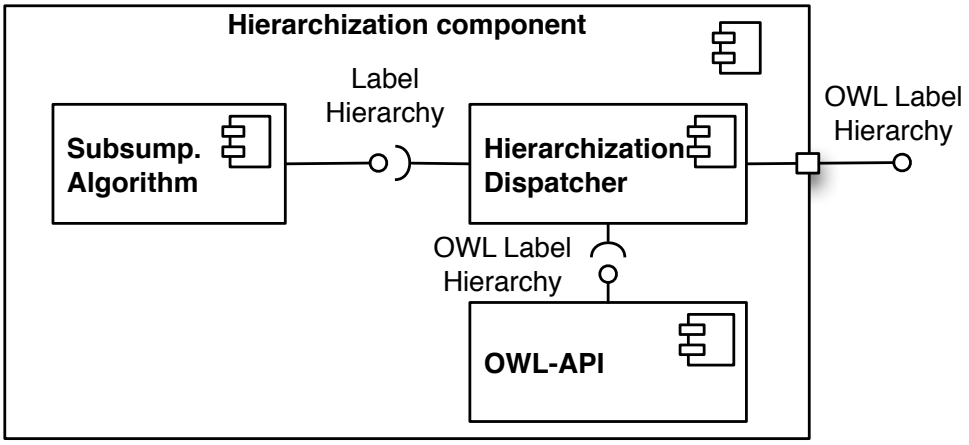


Figure 4.5 – Hierarchization Component Diagram (in UML notation)

4.6/ EVALUATION

This section presents and discusses the preliminary results of the proposed label hierarchy learning process. First, (i) the dataset, (ii) the environment and (iii) the settings used to test the process are described. Then, the process results are presented and discussed.

4.6.1/ TEST ENVIRONMENT

The dataset used in this evaluation is made of unstructured text articles extracted from dumps of the French version of *Wikipedia*. Four sub-datasets with different sizes are used as presented in Table 4.2.

Table 4.2 – Wikipedia-based Datasets

Dataset	Number of articles	Size (GB)
Wikipedia 1	174900	1.65
Wikipedia 2	407000	2.21
Wikipedia 3	994000	5.00
Wikipedia 4	2788500	11.00

The prototype is deployed on remote servers provided by Google, through their Google Compute Engine service that allows several ways to deploy a virtual Hadoop cluster and execute MapReduce Jobs over it. For this preliminary test, the cluster is composed of 4 nodes (1 master node and 3 workers). Each node corresponds to an instance in Google Compute Engine, i.e. a virtual machine with its proper resources.

Table 4.3 describes the specifications of the instances used for this test. The native Hadoop web interfaces in conjunction with Apache Ambari[2] is used to monitor the differ-

[2]https://ambari.apache.org/

ent sub-processes of the prototype. In particular, the main interest is to monitor the time and resource costs for the different phases.

Table 4.3 – Test node specifications

Resource type	Description
CPU (per node)	2.5GHz Intel Xeon E5 v2 (Ivy Bridge)
RAM (per node)	7.5GB
Disk (per node)	500GB

4.6.2/ SETTINGS

Some thresholds and settings used in the process have a high impact on the results. In the indexation step the seq2sparse tool from Mahout [3] is used to retrieve the collocation terms with the Lucene French analyzer and prune with the Minimum support, n-gram size, Maximum document frequency, Minimum document frequency and Log-Likelihood Ratio settings. More details about each Mahout option can be found in [The Apache Software Foundation, 2013]. Table 4.4 shows the different parameters and their used values. The same values are used for all datasets.

Table 4.4 – Execution Settings

Parameter	Step	Value
Minimum support	Indexation	1750
n-gram size	Indexation	2
Maximum document frequency	Indexation	90%
Minimum document frequency	Indexation	1
Log-Likelihood Ratio	Indexation	17500
Label Detection Threshold (lT)	Hierarchization	2%
Subsumption st2 Threshold	Hierarchization	10%
Subsumption st1 Threshold	Hierarchization	90%

4.6.3/ RESULTS

The aim of the preliminary test is to check the scalability of the classification process according to sub-datasets of different sizes from the same dataset. To that end, the following indicators are monitored: (1) The execution time for each step and dataset; (2) The number of extracted Terms from each dataset; (3) The number of learned Labels from each dataset; (4) The number of learned Subsumption relations.

The number of extracted terms is depicted in Fig. 4.6 where the reader can observe that the number of terms grows according to the number of items. As the analyzed data is

[3] http://mahout.apache.org

textual data, a limit of growth (not visible with the dataset used) is expected to happen, i.e. the language dictionary and its derivations' size. Text processing such as stemming and synonym detection can be optimized to further restrict the number of terms detected.

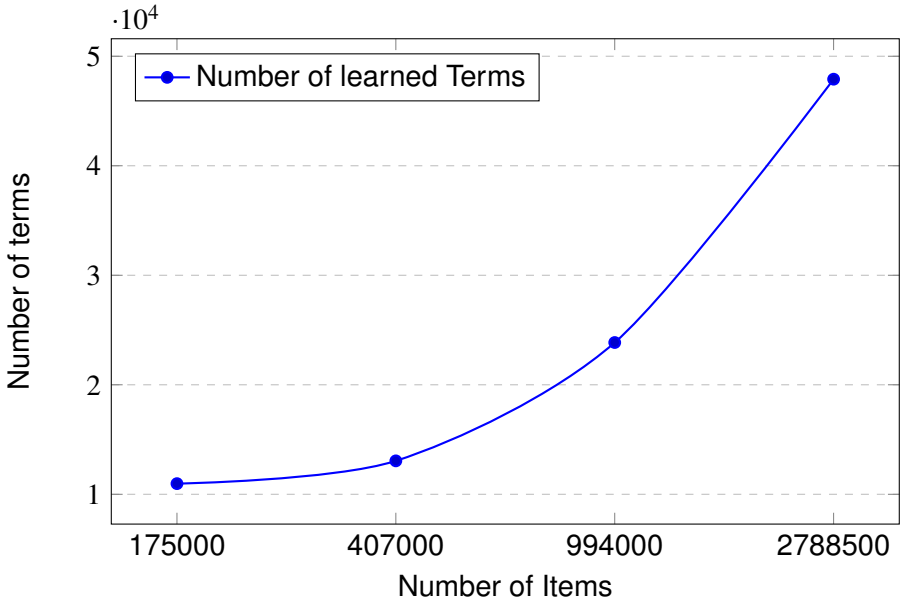


Figure 4.6 – Number of extracted Terms according to the number of items in each dataset.

The number of learned labels for each dataset is depicted in Fig. 4.7. As the label set is a subset of the set of terms extracted from an information retrieval approach, the extracted number of terms impacts the number of learned Labels. Nethertheless, one can observe that the number of learned labels decreases while the size of the dataset grows. This is a consequence of the label detection threshold being fixed for all datasets with the value of 2% (Table 4.4). As the dataset grows, this threshold makes it unlikely for a term to appear at the same rate.

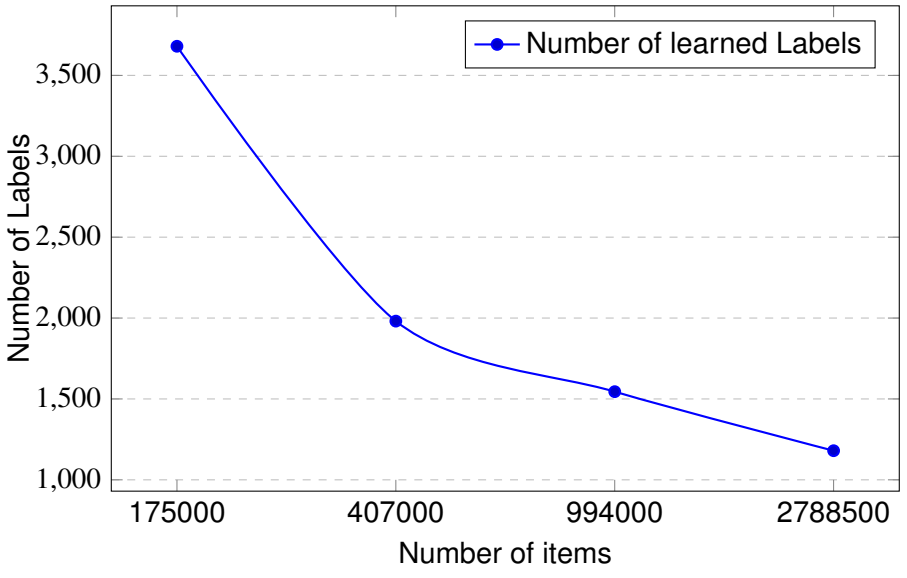


Figure 4.7 – Number of extracted Labels according to the number of items in each dataset.

The number of learned subsumption relations for each dataset is depicted in Fig. 4.8. The reader can observe a decrease in the number of learned relations as a consequence of the decrease of the number of learned labels depicted in Fig. 4.7.

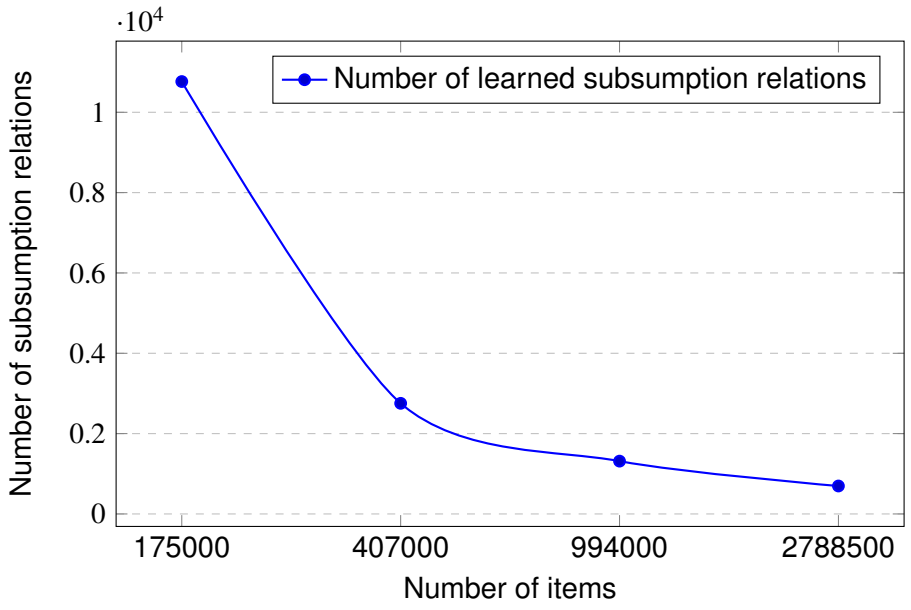


Figure 4.8 – Number of learned subsumption relations according to the number of items in each dataset.

The execution time of each step by dataset is depicted in Fig. 4.9. Observing the results, the computation time of the matrix creation is the most expensive of the process, regardless of the data size.

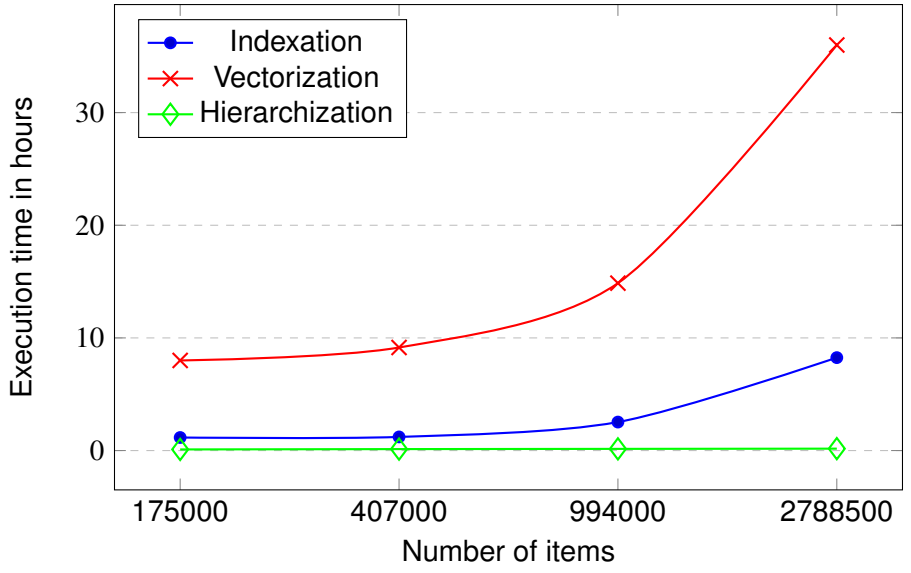


Figure 4.9 – Execution time of each step by dataset.

4.7/ SUMMARY

To perform Hierarchical Multi-Label classification in Big Data without overloading users, this chapter proposes an unsupervised process of learning a label hierarchy from unstructured text. The process is based on a hybrid approach between statistical and lexical approaches to learn the labels and a highly scalable subsumption method to learn the hierarchical relations between labels. The process prototype was successfully implemented in a scalable and distributed platform to process Big Data. The pertinence of using the automatically learned hierarchy in Hierarchichal Multi-label context is evaluated in the end of Chapter 5.

5

HIERARCHICAL MULTI-LABEL CLASSIFICATION USING WEB REASONING

The Semantic HMC is based on an unsupervised ontology learning process using scalable Machine-Learning techniques and Rule-based reasoning. The process is unsupervised, as no previously classified examples or rules to relate the data items with the labels exist. The ontology-described knowledge base (Abox+Tbox) used to represent knowledge in the classification system is automatically learned from huge volumes of data through highly scalable Machine Learning techniques and Big Data technologies. First, the taxonomy is automatically obtained and used as the first input for ontology construction. Then, for each taxonomical concept (classification labels), a set of rules is created in order to relate the data items to the taxonomy concepts. Next, the learned ontology is populated with the data items. In Chapter 3, the Semantic HMC proposes five individually scalable steps to reach the aims of Big Data analytics: (i) Indexation, (ii) Vectorization, (iii) Hierarchization, (iv)Resolution and (v) Realization.

The first three steps learn the label hierarchy from unstructured data as is described in the preceding Chapter 4. As a follow-up, this chapter focuses on the two last steps of the Semantic HMC process. It proposes a new process so as to hierarchically multi-classify items from huge sets of unstructured texts using DL ontologies and Rule-based reasoning. The process is implemented using scalable approaches that distribute the process by several machines in order to reach the high performance and scalability required by Big Data. The process is evaluated and compared with some multi-label classification algorithms from the state of the art where the Semantic HMC approach outperforms state of the art approaches in some cases.

The remaining of this chapter is structured into six sections. Section 5.1 presents the specific background knowledge and related work in classification using ontologies and web reasoners. Section 5.2 introduces the Resolution step where the classification rules are created. Section 5.3, systematizes the Realization step where the documents are classified according to the classification rules. Section 5.4 shows the implementation in a scalable and distributed platform to process Big Data. Section 5.5 evaluates the whole Semantic HMC process regarding some state of the art approaches and discusses the results obtained. Finally, section 5.6 summarizes this chapter.

5.1/ Specific Background and Related Work

In this section, some specific background and current related work in the automatic hierarchical multi-label classification of unstructured text is discussed by specifically emphasizing on the use of DL ontologies and reasoning in this field. Reasoning is used for ontology development, maintenance, as well as solving application issues [Moller et al., 2009]. Web reasoning can be used to improve the classification process. In [Fang et al., 2010] authors present a document classification method that uses ontology reasoning and similarity measures to classify the documents. Most of the literature focuses on describing or improving the classification processes using ontologies, but few take advantage of the reasoning capabilities of web reasoning to automatically multi-classify the items. In [Ben-David et al., 2010] authors introduce a generic, automatic classification method that uses Semantic Web technologies to perform the classification and represent the results. This method allows data elements from diverse sources and of various formats to be classified using a universal classification scheme. The proposed generic classifier is based on an ontology, which gives a description of the entities that need to be discovered and the classes to which these entities will be mapped to. In [Werner et al., 2014], the authors propose a method to semantically enrich the ontology used to hierarchically describe the domain and to process the classification of news using the hierarchy of terms. This ontology aims to reduce the gap between the expert's perspective and the representation of classification rules. To enrich the ontology and classify the documents, an out-of-the-box DL Web Reasoner such as Pellet [Sirin et al., 2007], FaCT++ [Tsarkov et al., 2006], or Hermit [Shearer et al., 2009] is used. Out-of-the-box reasoners can be used to classify small datasets but their scalability is limited and they cannot be used with large datasets [Werner et al., 2014] as it is required in a Big Data context.

However, Web Scale Reasoners [Urbani, 2013], instead of using traditional Description Logics approaches like Tableau [Sirin et al., 2007, Tsarkov et al., 2006], Resolution [Motik et al., 2006] or Hypertableau [Shearer et al., 2009], use entailment rules for reasoning over ontologies. In [Wu et al., 2013] authors describe a semantic web rule execution mechanism using MapReduce which can be used with OWL-Horst and with SWRL rules.

The difference of the method proposed in [Werner et al., 2014] is that instead of translating the rules into logical constraints of an ontology captured in Description Logic, these rules are translated into Horn Clause rules. The main interest in using Horn Clause rules is to reduce the reasoning effort, thus improving the scalability and performance of the system. The aim is to use more but simpler rules that will be applied to the ontology in order to classify items.

In summary, to the best of our knowledge, a classification process to automatically classify text documents in Big Data context by taking advantage of ontologies and rule-based reasoning to perform the classification is novel and outperforms state-of-the-art approaches in some specific cases.

5.2/ Resolution

The resolution step creates the ontology rules used to relate the labels and the data items, i.e. it establishes the conditions for $item_1$ to be classified as $label_1$. The rules will define

the necessary and sufficient terms of an item so that the item is classified in label.

In Vectorization step, a term co-occurrence frequency matrix $cfm(term_i, term_j)$ is created to represent the co-occurrence of any pair of terms in the collection of items C. Based on that, let $P(term_j|term_i)$ be the conditional proportion (number) of the items from collection C that are common to $term_i$ and $term_j$, in respect to the number of items in $term_j$ such that:

$$P_C(term_i|term_j) = \frac{cfm(term_i, term_j)}{cfm(term_j, term_j)} \tag{5.1}$$

Two thresholds are defined:

— Alpha threshold (α) such that $\alpha < P_C(term_i|term_j)$, where $term_i \in Label$ and $term_j \in Term$.

— Beta threshold (β) such that $\beta \leq P_C(term_i|term_j) \leq \alpha$, where $term_i \in Label$ and $term_j \in Term$.

These two thresholds are user-defined with a range of $[0, 1]$. Based on these thresholds, two sets of terms are identified (Fig.5.1):

— Alpha set ($\omega_\alpha^{(term_i)}$) is the set of terms for each label such that:

$$\omega_\alpha^{(term_i)} = \{term_j | \forall term_j \in Term : P_C(term_i|term_j) > \alpha\} \tag{5.2}$$

i.e. is the set of terms $term_j$ that co-occur with $term_i \in Label$ with a co-occurrence proportion higher than the threshold α.

— Beta set ($\omega_\beta^{(term_i)}$) is the set of terms for each label such that:

$$\omega_\beta^{(term_i)} = \{term_j | \forall term_j \in Term : \beta \leq P_C(term_i|term_j) \leq \alpha\} \tag{5.3}$$

i.e. is the set of terms that co-occur with $term_i \in Label$ with a co-occurrence proportion higher or equal to threshold β and lower than threshold α.

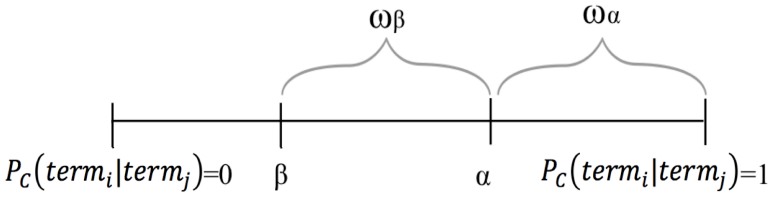


Figure 5.1 – Alpha and Beta sets

The process of rule creation uses thresholds as proposed in [Werner et al., 2014] to select the necessary and sufficient terms. Regarding the existence of Alpha and Beta sets for each item, four item categories are identified:

— Beta Empty set:

$$|\omega_\alpha^{(term_i)}| > 0 \wedge |\omega_\beta^{(term_i)}| = 0 \tag{5.4}$$

— Alpha Empty set:

$$|\omega_{\alpha}^{(term_i)}| = 0 \wedge |\omega_{\beta}^{(Label_i)}| > 0 \tag{5.5}$$

— Alpha and Beta not Empty set:

$$|\omega_{\alpha}^{(term_i)}| > 0 \wedge |\omega_{\beta}^{(term_i)}| > 0 \tag{5.6}$$

— Alpha and Beta Empty set:

$$|\omega_{\alpha}^{(term_i)}| = 0 \wedge |\omega_{\beta}^{(term_i)}| = 0 \tag{5.7}$$

Rules are created for the first three categories as follows. In an empty beta category, only the ω_{α} is considered. Items are classified with labels if:

$$\forall label \forall item \exists term : hasTerm(item, term) \wedge term \in \omega_{\alpha}^{label} \rightarrow isClassified(item, label) \tag{5.8}$$

i.e. if the item has at least one term in $\omega_{\alpha}^{(term_i)}$ it is classified with $term_i$, $term_i \in Label$. For each term that complies with the above rule, a SWRL rule is created. For example, for a $|\omega_{\alpha}^{(term_i)}| = |\{t_1, t_2\}|$, the generated SWRL rules are presented in Table 5.1. In empty alpha category only the ω_{β} is considered. Items are classified with labels if:

$$\begin{aligned} \forall label \forall item : |\{\forall term : hasTerm(item, term) \wedge term \in \omega_{\beta}^{label}\}| \geq \delta \\ \rightarrow isClassified(item, label) \end{aligned} \tag{5.9}$$

i.e. if the item has at least δ terms in $\omega_{\beta}^{(term_i)}$, it is classified with $term_i$, $term_i \in Label$. One SWRL rule is generated for each combination of $term_j \in \omega_{\beta}^{(term_i)}$ where the number of combined terms is at least $\delta = \lceil |\omega_{\beta}^{(term_i)}| * p \rceil$, and $0 \leq p \leq 0.5$. For example, for a $|\omega_{\beta}^{(term_i)}| = |\{t_1, t_2, t_3\}| = 3$ and $p = 0.5$ resulting in $\delta = \lceil 3 * 0.5 \rceil = 2$, the generated SWRL rules are presented in Table 5.2. The set of generated beta rules is the combination $\binom{n}{m}$ of m terms of a larger set of n elements. Regarding the proposed approach, n is the number of possible terms $|\omega_{\beta}^{(term_i)}|$, and m is the minimum number of terms δ in each rule (e.g. $\binom{20}{10} = 184756$). In order to limit the number of rules for each label, the value of $n \leq 10$ is fixed. The terms are selected by ranking the terms in $\omega_{\beta}^{(term_i)}$ using the conditional proportion $P_C(term_i|term_j)$ as the ranking score.

Table 5.1 – Generated Alpha Rules (Example)

Alpha rules
$Item(?it), Term(?t_1), Label(?t_1), hasTerm(?it, ?t_1) \rightarrow isClassified(?it, ?t_1)$
$Item(?it), Term(?t_2), Label(?t_2), hasTerm(?it, ?t_2) \rightarrow isClassified(?it, ?t_2)$

Notice that the rules that encompass more than δ terms are not necessary because the combination of any δ terms is sufficient to classify the item.

In non-empty alpha and beta category, beta and alpha rules are both considered. Alpha rules are evaluated as presented in the empty beta category. Beta rules are evaluated as presented in the empty alpha category but with a value of $q = p * 2$. It corresponds to $\delta = \lceil |\omega_{\beta}^{(term_i)}| * q \rceil$, with $0 \leq q \leq 1$ and $q = p * 2$.

Table 5.2 – Generated Beta Rules (Example)

Beta rules
$Item(?it), Term(?t_1), Term(?t_2), Label(?t_3), hasTerm(?it, ?t_1), hasTerm(?it, ?t_2) \rightarrow isClassified(?it, ?t_3)$
$Item(?it), Term(?t_1), Term(?t_3), Label(?t_2), hasTerm(?it, ?t_1), hasTerm(?it, ?t_3) \rightarrow isClassified(?it, ?t_2)$
$Item(?it), Term(?t_2), Term(?t_3), Label(?t_1), hasTerm(?it, ?t_2), hasTerm(?it, ?t_3) \rightarrow isClassified(?it, ?t_1)$

For the concepts in the fourth category (Alpha and Beta Empty) no enrichment rules are created because the cardinality of the sets is zero. The result of the resolution phase is the set of all the necessary and sufficient rules to classify an item with a label.

5.3/ REALIZATION

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:

$$\omega_{\gamma}^{(item_i)} = \{term_j | \forall term_j \in Term \ \wedge \gamma < tfidf_{(item_i, term_j, C)}\} \tag{5.10}$$

where γ is the relevance threshold, $\gamma < tfidf_{(item_i, term_j, C)}$, $term_j \in Term$, $item_i \in Item$ and $tfidf$ as calculated in the Vectorization step.

The classification sub-step performs the multi-label hierarchical classification of the items.

The rule-based inference engine uses 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 on a hierarchical label structuring (Hierarchical Multi-label Classification). To infer the most specific labels, the rules generated in the resolution step are used. In addition, the following SWRL rule is used to classify an item with any subsuming label:

$$\begin{gathered} Item(?item), Label(?labelA), Label(?labelB), broader(?labelA, ?labelB), \\ isClassified(?item, ?labelA) \rightarrow isClassified(?item, ?labelB) \end{gathered} \tag{5.11}$$

These rules can be applied either in a forward chaining (i.e. materialization) or backward chaining. Based on these two types of rule-based reasoning, two types of classification are proposed: classification before query time and classification at query time.

Classification before query time (Fig. 5.2) is performed using a forward-chaining inference engine to create a closure with all inferred data, i.e. the inference rules are applied over the entire ontology-described knowledge base until all possible data is derived and materialized. After the closure has been calculated, the query engine directly queries the closure, retrieving the classifications very quickly. On the other hand, the closure must be updated for every change in the ontology-described knowledge base. Therefore, creating

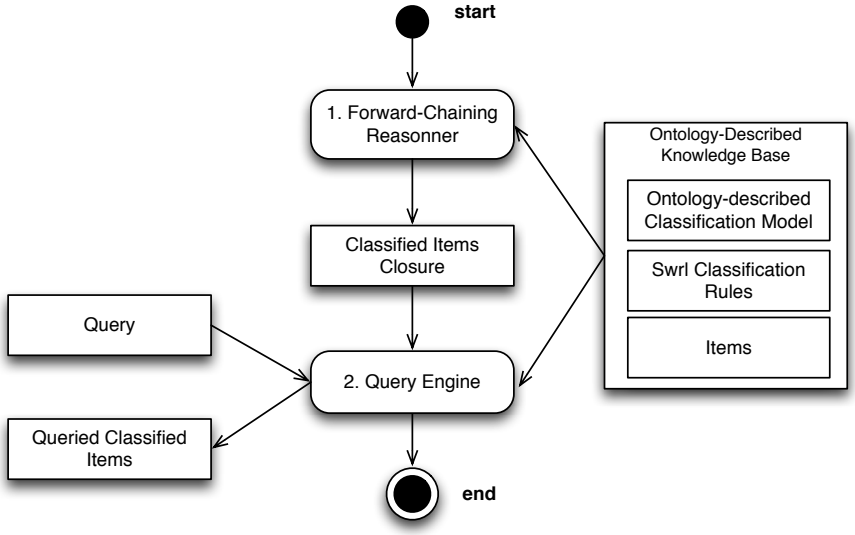


Figure 5.2 – Classification before query time

a closure of inferred data can be expensive due to data volume, velocity of changes, and the quantity and complexity of rules.

Classification on query time (Fig. 5.3) is performed by backward-chaining inference applying the rules only over the strictly necessary data to answer the query. By applying the rules over the strictly necessary data, it has the advantage of addressing the rapidly changing data feature of Big Data. However, the main disadvantage is that, for each query, activating the inference engine is always necessary, which is affected by the volume and quantity and expressivity of rules.

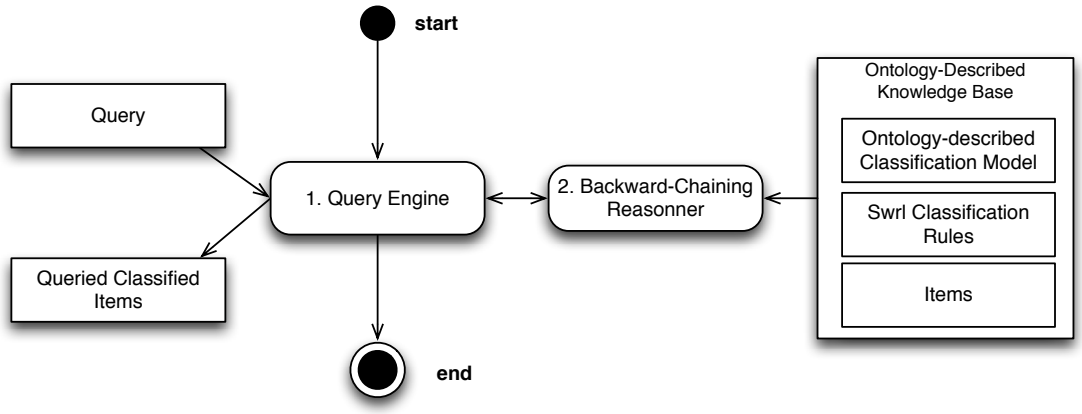


Figure 5.3 – Classification on query time

5.4/ IMPLEMENTATION

This section describes the implementation of the last two steps of the Semantic HMC process. The process is implemented as a Java application, as many tools used to develop

the prototype are developed in Java.

As in the first three steps (indexation, vectorization and hierarchization) of the Semantic HMC process, Big Data processing techniques such as MapReduce are used to implement the Resolution and are part of the realization step. The rule creation algorithm described in the previous section is implemented in MapReduce, and deployed in a Hadoop server. The Hadoop implementation of MapReduce was chosen because of its open-source nature and its ease of integrating other tools used in the process. The vectors, the co-occurrence matrix created in the last chapter, are stored in HDFS and are now used in the resolution and realization steps.

The next subsections describe the implementation details of the Resolution and Realization steps of the Semantic HMC process.

5.4.1/ RESOLUTION

The resolution process creates the ontology rules used to relate the labels and the data items. It is assumed that α and β thresholds are user-defined settings. The rule creation process is divided into a sub-process for each $label_i \in Label$. In each sub-process $\omega_{\alpha}^{(label_i)}$ and $\omega_{\beta}^{(label_i)}$ sets are calculated using the co-occurrence matrix, and then classification rules are created for each label. Exploiting a huge co-occurrence matrix so as to create ontology rules is a very intensive task, thus this process is also distributed to several machines in the MapReduce paradigm. A MapReduce job creates the rules from the co-occurrence matrix.

The set of pairs $< (term_i, term_j), P(x|y) >$ is used as the input of the map function. The $(key, value)$ pairs are defined as:

— key is a tuple $(term_i, term_j)$ where both $term_i$ and $term_j$ are terms identified in the Vectorization step.

— $value$ is the proportion $P(x|y)$

In the map phase, the α and β thresholds are applied to the proportion $P(x|y)$ of each pair $< (term_i, term_j), P(x|y) >$ where $term_i \in \omega_{lT}$ or $term_j \in \omega_{lT}$. The map function outputs a list of $< (RuleType, label_i), term_j >$ pairs where:

— RuleType is a descriptor for the type of rule (alpha or beta)

— $label_i$ is the label that was related by the new rule

— $term_j$ is a term used to relate items with $label_i$ which can be the term of an alpha rule, or a term comprised in a beta rule

According to the MapReduce paradigm, pairs are shuffled by key (i.e.($RuleType$, $label_i$)) and the MapReduce "reduce" function is executed for each set of pairs with the same key. The "reduce" function aggregates the rules by $label_i$ and outputs the set of alpha terms $\omega_{\alpha}^{(term_i)}$ and beta terms $\omega_{\beta}^{(term_i)}$ for each $label_i$. The rules are serialized in SWRL language and stored in the ontology-described knowledge base using the OWL-API library. Figure

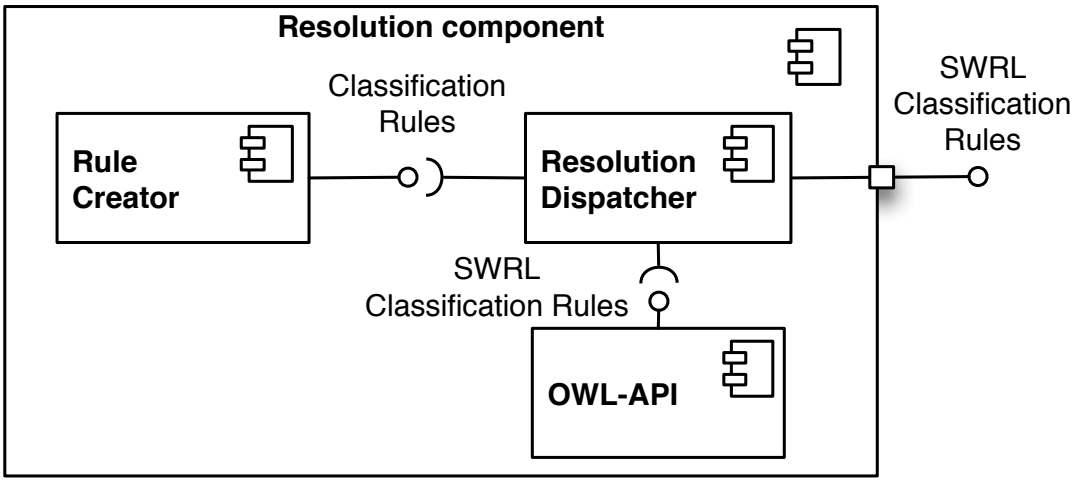


Figure 5.4 – Resolution Component Diagram (in UML notation)

5.4 depicts the logical component diagram for Resolution step (Hadoop not represented) where the resolution specific coordination is described trough a dispatcher.

The generated rules, along with the label hierarchy, are used in the Realization process to classify new items.

5.4.2/ REALIZATION

The realization step populates the ontology and performs the multi-label hierarchical classification of the items.

First, the ontology is populated with new items and the most relevant terms to describe each document in an assertion level (Abox). The $tfidf$ vectors for each document calculated in vectorization allow measuring term relevancy in a text document (item) and calculate the set of relevant terms $\omega_{\gamma}^{(item_i)}$. In order to store, manage and query the ontology-described knowledge base (Tbox+Abox), a triple store is used. Because highly expressive forward chaining DL reasoners do not scale well and, on the other hand, because Web-Scale Reasoners like WebPie based on the MapReduce programming model are limited to low expressive ontologies such as the OWL-Horst fragment, a decision was made to adopt the classification at query time approach in the preliminary prototype, by using a triple store with a backward-chaining inference engine.

Due to backward-chaining query performance issues identified in [Farias et al., 2015], a rule selection method was developed to execute only the rules needed to classify the items for each type of query. Two main query types were identified: (1) to retrieve all items classified with a label and (2) to retrieve all labels that classify an item.

To retrieve all items classified with a label $label_i$ only the rules with $label_i$ in the rule head (i.e. $isClassified(?item, label_i)$) are activated. To retrieve the labels that classify an item $item_i$ only the rules with at least one term $term_i \in \omega_{\gamma}^{(item_i)}$ in the rule tail (i.e. $(hasTerm(?item, term_i))$ are activated.

The OWL-API library is used to populate the OWL ontology with new items. A scal-

able triple store called Stardog (http://docs.stardog.com) is used to store and query the ontology-described Knowledge Base (Tbox+Abox). Stardog is also used to perform reasoning by backward-chaining inference as well as SWRL rules inference. The rule selector was developed in java and interacts with Stardog to optimize query performance. Figure 5.5 depicts the logical component diagram for the Realization step (Hadoop not represented) where the realization specific coordination is described trough a dispatcher.

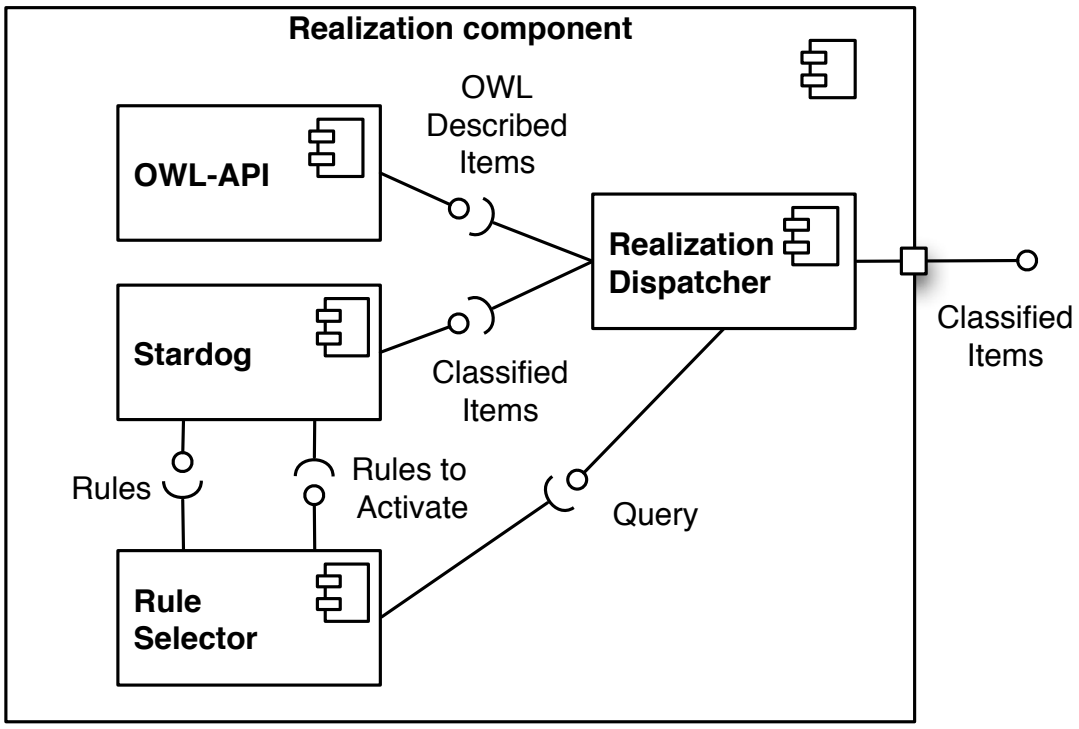


Figure 5.5 – Realization Component Diagram (in UML notation)

5.5/ EVALUATION

In this section, the classification performance of the Semantic HMC process is evaluated for unstructured text classification in a Big Data context. First, preliminary results of the proposed classification process are discussed regarding a large dataset extracted form Wikipedia. Secondly, the quality of the results is evaluated using a smaller dataset that is often used in multi-label classification literature called Delicious [1] [Tsoumakas et al., 2008]. Finally, the classification results obtained by some algorithms from the state of the art are compared and discussed.

5.5.1/ EXPERIMENTS WITH LARGE DATASET

The preliminary results of the proposed classification process of a large dataset of unstructured text documents are discussed. Firstly, the dataset, environment and settings

[1] http://mulan.sourceforge.net/datasets-mlc.html

used to test the process are described. Then, the experiment's results are presented and discussed. The dataset is composed of unstructured text articles. The articles are extracted from dumps of the French version of Wikipedia with different sizes as described in Table 5.3.

Table 5.3 – Wikipedia-based Datasets

Dataset	Number of articles	Size (GB)
Wikipedia 1	174900	1.65
Wikipedia 2	407000	2.21
Wikipedia 2	994000	5.00

Some thresholds and settings used in the process have a strong impact on the results. Table 5.4 shows different parameters and their values used in preliminary results. The same values are used for all datasets. The co-occurrence matrix and the hierarchy calculated on Chapter 4 are used as input. The number of terms, labels, and subsumption relations are presented in Table 5.5.

Table 5.4 – Execution Settings

Parameter	Step	Value
Alpha Threshold	Resolution	90
Beta Threshold	Resolution	80
Term ranking (n)	Resolution	5
p	Resolution	0.25
Term Threshold (γ)	Realization	2

Table 5.5 – Number of Terms, Labels and Hierarchy relations

Dataset	Wiki 1	Wiki 2	Wiki 3
Number of Terms	10973	13053	23859
Number of Labels	3680	1981	1545
Number of relations	10765	2754	1315

The aim of the preliminary test is to check the scalability of the system according to the number of items from the same dataset. To that effect, the following indicators are monitored: (1) The number of learned classification rules (i.e. α and β rules); (2) The number of classifications (i.e. isClassified relations) from each sub-dataset.

In Chapter 4 it was demonstrated that the number of labels decreases when the size of the dataset grows. The number of learned classification rules (i.e. α and β rules) for each sub-dataset is depicted in Fig. 5.6. The reader can observe a decrease in the number of learned rules as a consequence of the decreased of the number of learned labels.

The number of classification relations (isClassified) for each sub-dataset is depicted in Fig. 5.7. The reader can observe an increase in the number of classifications while the

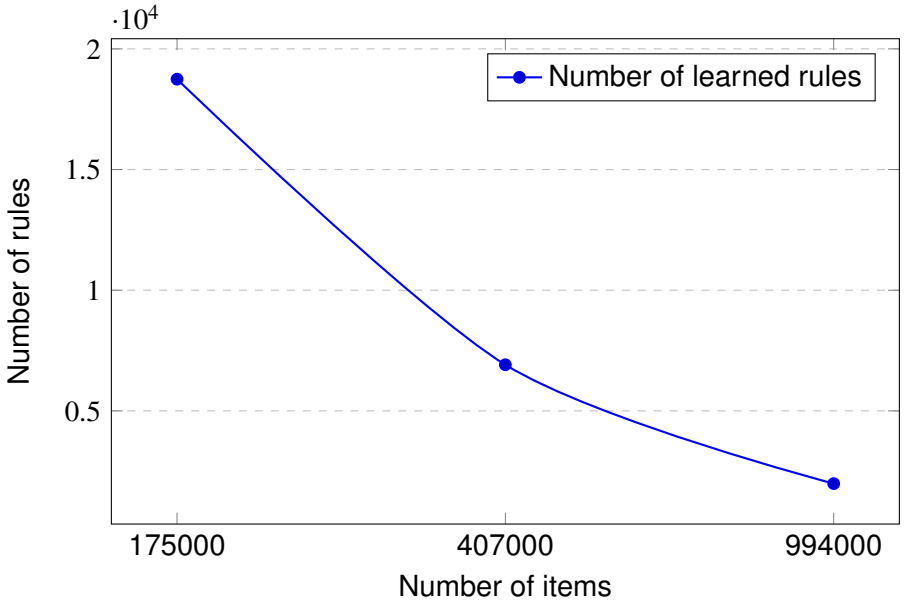


Figure 5.6 – Number of learned rules according to each dataset.

size of the dataset grows even if the number of rules decrease.

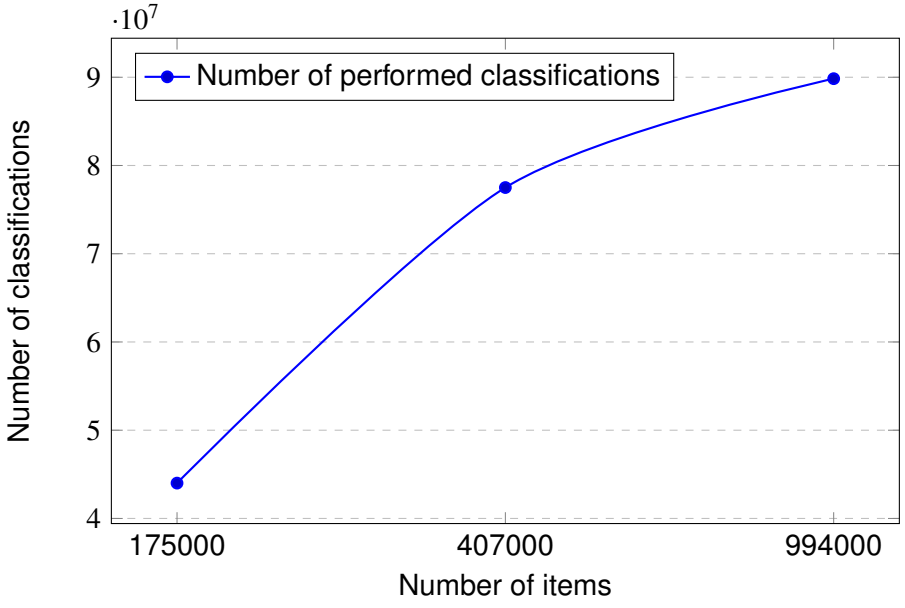


Figure 5.7 – Number of classifications (learned isClassified relations) according to each dataset.

5.5.2/ Quality Evaluation

In this subsection, the classification performance of the Semantic HMC process for unstructured text classification in a Big Data context is evaluated. First there's a description of the dataset, the test environment and the experimental settings used to evaluate the process. Then, the experimental results are presented and discussed.

Comparing unsupervised multi-label methods' quality is subjective because different labels are considered according to the label selection methods used. To be able to compare the proposed approach with the approaches from the state of the art, a pre-defined set of labels is used instead of automatically learned labels, as it is described in the previous experiment. The evaluation is done using a pre-labelled dataset (supervised), composed of training and test data. The training set is used to learn hierarchical relations between the pre-defined labels and classification rules. The test set is used to calculate the classification quality of the algorithm based on standard quality measures (i.e. precision, recall, F-measure).

The Delicious dataset is used to perform this evaluation. This dataset is composed of labeled textual data from extracted web pages of the Delicious social bookmarking website [Tsoumakas et al., 2008]. Table 5.6 shows the dataset specifications. The Delicious dataset contains very few features (words) compared to the number of labels; which makes accurate classification difficult [Papanikolaou et al., 2015]. Furthermore, it has been used to evaluate several multi-label classification systems, thus providing a good baseline for comparison in the proposed approach.

Table 5.6 – Delicious dataset specifications

$\|Train\|$	$\|Test\|$	$\|Labels\|$	$\|Terms\|$
12,910	3,181	983	500

To evaluate the quality of the SHMC process, previous studies in HMC are used as reference (Section 2.3.5). The classification quality is evaluated based on standard quality measures (i.e. precision, recall, F-measure). The Area Under the PR Curve (AUPRC) measure is not used because the process doesn't allow direct label ranking for each item nor consequently the use of a threshold to limit the number of labels for each document.

The Hierarchization phase of the Semantic HMC process automatically generates hierarchical relations between labels. This hierarchy, along with the classification rules created in the Resolution step, are used to perform hierarchical multi-label classification. Fig. 5.8 shows a sample of the hierarchichal relations between labels that were automatically created for the Delicious dataset. The set of parameters used to create the hierarchy and classification rules is described in table 5.7. These parameters can have a strong impact in the quality of the results. The Top and Bottom Tresholds are used to calculate hierarchichal relations between labels as defined in Chapter 4.

Table 5.8 shows the results obtained by the Semantic HMC process to classify the Delicious dataset. It is observed that the micro-averaged precision and recall are higher than the macro-averaged precision and recall. Plus, the precision is lower than the recall in both cases.

5.5.3/ Comparison with the state of the art

Table 5.9 shows the Macro-F1 measure and Micro-F1 measure obtained when classifying the Delicious dataset. The results of the proposed process (SHMC) are compared with several results from state-of-the-art approaches with the same dataset [Madjarov et al., 2012, Sucar et al., 2014, Papanikolaou et al., 2015].

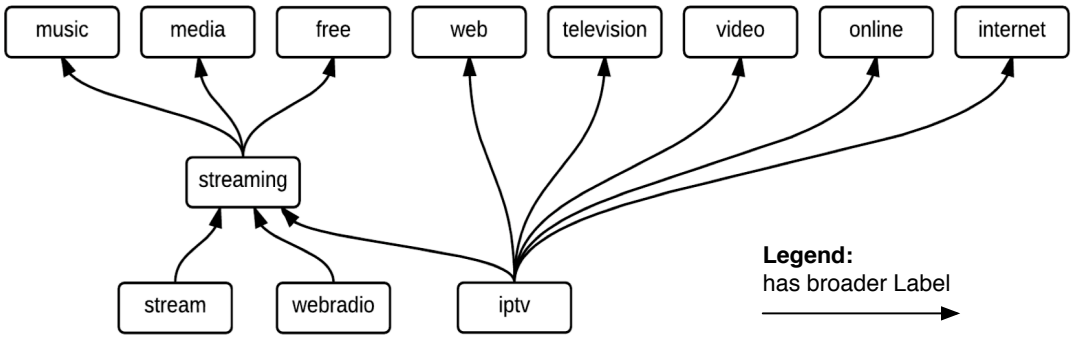


Figure 5.8 – Automatically generated hierarchy from Delicious dataset (sample)

Table 5.7 – Execution Settings for the Semantic HMC to classify the Delicious Dataset

Parameter	Step	Value
Top Threshold	Hierarchization	50
Bottom Threshold	Hierarchization	40
Alpha Threshold	Resolution	20
Beta Threshold	Resolution	10
Term ranking (n)	Resolution	5
p	Resolution	0.25
Term Threshold (γ)	Realization	2

Table 5.8 – Quality results for the Delicious Dataset

	Precision	Recall	F1-measure
Micro	0.284	0.74	0.410
Macro	0.0676	0.178	0.0979

The state-of-the-art approaches used for comparison are: Binary relevance (BR) [Tsoumakas et al., 2007], Classifier chaining (CC) [Read et al., 2011], TNBCC [Sucar et al., 2014], Path-BCC [Sucar et al., 2014], Hierarchy Of Multi-label Classifiers (HOMER) [Tsoumakas et al., 2008] , Multi-label k-nearest neighbors (ML-kNN) [Zhang et al., 2007], Random forest of predictive clustering trees (RF-PCT)[Kocev et al., 2007], Random forest of ML-C4.5 (RFML-C4.5) [Kocev et al., 2007] and the Collapsed Gibbs Sampling (CGS) algorithm [Griffiths et al., 2004].

One can observe in Table 5.9 that the Semantic HMC approach outperforms the state-of-the-art approaches in micro F1-measure, while the macro F1-measure is comparable to most other approaches. These results show that the quality classification performance of the proposed ontology-based approach is comparable to the quality performance of the selected algorithms from the state-of-the-art in machine learning.

Table 5.9 – Performance of various algorithms to classify the Delicious dataset

Algorithm	Macro F1	Micro F1
SHMC	0.0979	**0.410**
CGS_p	0.10378	0.29740
TNBCC	0.0880	N/A
BRM	0.0812	N/A
Path-BCC	0.084	N/A
BR	0.096	0.234
CC	0.100	0.236
HOMER	0.103	0.339
ML-kNN	0.051	0.175
RFML-C4.5	**0.142**	0.269
RF-PCT	0.083	0.248

5.6/ SUMMARY

This chapter proposes a new approach to automatically classify text documents by taking advantage of ontologies and rule-based reasoning to perform the classification in a Big Data context. To that effect, it describes in detail the two last steps of the classification process (Resolution and Realization). In the Resolution step, for each label a set of rules is created to relate the data items to the taxonomy concepts. In the Realization step, the learned ontology is populated with the data items resulting in an ontology-described Classification Model. A rule-based web reasoner is used to classify the items with labels. Despite the fact that both types of classification can be used in the Semantic HMC process, a careful combination of both processes is necessary due to the type of use cases in the system (i.e. retrieve all data or parts of data). Due to the state of the art limitations of reasoners, only the classification on query time was considered, experimented and evaluated. The process prototype was successfully implemented in a scalable and distributed platform to process Big Data.

The experiment section highlights two aspects of the Semantic HMC process:

1. First, the process can learn classification rules from huge amount of data and classify documents automatically.

2. Secondly, evaluation results prove that the classification performance of the Semantic HMC process that uses ontologies and rule-based reasoning to classify unstructured text documents is comparable to the performance of algorithms from the state-of-the-art in the machine learning field.

III

Maintenance of the Semantic Hierarchical Multi-label Classification Process

6

HIERARCHY RELATIONS MAINTENANCE REGARDING DATA STREAMS

The automatic concept (label) hierarchy extraction from unstructured documents is not a trivial process and requires proper techniques for document analysis and representation. In the context of Big Data, this task is even more challenging due to Big Data characteristics. The previous Semantic HMC process automatically learns the rule-enriched ontology-described classification model from a very large set of text documents (Chapter 4).

In Big Data, however, relearning the hierarchy for each new document is infeasible due to Big Data characteristics. Most of the existing literature focuses on learning concept hierarchies from texts [Medelyan et al., 2013, Caraballo, 1999]. Few works have been published in maintaining the hierarchy once it is created without relearning the whole hierarchy. This chapter aims to present a maintenance process of the ontology-described label hierarchy relations with regards to a stream of unstructured text documents in the context of Big Data, incrementally updating the label hierarchy.

The remaining of this chapter is structured in eight sections. Section 6.1 presents the specific background knowledge and related work in automatic hierarchy learning from unstructured texts. Section 6.2 proposes the hierarchy relations maintenance process regarding a stream of text documents. Sections 6.3 and 6.4 describe in detail the proposed maintenance process. Section 6.5 shows the implementation in a scalable and distributed platform to process Big Data. Section 6.6 discusses the results obtained. Finally, the last section 6.7 summarizes this chapter.

6.1/ SPECIFIC BACKGROUND AND RELATED WORK

This section introduces some background and discusses the current related work about automatic concept hierarchy learning from streams of text.

Most work in this area focuses on hierarchy creation and few works have been done on hierarchy maintenance since it's been created, even less in Big Data context. Liu et al. [Liu et al., 2012b] automatically induce a hierarchy by combining phrases extracted from the document/item collection using clustering with context knowledge derived from a knowledge base. Recent efforts have been made to organize text corpora using evolving multi-branch trees [Wang et al., 2013], known as topic trees. Cui et al. [Cui et al., 2014]

present a visual topic analytics system, called RoseRiver, to help users better understand the hierarchical topic evolution at different levels of granularity.

Compared to existing approaches, maintaining an automatically induced hierarchy from Big Data requires simple and greatly scalable methods due to its high volume and velocity of data production. Due to the simplicity and its relation between the processing speed and ability to provide good concept hierarchical relations [De Knijff et al., 2013], the subsumption method is used to learn and maintain the relations between concepts. The proposed approach is the first one to apply a subsumption method in order to maintain concept hierarchy relations (Hierarchy Maintenance) with regards to a stream of unstructured text documents in Big Data context.

6.2/ Hierarchy Relations Maintenance Process

This section describes the hierarchy relation maintenance process that aims to incrementally update the label hierarchy used to multi-classify documents by using a stream of new documents. Label hierarchy is automatically learned based on a Description Logic ontology presented in Table 3.1 (Chapter 3).

In Chapter 4 hierarchy relations (i.e. the hierarchy created using a traditional subsumption method) are induced from a collection of C documents. This chapter proposes the maintenance of hierarchy relations using a stream Stm of new documents. The new documents from Stm are included in collection C originating a new collection C' impacting the term co-occurrence and consequently the hierarchy. In order to persist the co-occurrence values, a term co-occurrence frequency matrix is used to represent the co-occurrence of any pair of terms in a collection of documents as described in Chapter 4.

Two steps comprise the maintenance process (Fig. 6.1):

1. The co-occurrence matrix maintenance step that calculates the impact of new documents in the co-occurrence matrix;
2. The hierarchical relationship maintenance step that calculates how changes in the co-occurrence can matrix impact the hierarchical relations.

These maintenance steps are described in detail in the following sections.

6.3/ Co-occurrence Matrix Maintenance

The co-occurrence matrix used by the subsumption method is impacted with the inclusion of new text documents in the document collection. The set of documents C is a collection of n documents. The sub-set of $documents \subseteq C$ that contains a $term \in Term$ is represented by a set of vectors in the form: $vector_{term} < d_1, d_2, \ldots, d_n >$.

However, in document streaming context, collection C will be constantly evolving. For each new text document doc' in the stream, the collection of documents C will evolve for a C' collection impacting the co-occurrence matrix cfm. Assuming that no new terms are extracted from the new document, the set of terms $term_i$ in the new document doc' is

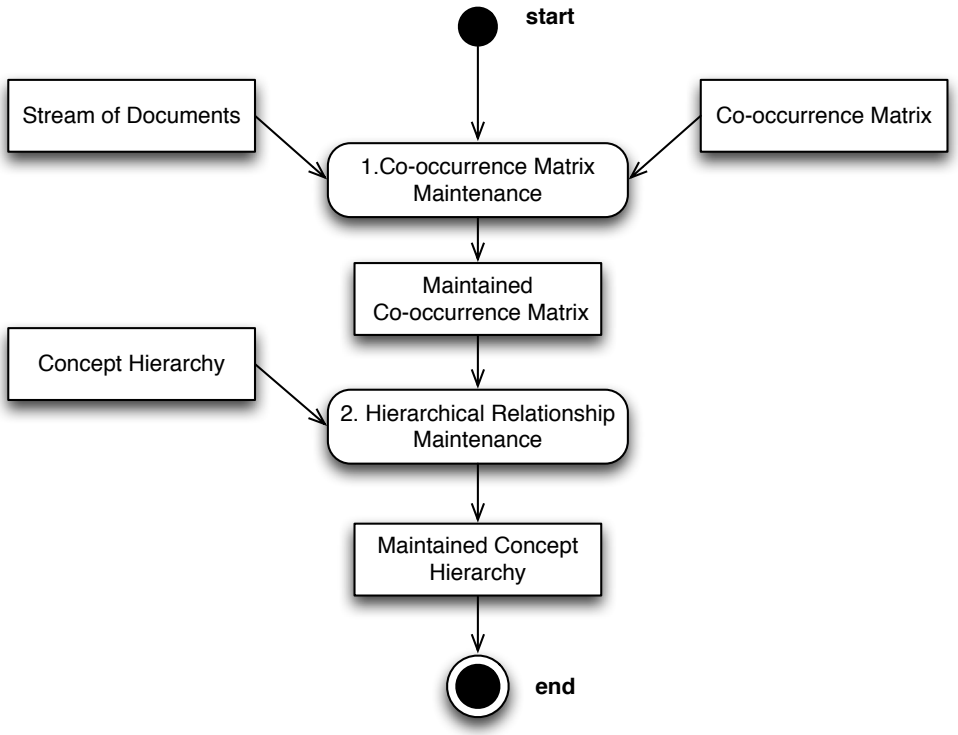


Figure 6.1 – Hierarchy Maintenance Proccess

defined as:

$$setTerms_{doc'} = \{term_i \in Term : term_i \in doc'\} \tag{6.1}$$

All vectors of documents $vector_{term}$ where $term \in setTerms_{doc'}$ are updated by adding the new document doc' such as: $vector_{term} < d_1,\ d_2,\ d_n,\ doc' >$. Hence for each pair of terms $(term_i, term_j) |\ term_i,\ term_j \in setTerms_{doc'}$ the co-occurrence matrix is impacted, such as:

$$cfm_{C'}\left(term_i, term_j\right) = cfm_C\left(term_i, term_j\right) + 1 \tag{6.2}$$

For example, a matrix cfm_C with 3x3 terms from a C collection as depicted in Fig.6.2 (A) is considered. By including a new document doc_1 in collection C, the collection will evolve for C', impacting the co-occurrence matrix.

Let the set of terms for document doc_1 be $setTerms_{doc_1} = \{term_1, term_2\}$, the co-occurrence matrix evolves into $cfm_{C'}$ (Fig.6.2 (B)) according to the previous equation 6.2. In Fig. 6.2 (B) one can observe the increase in one document (+1) of:

— $cfm_C\,(term_1, term_1)$ which is the occurrence of $term_1$ in all documents of collection C;

— $cfm_C\,(term_2, term_2)$ which is the occurrence of $term_2$ in all documents of C;

— $cfm_C\,(term_1, term_2)$ and $cfm_C\,(term_2, term_1)$ which is the number of documents in collection C where $term_1$ and $term_2$ co-occurred.

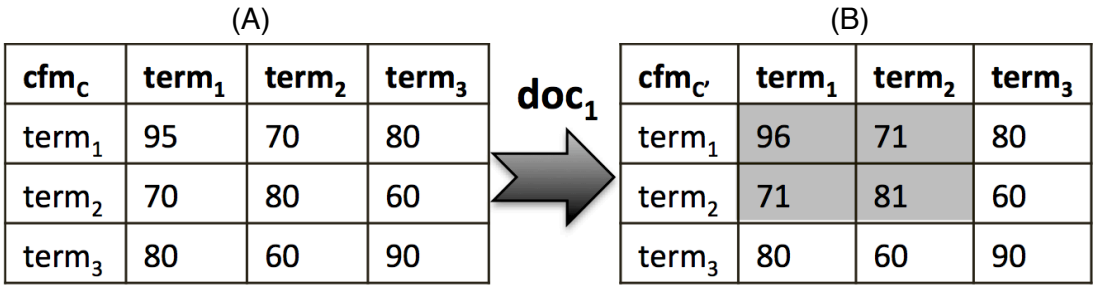


Figure 6.2 – Example of a matrix maintenance evolution

6.4/ HIERARCHICAL RELATIONSHIP MAINTENANCE

Hierarchical relations are maintained according to the evolved co-occurrence matrix without re-processing the entire hierarchy. A method based on the proposal (Chapter 4, Section 4.4) is used. However, for simplicity of explanation, in this chapter only one (instead of two) threshold st is used to calculate hierarchical relations between terms. So, $term_x$ potentially subsumes $term_y$ if:

$$(P_C\left(term_x\,|term_y\,\right) \geq st) \wedge (P_C\left(term_y\,|term_x\,\right) < st) \tag{6.3}$$

where:

— $P_C\left(term_x|term_y\right)$ is the conditional proportion (number) of the documents from collection C common to $term_x$ and $term_y$, in respect to the number of documents in $term_y$ such that:

$$P_C\left(term_x\,|term_y\right) = \frac{term_x\ \cap term_y}{term_y} = \frac{cfm_C\ (term_x\ ,\ term_y)}{cfm_C\left(term_y, term_y\right)} \tag{6.4}$$

— $P_C\left(term_y|term_x\right)$ is the conditional proportion (number) of the documents from collection C common to $term_x$ and $term_y$, in respect to the number of documents in $term_x$ such that:

$$P_C(term_y|term_x) = \frac{term_x \cap term_y}{term_x} = \frac{cfm_C\ (term_x,\ term_y)}{cfm_C\ (term_x, term_x)} \tag{6.5}$$

— $st \in [0, 1]$ is the subsumption co-occurrence threshold.

Hence, if $term_x$ appears in at least a proportion st of the documents in which $term_y$ also appears, and $term_y$ appears in less than a proportion st in which $term_x$ appears, then the hierarchical relationship between x and y is statistically relevant and $term_x$ potentially subsumes $term_y$, i.e. $term_y \prec term_x$. By applying this method to all terms, the result is a subsumption hierarchy of terms. The hierarchy relations can be induced with all

statistically relevant hierarchical relationships (i.e. DAG) or with only some relationships (i.e. Tree).

Since the hierarchy is created from the initial collection of documents C, it must be maintained according to the stream of new documents. Two types of changes in the co-occurrence matrix impact the conditional proportions used to calculate the hierarchal relations: (1) changes in the co-occurrence $cfm_C\left(term_i,\ term_j\right)$ where $term_i \neq term_j$ and (2) changes in the main diagonal $cfm_C\left(term_i,\ term_i\right)$.

6.4.1/ Change in the Co-occurrence

A change in co-occurrences $cfm_C\left(term_i,\ term_j\right)$ where $term_i \neq term_j$, impacts the numerator of the two conditional proportions described by equations (6.4) and (6.5) used by the subsumption method (6.3) so as to calculate hierarchical relations. Hence, only the $P_C(term_i|term_j)$ and $P_C(term_j|term_i)$ proportions used to create the hierarchical relationships between $term_i$ and $term_j$ are impacted.

As an example, consider the set of terms $\{term_1, term_2, term_3, term_4, term_5\}$ with the induced hierarchy relation as depicted in Fig.6.3 (A). A co-occurrence change in the matrix $cfm_C(term_1, term_2)$ will impact the relationship between the terms $term_1$ and $term_2$ (Fig.6.3 (B)).

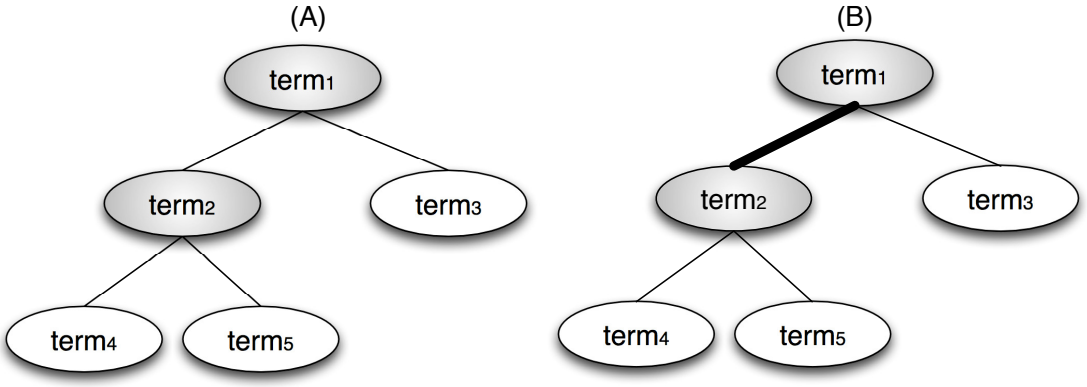


Figure 6.3 – Impact of the hierarchical relationships (example) where (A) is the original hierarchy with a change in the co-occurrence matrix between $term_1$ and $term_2$, and (B) represents the hierarchical relationships impacted by the change in the co-occurrence between $term_1$ and $term_2$.

6.4.2/ Change in the Main Diagonal

A change in the matrix's main diagonal $cfm_C\left(term_i,\ term_i\right)$ to $cfm_{C'}\left(term_i,\ term_i\right)$ will impact the denominator of the two conditional proportions described by equations (6.4) and (6.5), used by the subsumption method (6.3) to calculate hierarchical relations. Therefore, all proportions $P_C(term_i|term_n)$ and $P_C(term_n|term_i)$, where $term_n \in Term$ and $term_n \neq term_i$, are impacted. In other words, all relationships between that term and all other terms from collection C are impacted.

As an example, consider the set of terms $\{term_1, term_2, term_3, term_4, term_5\}$ with the induced hierarchy relationships as depicted in Fig. 6.4 (A). A main diagonal change in the matrix $cfm_C(term_2, term_2)$ impacts the hierarchical relationships with all other terms in the set (Fig. 6.4 (B)). The broken line in Fig.6.4 (B) represents the impacted relation between $term_2$ and $term_3$ even if it is not considered a hierarchical relationship regarding the collection C.

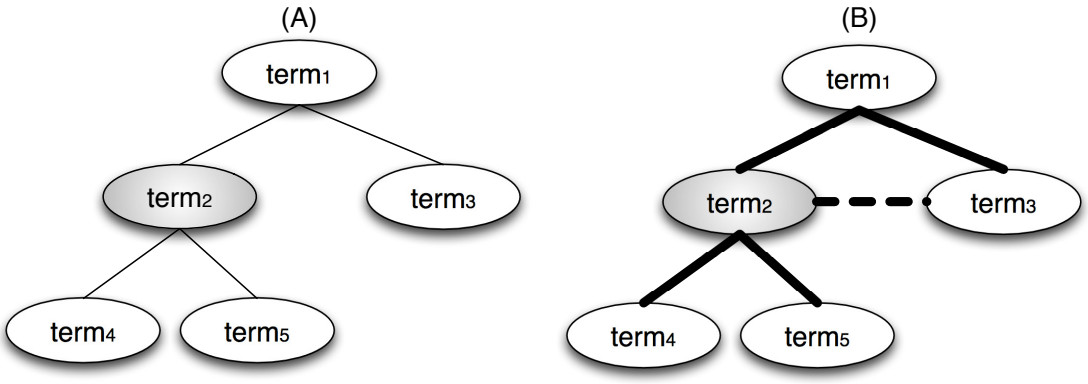


Figure 6.4 – Impact of the hierarchical relationships (example) where (A) is the original hierarchy with change in the co-occurrence matrix main diagonal in $term_2$, and (B) is the hierarchical relationships impacted by the change in the main diagonal in $term_2$.

6.4.3/ MAINTENANCE MODIFICATIONS TO THE HIERARCHY

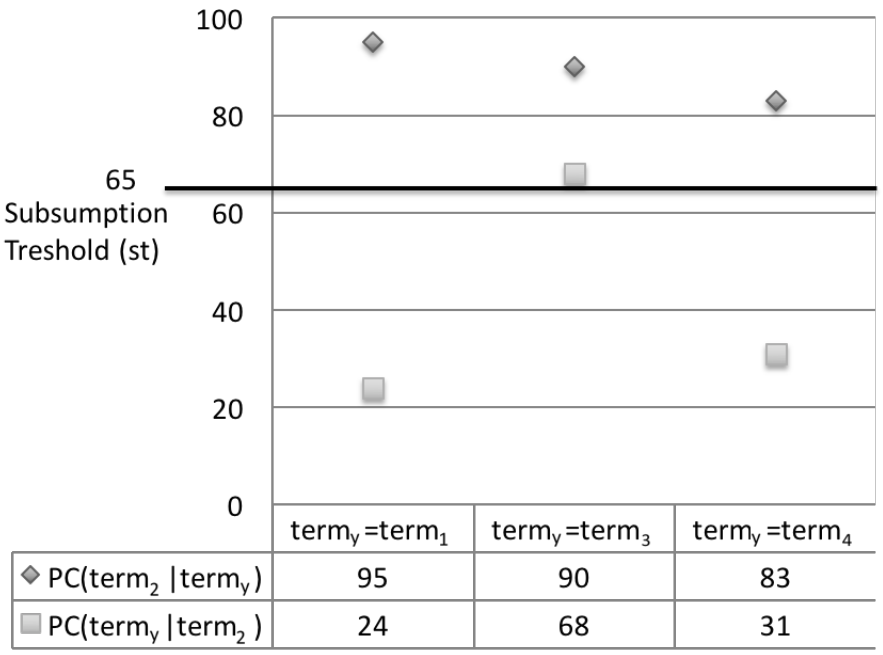
Only the statistically relevant hierarchical relationships obeying the subsumption method are induced as relationships in the output hierarchy. After recalculating all the impacted proportions by C' (i.e. $P_{C'}(term_x|term_y)$ and $P_{C'}(term_y|term_x)$) the maintenance modifications to the hierarchical relationships regarding the previous collection C are derived. A maintenance modification to the hierarchy can be of two types: (i) Add, when a new hierarchy relation is induced and (ii) Delete, when a hierarchy relation is no longer induced.

An Add maintenance modification is derived when a relationship between two terms is not induced as a hierarchy relation in C but is induced in C'. On the other hand, a Delete maintenance modification is derived when a relation is induced as hierarchy relation in C but is not induced in C'.

For example, consider the set of terms $\{term_1, term_2, term_3, term_4\}$ for the initial collection of documents C and that the subsumed terms of t_2 are calculated with a subsumption threshold $st = 65$. The proportions of $term_2$ and all other terms ($term_y$) in collection C as well as the subsumption threshold are depicted in Fig. 6.5.

According to the subsumption method, one can observe in Fig. 6.5 that $term_1$ and $term_4$ are subsumed terms of $term_2$ as $(P_C(term_2|term_1) > st) \wedge (P_C(term_1|term_2) < st)$ and $(P_C(term_2|term_4) > st) \wedge (P_C(term_4|term_2) < st)$. However $term_3$ does not subsume $term_2$ because the subsumption condition with $st = 65$ is not granted $(P_C(term_2|term_4) > st) \wedge (P_C(term_4|term_2) > st)$. Hence the initial hierarchichal relations created for the $term_2$ are $term_1 \prec term_2$ and $term_4 \prec term_2$.

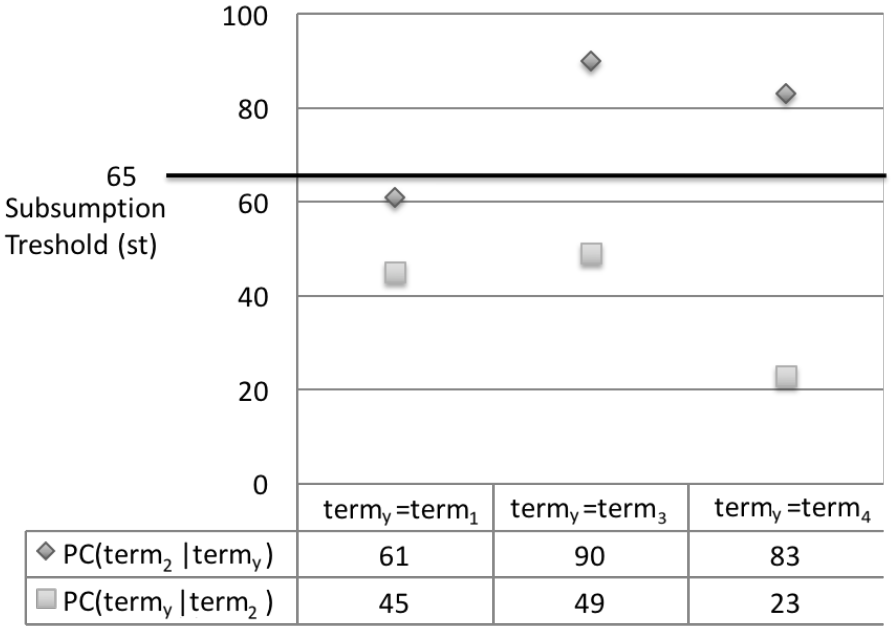
With the arrival of new documents with $term_1$ and $term_2$, collection C will evolve into C'.



	$term_y$ =$term_1$	$term_y$ =$term_3$	$term_y$ =$term_4$
◆ PC($term_2$ |$term_y$)	95	90	83
▫ PC($term_y$ |$term_2$)	24	68	31

Figure 6.5 – Calculate the hierarchical relations between $term_2$ and a $term_y$ in the collection C (example).

Considering the impacted proportions as depicted in Fig. 6.6:



	$term_y$ =$term_1$	$term_y$ =$term_3$	$term_y$ =$term_4$
◆ PC($term_2$ |$term_y$)	61	90	83
▫ PC($term_y$ |$term_2$)	45	49	23

Figure 6.6 – Calculate the hierarchical relations between $term_2$ and a $term_y$ in collection C' (example).

— A Delete maintenance modification to the hierarchy is identified between $term_2$

and $term_1$ where the relation is no longer induced $(P_{C'}(term_2|term_1) < st) \wedge (P_{C'}(term_1|term_2) < st)$;

— An Add maintenance modification to the hierarchy is derived between $term_2$ and $term_3$ where the hierarchical relationship is induced $(P_{C'}(term_2|term_3) > st) \wedge (P_{C'}(term_3|term_2) < st)$.

6.5/ IMPLEMENTATION

This section describes an implementation of the proposed hierarchy maintenance process. The proposed process is implemented in a scalable and distributed platform for Big Data. Hierarchical relationships are maintained using streamed documents that use the distributed and highly scalable broker Apache Kafka [1].

The distributed real-time computation system Apache Storm [2] is used to process text document streaming in order to maintain the hierarchy (Fig. 6.7).

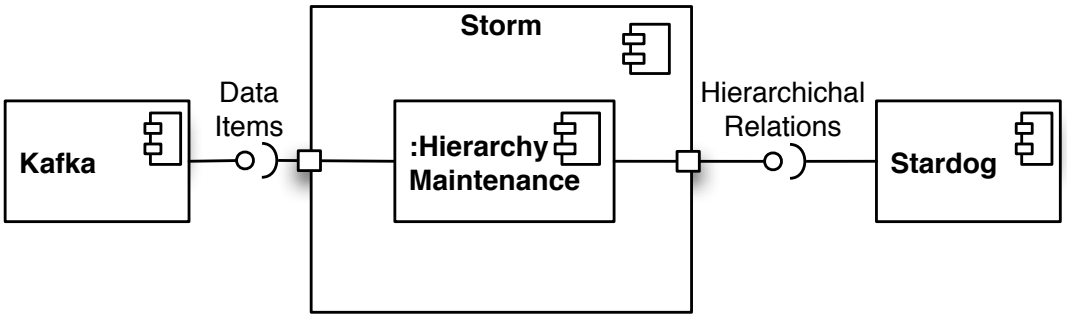


Figure 6.7 – Hierarchy Maintenance Component Diagram (in UML notation)

The Storm's architecture is based on: Tuples (Storm data item abstraction); Stream (unbound list of Tuples); Spout (source of a Stream); Bolts (Process the Streams of Tuples from Spouts to create new Streams) and Topologies (a directed graph of Spouts and Bolts). Four components comprise the maintenance topology (Fig. 6.8): one spout (in square) and three bolts (in ellipsis).

The spout reads the streamed documents from the Kafka broker providing an interface between Storm and Kafka. The first bolt splits the documents in terms using a pre-defined term list. Several methods exist in the literature for extracting the terms from a set of text documents, including linguistic [Hearst, 1992, Toutanova et al., 2000, Cimiano et al., 2003] and statistical approaches [Salton et al., 1988, Maedche et al., 2001]. Obtaining this list is out of the scope of this chapter and will be addressed later in Chapter 7 by using the methods described in Chapter 4, section 4.2.

The second bolt updates the co-occurrence matrix according to Section 6.3. The matrix is stored in a distributed and scalable Big Data store Apache HBase [3]. HBase is a non-

[1] http://kafka.apache.org
[2] https://storm.apache.org
[3] http://hbase.apache.org

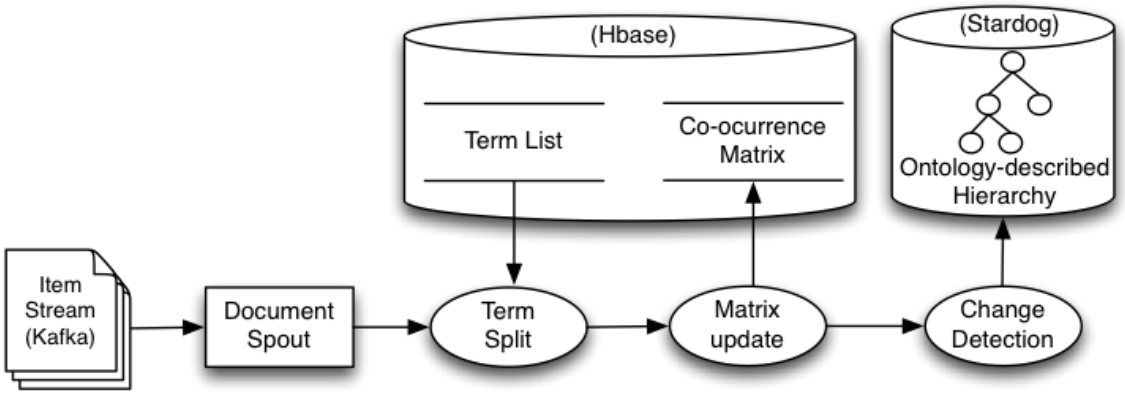


Figure 6.8 – Hierarchy Maintenance Process

relational column-oriented database that is built on top of HDFS (Hadoop Distributed File System). Its column-oriented architecture is a good choice to fit the co-occurrence matrix, as it provides efficient random access to each table cell.

The last bolt derives maintenance modifications to the hierarchy, as described in the previous section. Modification derivation is processed on-the-fly where no proportions are stored. These modifications to the hierarchy can be automatically applied to the hierarchy or recommended to a supervisor.

6.6/ EXPERIMENTS

In this section, the results of the proposed hierarchy maintenance process are discussed. This section focuses only on the number of maintenance modifications (add and delete of hierarchy relations) derived to maintain the initial hierarchy. The subsumption method performance to create relevant relations in the context of hierachichal multi-label classification has been addressed in Chapter 5. Initially, the dataset, the environment and the settings used to test the process are described. Then, the experiments results are presented and discussed.

6.6.1/ TEST ENVIRONEMENT

In order to make maintenance experiments more realistic, the dataset used is extracted from real French economy-related unstructured text documents. The dataset used to evaluate classification performance is composed of 61K text documents written by experts in competitive intelligence, provided by the Company First Eco. From the 61K documents, 45K are used to create the initial label hierarchy and the remaining 16K are used to simulate the stream of documents. Process statistics are monitored for each 4K of documents (4K, 8K, 12K and 16K) and a randomly selected sample view of the hierarchy is compared to the initial state and maintained with 16K. The extracted view is depicted in Fig. 6.9. As the dataset is composed by documents in French, the concepts in Fig. 6.9 are translated into English as accurately as possible.

Some thresholds and settings used in the Semantic HMC process have a high impact on

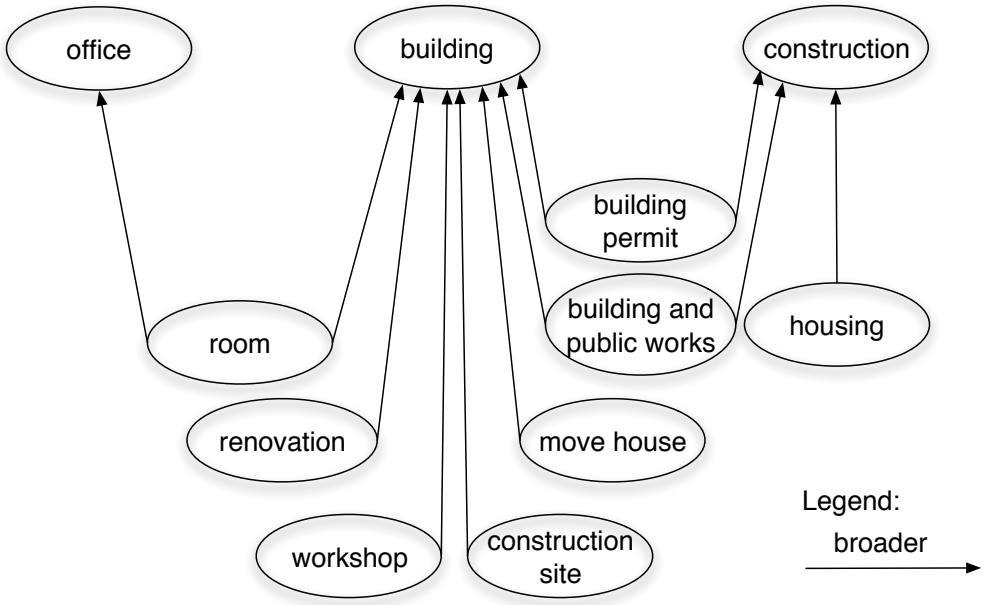


Figure 6.9 – Extracted sample view from the initially learned hierarchy

the results. Table 6.1 shows the different parameters used these tests and their values. The parameters are used to calculate the initial hierarchy and the Subsumption thresholds $st1$ and $st2$ are also used in the maintenance of hierarchy relations. More details about each option can be found in Chapter 4.

Table 6.1 – Process settings

Parameter	Step	Value
Label Derivation Threshold (lT)	Hierarchization	2
Log-Likelihood Ratio	Indexation	4500
Maximum document frequency	Indexation	15
Minimum document frequency	Indexation	1
Minimum support	Indexation	2
n-gram size	Indexation	2
Subsumption st1 Threshold	Hierarchization	20
Subsumption st2 Threshold	Hierarchization	10

The prototype is deployed on a cluster composed of 4 nodes (1 master node and 3 workers) (Fig. 6.10). All nodes are equipped with Intel Core I5 Quad Core processors and 8GB of RAM, except for the master node, which is equipped with 16GB. The Stardog triple-store is deployed on a dedicated server with 8GB of RAM.

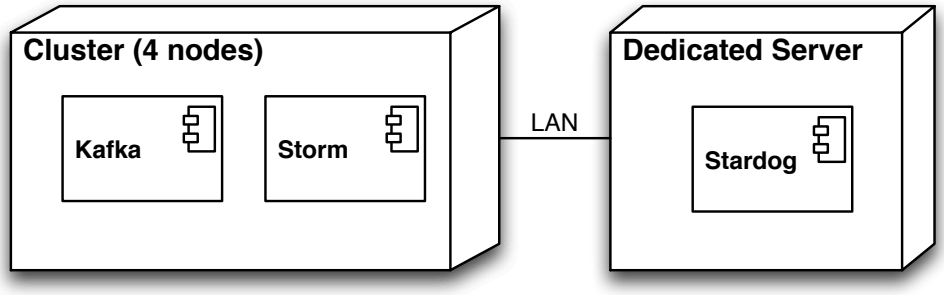


Figure 6.10 – Hierarchy Maintenance Deployment Diagram

6.6.2/ RESULTS

The hierarchy is initially created according to Chapter 4 and then maintained using the sub-data of 16k set-streamed documents using Apache Kafka.

The accumulated number of hierarchy maintenance modifications derived during the stream processing (4K, 8K, 12K and 16K of processed documents) is depicted in Fig. 6.11. An almost linear increase of the number of hierarchical relations deleted can be observed, as well as a slow-down in the addition of relations concerning the deletion, which translates to a decrease of the number of hierarchical relationships.

This decrease of hierarchical relations is in accordance with the results obtained in Chapter 4, where the number of hierarchical relations decrease with the increase of the number of documents considered to create the hierarchy.

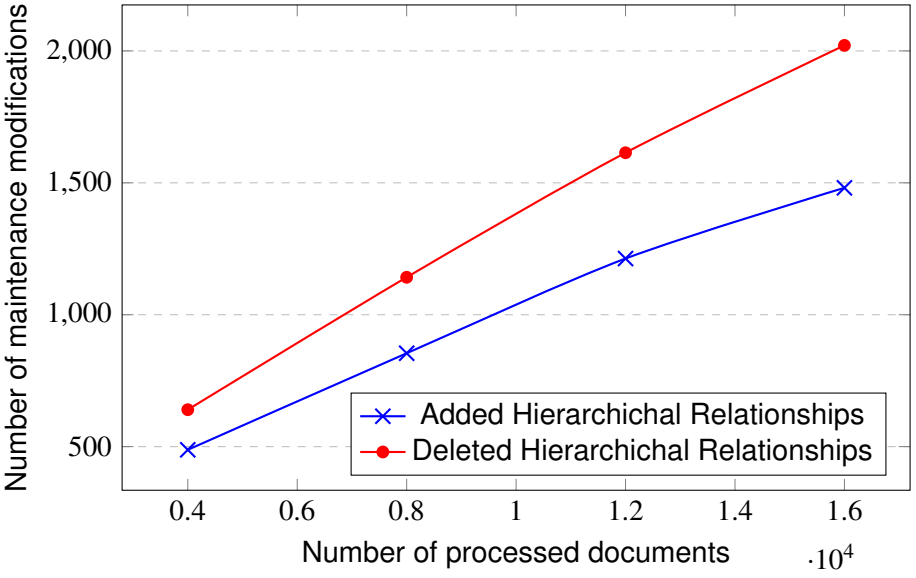


Figure 6.11 – Accumulated number of maintenance modifications to the hierarchy

Table 6.2 presents the number of maintenance modifications done to the hierarchy grouped by each 4k documents (until 4k, 4k to 8k, 8k to 12k and 12k to 16k). A decrease in the number of maintenance modifications done to the hierarchy can be observed as the collection of documents increase. Furthermore, as the thresholds values are fixed

and the collection of documents increases, by observing Fig. 4.8 in Chapter 4, it is observed that the total number of relations decrease as the collection increases. In fact, it is observed that the number of hierarchy relations deleted is superior to the number of new relations, making the total number of Hierarchical relations decrease.

Table 6.2 – Number of maintenance modifications performed to the hierarchy

	0k to 4K	4K to 8k	8K to 12K	12K to 16K
AddHRelationship	488	366	359	268
DeleteHRelationship	640	502	472	407

The extracted sample view of the hierarchy depicted in Fig. 6.9 was maintained according to the 16k document stream. The result, after being maintained with the 16k documents is depicted in Fig. 6.12. Three new hierarchical relationships and one deletion can be observed:

— A deleted relationship between the concept room and office.

— A new relationship between the concept construction materials and construction concept.

— A new relationship between the concept construction materials and product.

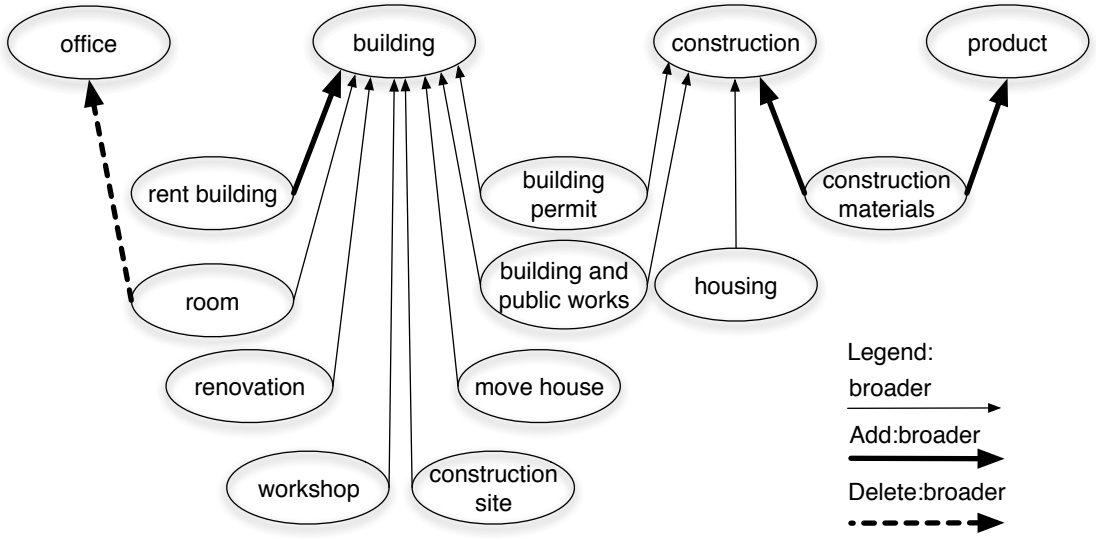


Figure 6.12 – Extracted sample view from the label hierarchy after maintained with 16K

The pertinence and relevancy of using the hierarchical relationship maintenance method in the context of hierarchical multi-label classification is evaluated in Chapter 7.

6.7/ SUMMARY

In order to automatically maintain the hierarchical relationships induced from text documents in Big Data context without relearning the entire hierarchy, this chapter proposes a

maintenance process regarding a stream of text documents. The proposed approach is the first to apply a subsumption method in order to maintain concept hierarchy relationships (Hierarchy Maintenance) with regards to a stream of unstructured text documents in Big Data context. Two steps comprise the maintenance process:

1. The co-occurrence matrix maintenance step that calculates the impact of new documents in the co-occurrence matrix,
2. The hierarchical relationship maintenance step that calculates how changes in the co-occurrence matrix impact the hierarchical relations.

The prototype was successfully implemented and deployed in a scalable and distributed platform to process Big Data. The results show the pertinence of the proposed method to maintain a concept hierarchy regarding a stream of unstructured text documents in Big Data context.

Chapter 7 uses the proposed hierarchy maintenance process in the Semantic HMC process so as to adapt the ontology-described Label hierarchy used to classify documents according to a stream of data in Big Data context.

7

ADAPTIVE LEARNING PROCESS FOR THE ONTOLOGY-DESCRIBED CLASSIFICATION MODEL

Despite the previous Semantic HMC process being able to classify large volumes of unstructutred text documents, the system is unable to adapt to the changes occurring in Big Data. In fact, Big Data not only implies a huge volume of data but also a big velocity stream of data with big variety. The underlying process generating a data stream can be characterized as stationary or nonstationary. In a stationary stream, it is assumed that the data distribution will not change over time, whereas in a nonstationary stream the probabilistic properties of the data will change over time. The learning algorithms that assume a stationary data distribution, in general, produce a classifier that does not change over time, hence being called a static classifier. In many real-world scenarios, however, the data is nonstationary (or evolving or drifting). In nonstationary data distribution, the performance of a non-adaptive classification model trained under the false stationary assumption may degrade the classification performance [Tsymbal, 2004]. On the other hand, adaptive learning refers to updating classification models online during their operation, regarding nonstationary data streams [Gama et al., 2014].

This chapter proposes a new Adaptive Learning process to consistently adapt an Ontology-described classification model used for hierarchical multi-label classification regarding a nonstationary stream of unstructured text data in Big Data context. The process is instantiated for the specific problem of the Semantic HMC process presented in the Part I of this thesis. The Adaptive learning process is implemented for the specific problem of the Semantic HMC according to the goals of Big Data analytics. To that effect, the process is implemented using Big Data technologies to process huge streams of data at high velocity. These technologies distribute stream processing by several machines to reach an high scalability.

The experiments made with the adaptive classification model process for the Semantic HMC are performed by comparing the classification performance with a static classification model as proposed in the second part of this thesis (Chapters: 3, 4 and 5) with the same classification model, but adapted according to the proposed adaptive process.

The remaining of this chapter is structured in six sections. Section 7.1 presents the specific background knowledge and related work in adaptive learning. Section 7.2 proposes a new Adaptive Learning process to consistently adapt an Ontology-described classification model used for hierarchical multi-label classification. Sections 7.3, describes in detail

the adoption of the proposed adaptive process to adapt the classification model of the Semantic HMC process. Section 7.4 shows the implementation of the adaptive process in a scalable and distributed platform. Section 7.5 describes the experiments performed with the adaptive process and discusses the obtained results. Finally, the last section 7.6 summarizes this chapter.

7.1/ Specific Background and Related Work

In this section, the current related work in adaptive learning and ontology evolution is presented and discussed. The first subsection describes some background and related work in adaptive learning regarding the data stream classification problem. The second subsection describes some background and related work in ontology evolution. The last subsection discusses the related work.

7.1.1/ Adaptive Learning

The Data stream classification problem has been widely studied in the literature [Masud et al., 2010, Spinosa et al., 2008, Katakis et al., 2006, Wenerstrom et al., 2006, Wang, 2006, Hulten et al., 2001, Gao et al., 2007, Wang et al., 2003].

Two of the most challenging and well-studied aspects of data streams are their infinite length and concept drift [Gama et al., 2014]. Most of the existing data stream classification techniques are designed to handle these two aspects of the classification process [Masud et al., 2010, Spinosa et al., 2008, Katakis et al., 2006, Wenerstrom et al., 2006, Wang, 2006, Hulten et al., 2001, Gao et al., 2007, Wang et al., 2003].

In addition to infinite length and concept drift aspects, some have also studied concept evolution and feature evolution [Masud et al., 2010, Spinosa et al., 2008, Katakis et al., 2006, Wenerstrom et al., 2006]. Concept Evolution occurs when new classes (labels in Semantic HMC) used in item classification emerge in the underlying stream of text items [Masud et al., 2010], [Spinosa et al., 2008]. In the literature, the feature space is a vocabulary, usually composed of a high number of words, used to describe data items. In data stream, new words that can describe the items must be introduced in the feature space.

7.1.2/ Ontology Evolution

Ontologies inevitably change over time. An ontological change is an action performed over an ontology, whose result is an ontology different from the original version [Klein et al., 2003].

Migrating ontologies to their last versions can generate gaps that must be re-examined and others in which no re-examination is justified. In [Tzitzikas et al., 2013], Tzitzikas et al. formalize and propose principles, techniques and tools for managing uncertainties incurred by such migrations specifically for:

— Automatically identifying descriptions that are candidates for specialization;

— Recommending possible specializations;

— Updating the available descriptions (and their candidate specializations), after the user (curator of the repository) accepts/rejects such recommendations.

More recently, Pittet et al. [Pittet et al., 2014] propose a management methodology of changes for managing the ontology life-cycle called OntoVersionGraph. The methodology includes ontology evolution and versioning features, in conjunction with contextual view modelling.

7.1.3/ Discussion

Diverse applications of adaptive learning exist in literature, including adapting classification models regarding streams of data. Several studies were proposed to detect feature evolution, concept evolution and concept drift. Furthermore, efforts to adapt data mining and machine learning algorithms on distributed stream processing engines were recently published. Nevertheless, current adaptive learning processes do not consider the possibility of using ontologies to describe the learned classification model in Big Data context.

On the other hand, processes for ontology evolution and change management are adequate for general ontologies and consistently handle changes to the ontology. However, these general ontology evolution processes do not capture the specific semantics of a classification model like the one necessary and proposed for adoption in Semantic HMC.

To the extent of our knowledge, this approach is the first to propose an adaptive learning process that consistently and semantically evolves an ontology-described classification model used for hierarchical multi-label classification regarding a nonstationary stream of unstructured text data in Big Data context.

7.2/ Classification Model Adaptive Learning Process

This section proposes a new Adaptive Learning process to consistently adapt an Ontology-described classification model used for hierarchical multi-label classification. The adaptive learning process focuses on adapting the ontology-described classification model regarding a stream of unstructured text documents in Big Data context. Three aspects of data stream processing are addressed in the proposed adaptive learning process:

— **Feature evolution:** occurs when new terms/words (features) emerge from the data stream and old features fade away;

— **Concept evolution:** occurs when new classes/labels that are 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).

The adaptive learning process manages the impact of changes in the classification model according to the specific semantics of the classification process.

7.2.1/ ADAPTIVE PROCESS ONTOLOGY

The adaptive learning process is described by a conceptual model presented in Fig. 7.1. Seven main concepts describe the adaptive learning process:

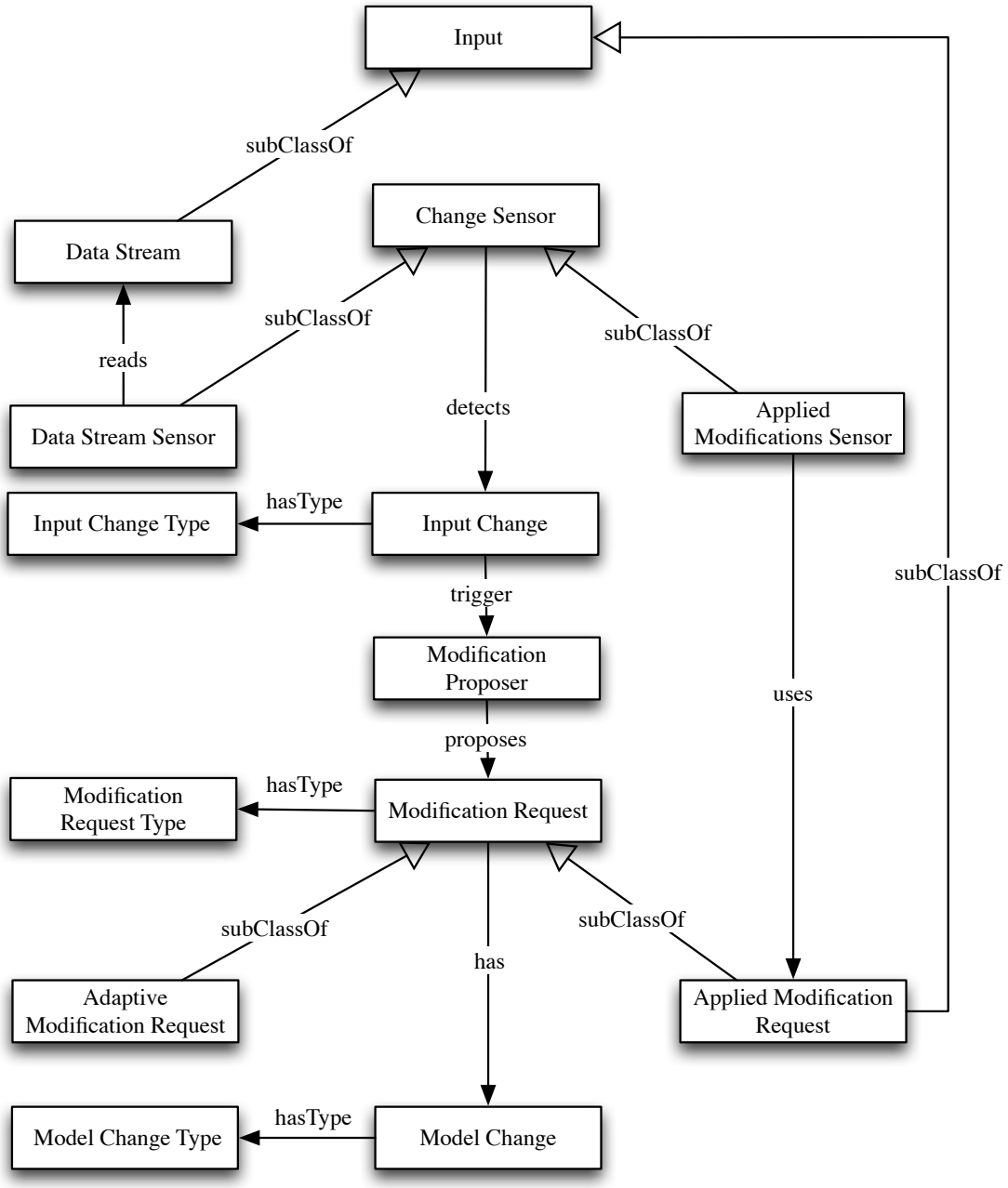


Figure 7.1 – Adaptive learning process conceptual model

— **Input:** Input for the adaptive learning process. I.e. the data stream and the modifications already applied to the classification model.

— **Change Sensor:** Adaptive learning component that detect changes in the inputs. I.e.:

1. the data stream sensor that reads the data stream to detect changes on their data distribution properties
2. the impact change sensor that detects changes performed to the classification model.

— **Input Change:** The change detected in an input by a change sensor. The input change is described by the type of detected change.

— **Modification Proposer:** The adaptive learning component that proposes modifications to adapt the classification model according to input changes.

— **Modification Request:** The modification proposed by a modification proposer to adapt the classification model. The modification request is described by a modification request type and composed of changes to the classification model that together have specific meaning/semantics.

— **Model Change:** The change performed to the ontology-described classification model. The changes are described by a model change type.

The semantic of the maintenance process is captured in the modification request type and model change type classes. I.e. the proposed generic maintenance process is driven/constrained by the types of modification request and model change defined in the adaptive process ontology, hence supporting the configurability of the process.

7.2.2/ CLASSIFICATION MODEL ADAPTIVE PROCESS

The classification model's adaptive process is the set of activities necessary to maintain the classification model in accordance with the data stream, but independent of any particularities of the modification and model change types. The process uses a Single-model incremental approach [Hulten et al., 2001, Wang, 2006] where the classification model is incrementally adapted regarding new data.

Four main steps compose the adaptive process (Fig. 7.2): (1) Input Change Detection, (2) Modification Request Derivation, (3) Model Changes Translation and (4) Classification Model Evolution. The Classification Model Evolution step is composed of four sub-steps: (4.1) Apply Changes, (4.2) Consistency Check, (4.3) Resolve Inconsistencies and (4.4) Modification Commit.

The input change detection step detects the Input Changes according to (i) the Input Data Stream and (ii) the previously applied modifications (i.e. these may cause new modifications requests).

The Modification Request Derivation step derives the modifications requests of the classification model, according to the detected Input Changes. Each Modification Request Type defines the necessary and sufficient conditions (i.e. input changes that must be present) for its instantiation.

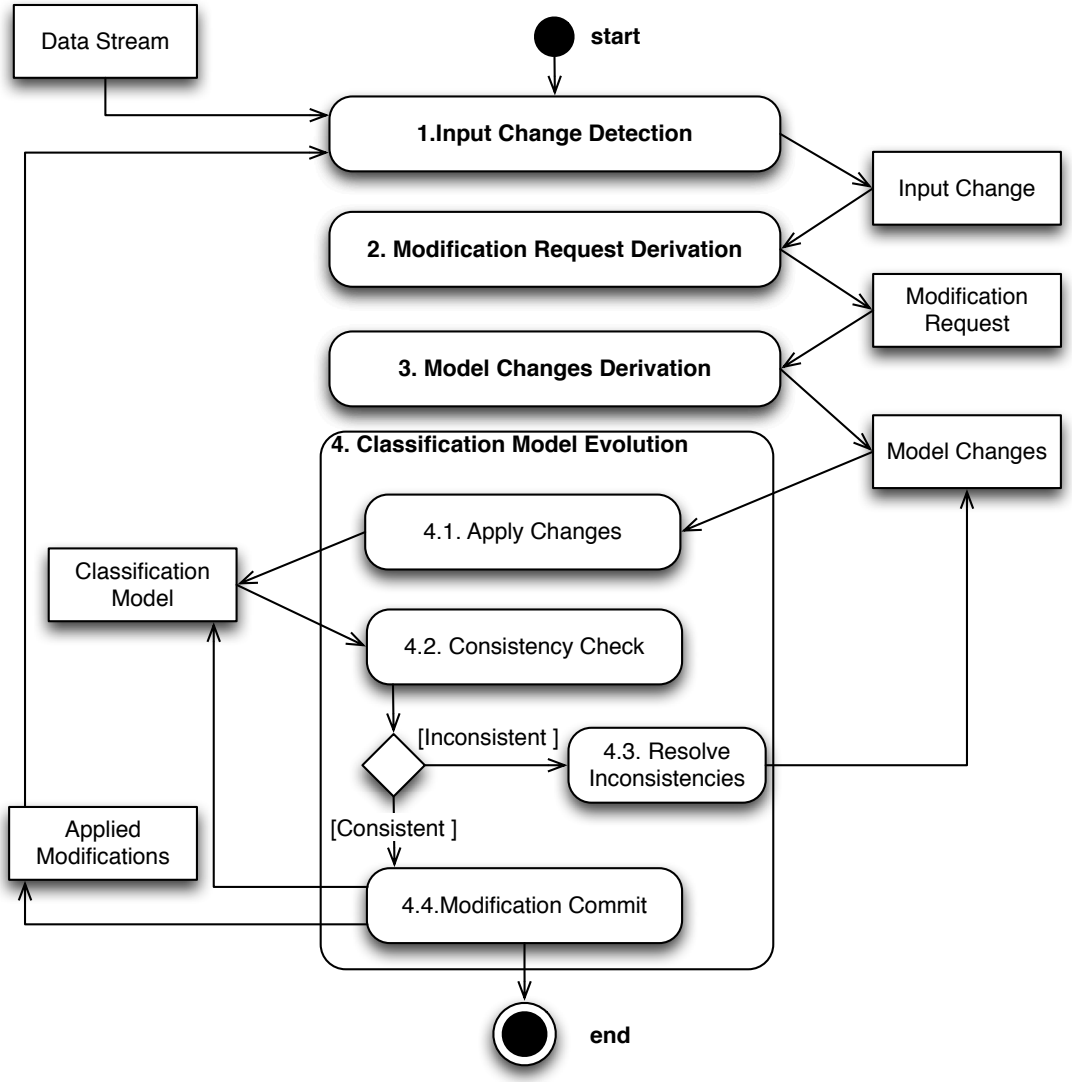


Figure 7.2 – Adaptive Process for an ontology-described classification model

During the Model Changes Translation step, the process analyses the proposed Modification Requests and generates necessary the Model Changes so as to apply the modification request. The type of Modification Request determines the Model Changes. Model changes are either addition or deletion axioms to the ontology-described Classification Model, corresponding to Feature Evolution (i.e. Term evolution), Concept Evolution (i.e. Label evolution) or Concept drift (i.e. Hierarchy and rules evolution).

The last step of the adaptive process (Classification Model Evolution) corresponds to a typical ontology evolution approach proposed in the literature (e.g. [Stojanovic et al., 2002, Klein et al., 2003]) yet considering the specific semantics of the Classification Model. During the Classification Model Evolution step, modification requests (i.e. the set of model changes translated in step 3) are applied to the Classification Model (4.1 Apply Changes). Once applied, an ontology consistency check is executed (4.2 Consistency Check) on the classification model. If inconsistencies are observed in the Classification Model, they are then reviewed by an external actor (i.e. a supervisor)

and changes are deferred until the inconsistencies are resolved (4.3 Resolve Inconsistencies). Otherwise, the changes are committed (4.4 Modification Commit).

7.3/ Adaptive Process Instantiation for the Semantic HMC

The Conceptual Classification Model Adaptive Process proposed in the previous section is adopted/adapted to the Classification Model of the Semantic HMC process described in Part I of this thesis. The Classification Model is described by a DL ontology presented in Table 3.1 of Chapter 3, section 3.1.

The rest of this section describes in detail the specific instantiation of each step of the adaptive process (Input Change Detection, Modification Request Derivation, Model Change Translation and Classification Model Evolution) to adapt the Classification Model of the Semantic HMC process.

7.3.1/ Input Change Detection

Regarding the Specific semantics of the Semantic HMC (i.e. Inverted Index, Co-occurrence Matrix), four Input Change Types are identified (Table 7.1) .

Table 7.1 – Input Change Types

Input Change Type	Description
newDocument	A new document is added to the collection from the Data Stream increasing the Collection.
newTerm	A new term is detected in the Data Stream.
coOcurrence	Document frequency of a term has changed.
appliedModification	Modification Request applied with success to the classification model.

Let $iChange(InputChangeType,\ parameters)$ be the input change composed of an Input Change Type and a list of relevant parameters. To detect the Input Changes, four Input Change Sensors are proposed to detect changes in each input:

— **Document Sensor:** detects whenever a new document from the Data Stream doc' is added to collection C, increasing their cardinality from $|C|$ to $|C'|$ such as: $|C'| = |C| + 1$. Hence, an Input Change $iChange$ of the type CardinalityChange is detected i.e. $iChange(newDocument,\ doc)$.

— **The New Term Sensor:** detects new terms in the Data Stream using the new terms $newTerms_{doc'}$ identified in each new document doc' to update the Inverted Index structure. Hence, an Input Change $iChange$ of the type NewTerm is detected for each $term_d \in newTerms_{doc'}$, i.e. $iChange(newTerm, term_d)$.

— **Co-ocurrence Matrix Frequency Sensor:** detects changes in the co-ocurrence frequency values. The change is detected when the Co-ocurrence Matrix cfm_C

is impacted by the Data Stream as described in equation (2). For each impact $cfm_{C'}\left(term_i, term_j\right)$ an Input Change $iChange$ of the $coOcurrence$ type is detected i.e. $iChange(coOcurrence\ , \{term_i, term_j\})$.

— **Applied Modifications Sensor:** detects new Applied Modification Requests. For each new Applied Modification, an Input Change $iChange$ of the type $appliedModification$ is detected, i.e. $iChange(appliedModification\, mod)$.

7.3.2/ Modification Request Derivation

A modification request captures high-level semantics of the modifications to occur in the classification model (Table 7.2).

Table 7.2 – Modification Request (MR) Types

MR Type	Adaptive Type	Description
AddTerm	Feature Evolution	Add a new term to the Classification Model
AddLabel	Concept Evolution	Add a new Label to the Classification Model
DeleteLabel	Concept Evolution	Delete a Label from the Classification Model
AddHRelation	Concept Drift	Add a Hierarchichal Relation between two Labels to the Classification Model
DeleteHRelation	Concept Drift	Delete a Hierarchichal Relation between two Labels from the Classification Model
AddAlphaTerm	Concept Drift	Add a Alpha Relation between a Term and a Label to the Classification Model
DeleteAlphaTerm	Concept Drift	Delete a Alpha Relation between a Term and a Label from the Classification Model
AddBetaTerm	Concept Drift	Add a Beta Relation between a Term and a Label to the Classification Model
DeleteBetaTerm	Concept Drift	Delete a Beta Relation between a Term and a Label from the Classification Model

Each modification request type defines the necessary and sufficient conditions for its instantiation:

— **AddTerm:** A $newTerm$ input change is detected.

— **AddLabel:** An $iChange$ is detected that is either:

- $iChange(appliedModification,\ mod)$ such that $mod \in AddTerm$.
- $iChange(newDocument, doc)$ and $P_c(term_i) \geq lT$ where (i) $P_c(term_i)$ is the proportion of items in a collection C in which the $term_i$ appears, (ii) lT is the label threshold, (iii) $term_i \in doc$ and (iv) $term_i \notin Label$.

— **DeleteLabel:** An $iChange$ is detected that:

- $iChange(newDocument, doc)$ and $P_c(term_i) < lT$ where (i) $P_c(term_i)$ is the proportion of items in a collection C in which the $term_i \in doc$ appears, (ii) lT is the label threshold and (iii) $term_i \in Label$.

— **AddHRelation:** An $iChange$ is detected that is either:

- $iChange(appliedModification, mod)$ such that $mod \in AddLabel$.
- $iChange(coOcurrence, \{term_i, term_j\})$ and by applying the Hierarchichal Relation Change Detection approach described in Chapter 6 for $term_i$ and $term_j$ detects an "Add" change that represents the creation of a Hierarchical Relation between $term_i \in Label$ and $term_j \in Label$.

— **DeleteHRelation:** An $iChange$ is detected that is either:

- $iChange(appliedModification, mod)$ such that $mod \in DeleteLabel$.
- $iChange(coOcurrence, \{term_i, term_j\})$ and by applying the Hierarchichal Relation Change Detection approach described in Chapter 6 for $term_i$ and $term_j$ detects a "Delete" change that represents the deletion of a Hierarchical Relation between $term_i \in Label$ and $term_j \in Label$.

— **AddAlphaTerm:** An $iChange$ is detected that is either:

- $iChange(appliedModification, mod)$ such that $mod \in AddLabel$ and $\exists term_j \in Term : P_{C'}\left(term_j | term_i\right) > \alpha$ where (i) α is the Alpha threshold and (ii) $term_i \in mod$.
- $iChange(coOcurrence, \{term_i, term_j\})$ and $P_{C'}\left(term_j | term_i\right) > \alpha$ where $term_j \in Term$ is not an Alpha term of $term_i \in Label$.

— **DeleteAlphaTerm:** An $iChange$ is detected that is either:

- $iChange(appliedModification, mod)$ such that $mod \in DeleteLabel$ and exist a $term_j \in Term$ that is an Alpha term of the $term_i \in mod$.
- $iChange(coOcurrence, \{term_i, term_j\})$ and $P_{C'}\left(term_j | term_i\right) \leq \alpha$ where $term_j \in Term$ is an Alpha term of $term_i \in Label$.

— **AddBetaTerm:** An $iChange$ is detected that is either:

- $iChange(appliedModification, mod)$ such that $mod \in AddLabel$ and $\exists term_j \in Term : \beta \leq P_{C'}\left(term_j | term_i\right) \leq \alpha$ where (i) β is the Beta threshold and (ii) $term_i \in mod$.
- $iChange(coOcurrence, \{term_i, term_j\})$ and $\beta \leq P_{C'}\left(term_j | term_i\right) \leq \alpha$ where $term_j \in Term$ is not a Beta term of $term_i \in Label$.

— **DeleteBetaTerm:** An $iChange$ is detected that is either:

- $iChange(appliedModification,\ mod)$ such that $mod \in DeleteLabel$ and exist a $term_j \in Term$ that is a Beta term of the $term_i \in mod$.
- $iChange(coOcurrence, \{term_i, term_j\})$ and $P_{C'}\left(term_j \middle| term_i\right) < \beta$ or $P_{C'}\left(term_j \middle| term_i\right) > \alpha$ where $term_j \in Term$ is a Beta term of $term_i \in Label$.

7.3.3/ Model Change Translation

Based on the derived modification requests, this step generates the classification model changes, i.e. a set of ontology-evolution changes that, once applied, correspond to the modification.

The first column of Table 7.3 shows the Description Logics syntax for the construction of axioms and facts of a knowledge base and the second column shows the basic changes description in an abstract syntax: classes (C), data types (T), relations (R), attributes (A), instances (I) and datavalues (V). Each change has an associated operation parameter $Operation = \{Add,\ Delete\}$ where Delete is the inverse operation of Addition (Add).

Table 7.3 – Model Changes By Type

DL Syntax	Basic changes abstract syntax
I	Instance(Operation, Instance)
$I \in C_i$	InstancesOf(Operation, Instance, Class)
$\{I_1,\ I_2\} \in R_i$	InstancesOfObjectProperty(Operation, Instance1, Instance2, ObjectProperty)
$\{I,\ V_i\} \in A_i$	InstanceOfDatatypeProperty(Operation, Instance, Datavalue, DatatypeProperty)

Therefore, each modification type defines the Classification Model changes to apply:

— **AddTerm:** Considering the $termString$ as the word (string) that originates the new term to include in the Classification Model, the Model Changes to apply are:

- $Instance(Add,\ term)$
- $InstancesOf(Add,\ term,\ Term)$, i.e. $term \in Term$
- $InstanceOfDatatypeProperty(Add,\ term,\ "termString",\ asString)$

— **AddLabel:** Considering that $term \in Term$ originates the Label to be included in the Classification Model, the Model Change to apply is:

- $InstancesOf(Add,\ term,\ Label)$

— **DeleteLabel:** Considering that $term \in Term$ originates the Label to be deleted from the Classification Model, the Model Change to apply is:

- $InstancesOf(Delete,\ term,\ Label)$

— **AddHRelation:** Considering $label1 \in Label$ as the broader label and $label2 \in Label$ as the narrower label to be included in the Classification Model, the Model Changes to apply are either:

- $InstancesOfObjectProperty(Add,\ label1,\ label2,\ broader)$
- $InstancesOfObjectProperty(Add,\ label2,\ label1,\ narrower)$

— **DeleteHRelation:** Considering $label1 \in Label$ as the broader label and $label2 \in Label$ as the narrower label to be deleted from the Classification Model, the Model Changes to apply are:

- $InstancesOfObjectProperty(Delete,\ label1,\ label2,\ broader)$
- $InstancesOfObjectProperty\ (Delete,\ label2,\ label1,\ narrower)$

— **AddAlphaTerm:** Considering $term \in Term$ as the relevant term to be the Alpha term of the label $label \in Label$, the Model Change to apply is:

- $InstancesOfObjectProperty(Add,\ label,\ term,\ hasAlpha)$

— **DeleteAlphaTerm:** Considering $term \in Term$ as the term that is no longer relevant to be the Alpha term of the label $label \in Label$, the Model Change to apply is:

- $InstancesOfObjectProperty(Delete,\ label,\ term,\ hasAlpha)$

— **AddBetaTerm:** Considering $term \in Term$ as the relevant term to be classified as Beta term of the label $label \in Label$, the Model Change to apply is:

- $InstancesOfObjectProperty(Add,\ label,\ term,\ hasBeta)$

— **DeleteBetaTerm:** Considering $term \in Term$ as the term that is no longer relevant to be the Beta term of the label $label \in Label$, the model change to apply is:

- $InstancesOfObjectProperty(Delete,\ label,\ term,\ hasBeta)$

7.3.4/ CLASSIFICATION MODEL EVOLUTION

The Model Changes translated in previous step are applied to the Classification Model making the Classification Model evolve. Because the Classification Model is described by an ontology, the evolution process will adopt/adapt a state of the art ontology evolution approach. Based on the elementary evolution phases proposed in [Stojanovic et al., 2002], four conceptual sub-steps are proposed: Apply Changes, Consistency Check, Resolve Inconsistencies and Modification Commit.

— **Apply Changes:** The Model Changes translated in step 3 (Model Changes Translation) are applied to the Classification Model. Instead of applying the Model Changes to the Classification Model used by the Semantic HMC, they are applied to a copy of it. This avoids turning the Classification Model incompatible or inaccessible to the Semantic HMC process before a final version is committed.

— **Consistency Check:** Once the Model Changes are applied to the Classification Model, a consistency check is performed by an ontology reasoner. If inconsistencies are detected, a new Change Modeling phase takes place.

— **Resolve Inconsistencies:** If inconsistencies are detected, new additional changes to the Classification Model are proposed in order to resolve the inconsistencies. The proposal of new additional changes is a human-based process.

— **Modification Commit:** If the adapted classification model is consistent, the changes are then committed and available to be used by the Semantic HMC process. Moreover, the Modification Request is classified as an applied modification and exploited in step 1 (i.e. Input Change Detection).

7.4/ ADAPTIVE PROCESS IMPLEMENTATION

This section describes the implementation of the adaptive process proposed in previous sections. The process is implemented as a Java application, because many libraries that support the proposed methods are developed in Java. Across all steps of the proposed adaptive process, Big Data processing techniques are used.

Each step of the adaptive process (Fig. 7.3) (i.e. Input Change detection, Modification Request Derivation, Model Changes Translation or Classification Model Evolution) is described by a directed graph of nodes that communicate via messages (i.e. Topology). The input text documents and the input/output data of each step (i.e. Input change, Modification Request, Model Changes and Applied Modifications) is represented as a distributed queue, which allows the replication of data across multiple machines. The ontology-described Classification Model is stored in a scalable triple store.

The data queues are handled by the distributed and highly scalable broker Apache Kafka[1]. Each queue is a Kafka topic that is distributed among the Kafka cluster (Fig. 2.9 of Chapter 2). The distributed real-time computation system Apache Storm[2] is used to handle each step of the adaptive learning process, providing distributed stream partition among the processing nodes. The Storm architecture is based on: Tuples (Storm data item abstraction); Stream (unbound list of Tuples); Spout (source of a Stream); Bolts (Process the Streams of Tuples from Spouts to create new Streams) and Topologies (a directed graph of Spouts and Bolts).

A scalable triple store called Stardog[3] is used to store and manage the ontology-described classification model. The specific data used in the Semantic HMC process is stored in a distributed and scalable Big Data store called Apache HBase[4]. HBase is a non-relational column-oriented database that is built on top of HDFS (Hadoop Distributed

[1] http://kafka.apache.org
[2] https://storm.apache.org
[3] http://docs.stardog.com
[4] http://hbase.apache.org

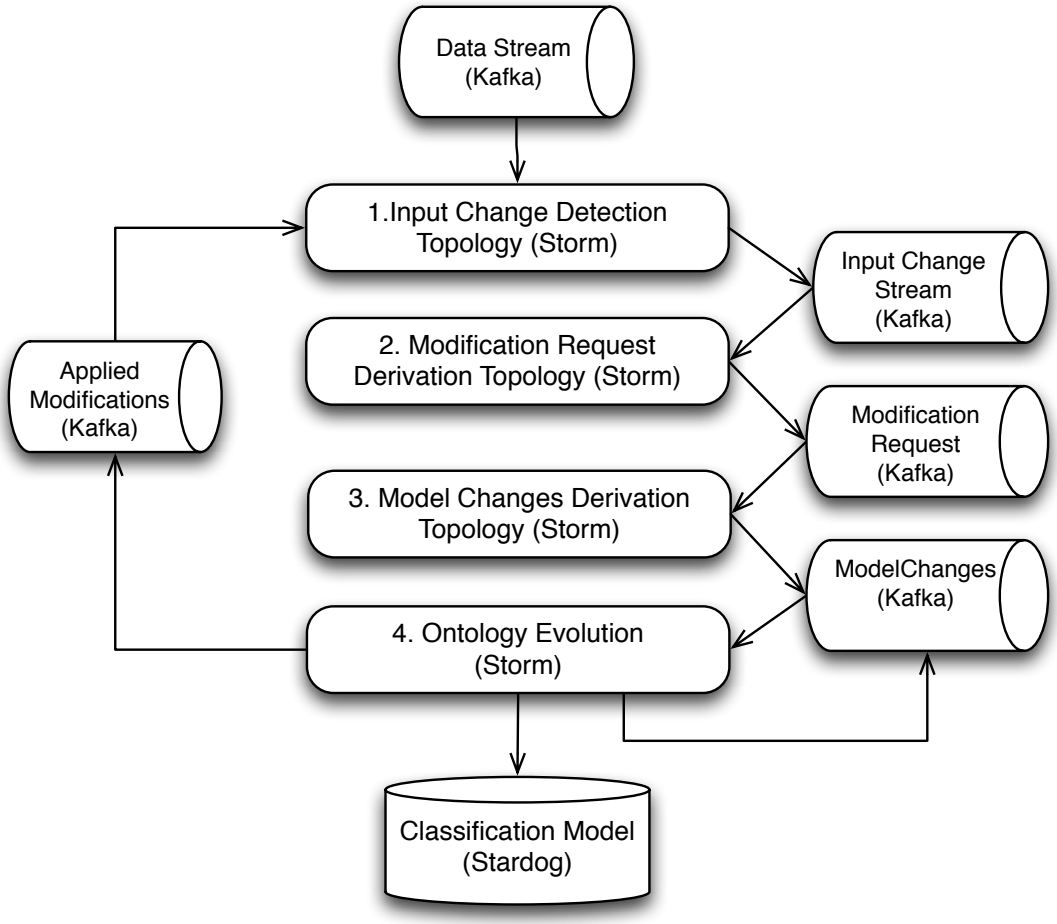


Figure 7.3 – Ontology-described Classification Model adaptive learning process implementation

File System). More details about the adaptive process data storage can be found in Appendix 1 Section A.1.

7.4.1/ Input Change Detection

In order to detect input changes regarding the specific semantics of the Semantic HMC (i.e. Inverted Index, Co-occurrence Matrix), a Storm topology with two spouts (Document Spout and Applied Modifications Spout) and seven bolts (Term Detection, Matrix Update, New Term Sensor, Matrix Frequency Sensor, New Document Sensor and Applied Modification Sensor) are proposed as depicted in Fig. 7.4.

The two input streams are captured in two brokers (i.e. Kafka topics): the Data Stream broker that contains the stream of unstructured text documents, and the Applied Modifications broker that contains the modifications already applied to the Classification Model. The documents from the Document Spout broker are retrieved for processing by the Document Spout in the Storm topology where modification requests are retrieved by the applied Modifications Spout.

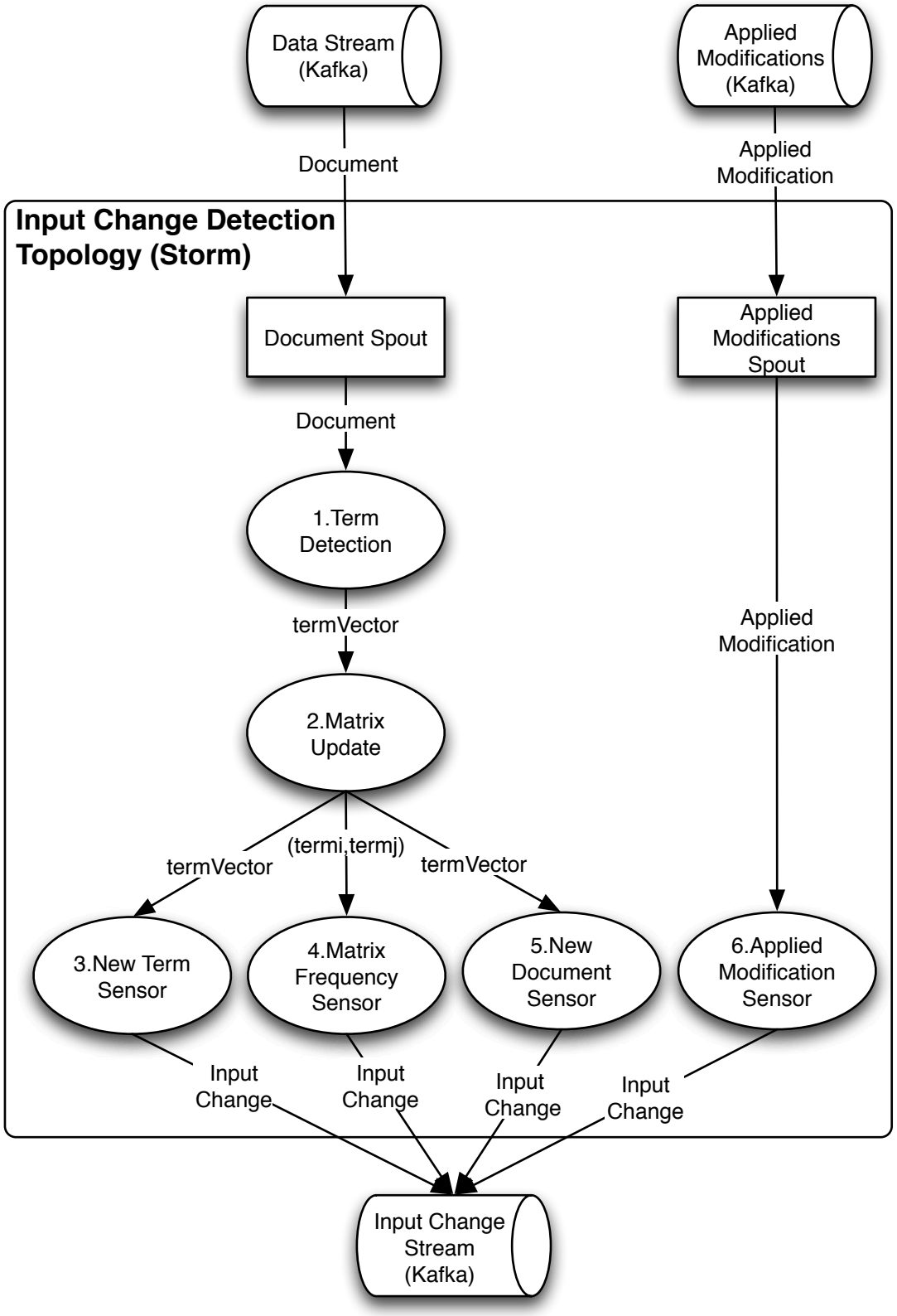


Figure 7.4 – Input Change Detection Topology

For each document retrieved from the Data Stream by the Document Spout, the term detection bolt detects the relevant words to describe that document (terms). The matrix update bolt updates the co-occurrence matrix according to each new document doc' emitted from the term detection bolt as proposed in Chapter 6, section 6.3.

More details about the implementation of Term Detection and Matrix Update can be found in the Appendix 1 Section A.2.

Based on the described bolts (Term Detection, Matrix Update) and the Applied Modifications spout, Input Change Sensors create the input changes and send them to the Input Change broker (i.e. Kafka topic). Four Input Change Sensors are proposed:

— **Matrix Frequency sensor:** The Matrix Frequency sensor outputs an InputChange of the type coOccurrence for each change in the co-ocurrency matrix performed by the Matrix Update. The Input Data is the combination of terms $(term_i, term_j)$ emitted by the Matrix Update bolt.

— **New Term sensor:** The New Term sensor receives the term vectors emitted by the matrix update bolt and verifies if the term already exists in the classification model. To perform this verification, a cache table in HBase called Terms that has an updated list of terms is queried. The Terms table allows lower latency requests to the updated list of Terms, rather than querying the Stardog triple store. For each new term (that doesn't exist in the cache table) a new InputChange of the type newTerm is created, in which the inputData is the new term.

— **New Document sensor:** The New Document sensor receives the term vectors emitted by the matrix update bolt. For each received term vector, this bolt creates and outputs a new InputChange of the type newDocument in which the Input Data is the received term vector.

— **Applied Modification sensor:** The Applied Modification sensor receives the modifications applied to the classification model from the applied modifications broker. For each applied modification received, this bolt creates and outputs a new InputChange of the type appliedModification in which the inputData is the applied modification.

7.4.2/ Modification Request Derivation

The modification request derivation topology gathers the modifications requests to be applied to the classification model, according to the Input Changes detected in the Input Change Topology. This topology (Fig. 7.5) consists of the spout Input Change spout that reads the input changes from the input change broker and the Modification Request Derivator bolt to derive the modifications requests based on the necessary and sufficient rules defined in section 7.3.2. The derived Modification Requests are sent to a new broker.

More details about the implementation of the Modification Request Derivation Bolt as well as the algorithms used to derive the Modification requests can be found in Appendix 1 Section A.3.

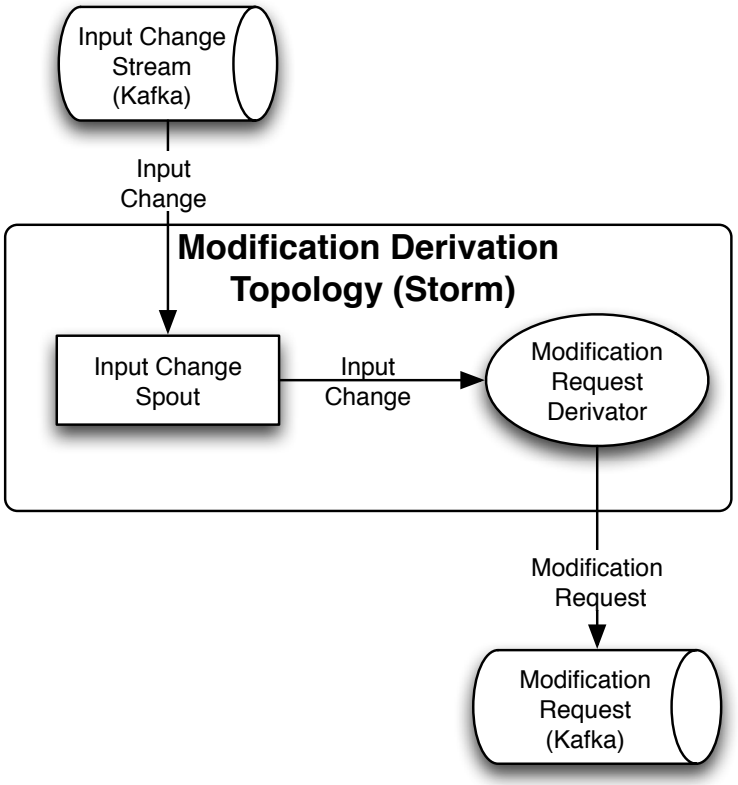


Figure 7.5 – Modification Request Derivation Topology

7.4.3/ Model Change Translation

Based on the derived Modification Requests and outputted by the Modification Derivation topology, the Model Changes translation topology generates Classification Model changes, i.e. a set of ontology-evolution changes that once applied correspond to the modification. The Model Changes Translation Topology (Fig. 7.6) consists of the Modification Request spout that reads the derived Modification Requests from the Modification Request broker and the Model Changes Translator bolt so as to translate the Modification Request into Model Changes, based on the necessary and sufficient conditions defined in section 7.3.3. The Translated Model Changes are sent to a new broker (Model Changes).

7.4.4/ Classification Model Evolution

To evolve the ontology-described classification model according to the translated model changes, four conceptual sub-steps are proposed in the previous section: Apply Changes, Consistency Check, Resolve Inconsistencies and Modification Commit. To that effect, the proposed Classification Model Evolution Topology (Fig. 7.7) is comprised of one spout (Model Change Spout) and three bolts (Apply Changes and Consistency Check, Resolve Inconsistencies and Modification Commit). The model change spout reads the Model Changes for a Modification Request from the Model Change Broker. The Model Changes are then applied to a temporary version of the Classification Model

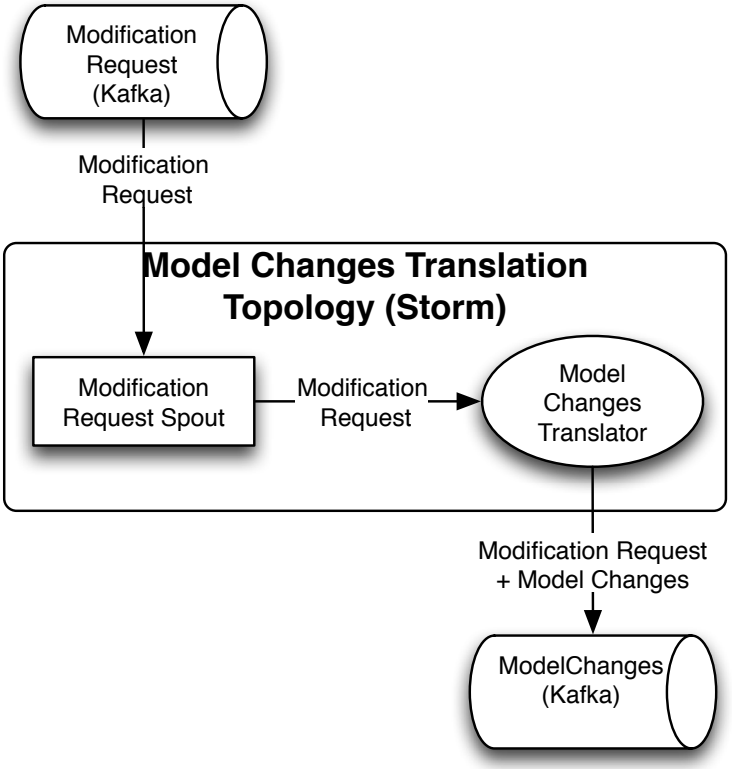


Figure 7.6 – Model Changes Translation Topology

and verified in case any inconsistency exists. In the case of inconsistency, the Modification Request is issued to the Resolve Inconsistencies bolt where the inconsistency is resolved through human interaction. It must propose new changes in order to solve the inconsistency and successfully apply the Modification Request. When no inconsistency is detected, the modification request is committed to the Classification Model in production and emitted to the broker of applied modifications. Notice that the Applied Modification broker is the same as the one used in the Input Change Detection Topology (Fig. 7.4).

7.5/ EXPERIMENTS

The experiments conducted over the adaptive classification model process for the Semantic HMC are made by comparing the classification performance at time t_i of (1) a static classification model created at time t_1 (baseline) and (2) a classification model created at time t_1 but adapted by the adaptive process into t_i.

The next subsections describe the test environment, and discuss the obtained results.

7.5.1/ TEST ENVIRONMENT

The dataset used in this experiment is composed ok 61K text documents written by experts in competitive intelligence, provided by the Company Actualis SARL.This dataset

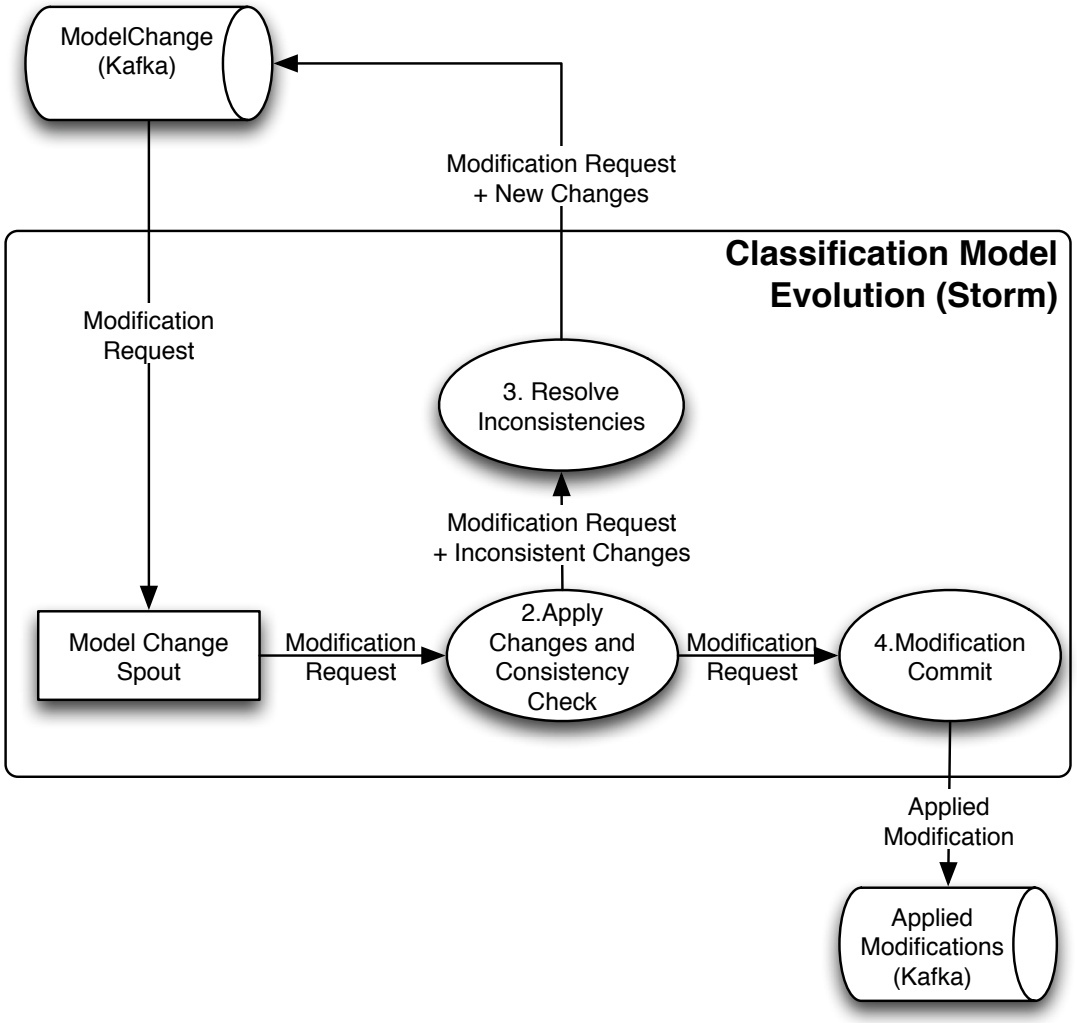


Figure 7.7 – Classification Model Evolution Topology

represents a nonstationary stream of unstructured text corpus from a French company where statistical properties change over time. The use of this dataset was decided instead of a bigger stream (e.g. Wikipedia dataset used in previous tests where the items are dispersed), in order to evaluate the quality of the classification model evolution and not just the velocity performance of the system. From the 61K documents, 45K are used to train the initial classification model (baseline) and the remaining 16K are used to stream and test the classification model. The process statistics as well as the precision and recall are monitored for each 4K of documents (4K, 8K, 12K and 16K).

Some thresholds and settings used in the Semantic HMC process have a high impact on the results. More details about each option can be found in Chapter 4 and Chapter 5. Table 7.4 shows the different parameters and their values that were used in the experiments. The values used in the experiments are empirical and a study on determining/optimizing these values must be done in the future.

The prototype is deployed on a cluster composed of 4 nodes (1 master node and 3 workers). All nodes are equipped with Intel Core I5 Quad Core processors and 8GB of RAM,

Table 7.4 – Semantic HMC settings

Parameter:	Step	Value
Log-Likelihood Ratio	Indexation	4500
Maximum document frequency	Indexation	15
Minimum document frequency	Indexation	1
Minimum support	Indexation	2
n-gram size	Indexation	2
Label Detection Threshold (lT)	Hierarchization	2
Subsumption $st1$ Threshold	Hierarchization	20
Subsumption $st2$ Threshold	Hierarchization	10
Alpha Threshold	Resolution	50
Beta Threshold	Resolution	30
p	Resolution	0.25
Term ranking (n)	Resolution	5
Term Threshold (y)	Realization	2

except for the master node, which is equipped with 16GB. The Stardog triple store is deployed on a dedicated server with an Intel Core I5 Quad Core and 8GB of RAM.

7.5.2/ Classification Quality Experiments

In order to compare the classification performance across the experiments, a label-based metric with micro and macro-averaged precision, recall and F1 measure are used as described in Section 2.3.5.

Unlike the Delicious dataset used to evaluate Semantic HMC and compare to the state of the art in Section 5.5.2, the dataset from real data used in this chapter is not pre-labelled and no list of pre-defined labels exist. Hence, it is not possible to directly compare the classification results to the pre-classified data (validation set) so as to calculate the precision, recall and F1.

However, as the labels are a subset of terms, it is possible to verify if the document has the term that originates that label, then being able to calculate the precision and the recall for that label. For example, if an item is classified with label L_1 and the document has the term T_1 where T_1 is the term that originates the label L_1 then it is correctly classified. In this case, in order to retrieve the classifications, label L_1 and all terms T_1 that originate the label L_1 are removed from the training set.

The classification performance of the classification model is compared by using Precision and Recall upon the adaptation (i.e. evolution) of three dimensions of the classification model (concept drift, feature evolution and concept evolution).

On the one hand, the number of input changes detected during stream processing (4K, 8K, 12K and 16K of processed documents) is presented in Table 7.5. The reader can observe that the number of changes detected per document is 1.6K on average, in which the number of changes in the co-occurrence matrix represents the major percentage of

detected changes.

Table 7.5 – Cumulative number of Input Changes

	4K	8K	12K	16K
AppliedModification	81352	111808	142668	170837
CoOcurrence	6599137	12956409	19366645	26077976
NewDocument	4000	8000	12000	16147
NewTerm	2369	4632	6767	8833
TOTAL:	6686856	13080847	19528077	26273790

On the other hand, the number of Derived Modification Request (Table 7.6) per document (10.5 in average) is much smaller that the number of detected changes. Most of the Modification Requests done to the model are changes to the Alpha and Beta terms. This means that a huge percentage of the changes applied to the Classification Model are changes to the Classification Rules.

Table 7.6 – Number of derived modification requests (cumulative and by interval)

	4K	8K	12K	16K	4K-8K	8K-12K	12K-16K
AddAlphaTerm	47833	56995	65821	73511	9162	8826	7690
AddBetaTerm	18842	26353	34394	41512	7511	8041	7118
AddHRelation	488	854	1213	1481	366	359	268
AddLabel	61	114	156	219	53	42	63
AddTerm	2369	4632	6766	8833	2263	2134	2067
DeleteAlphaTerm	4061	8068	12074	16069	4007	4006	3995
DeleteBetaTerm	6979	13506	20433	26913	6527	6927	6480
DeleteHRelation	640	1142	1614	2021	502	472	407
DeleteLabel	79	144	197	278	65	53	81
TOTAL:	81352	111808	142668	170837	14115	15405	14736

The total number of concept instances in the classification model are presented in Table 7.7. The following subsections describe the experiments in detail for each one of the

Table 7.7 – Number of concept instances in the classification model

	Baseline	4K	8K	12K	16K
Number of Classification Rules	51503	94475	99203	103390	105710
Number of Hierarchichal relations	1882	1730	1594	1481	1342
Number of Labels	550	532	520	509	491
Number of Terms (features)	58060	60429	62692	64826	66893

three dimensions of the classification model (concept drift, feature evolution and concept evolution).

7.5.3/ Adapt to Concept drift Experiments

We evaluate the concept drift performance by comparing the classification model with static feature space (i.e. static set of terms), static set of labels, (i) adapted with a dynamic rule set and a dynamic hierarchical relations and (ii) the same model with a static rule sets and hierarchical relations for t_i. The total number of hierarchical relations between the labels decreases as presented in Table 7.7. The total number of SWRL classification rules used to relate the items and the labels increases as is presented in Table 7.7.

During the execution of the adaptive process with concept drift adaptation, the reader can observe in Fig. 7.8 and Fig. 7.9 a global increase of: precision, recall and consequently F1 for micro and macro averages.

The final result of this concept drift adaptation is a gain between 12% (for the 4K) and 32,9% (for the 16K) in macro precision, and a recall gain between 5,5% (for the 4K) and 10,3% (for the 16K) as presented in Table 7.8. Plus, a micro-precision gain between 0,3% (for the 4K) and 8,4% (for the 16K) and a recall gain between 6,1% (for the 4K) and 11,5% (for the 16K) (Table 7.8).

The total number of classification rules, hierarchical relations and classification performance increase with concept-drift adaptation.

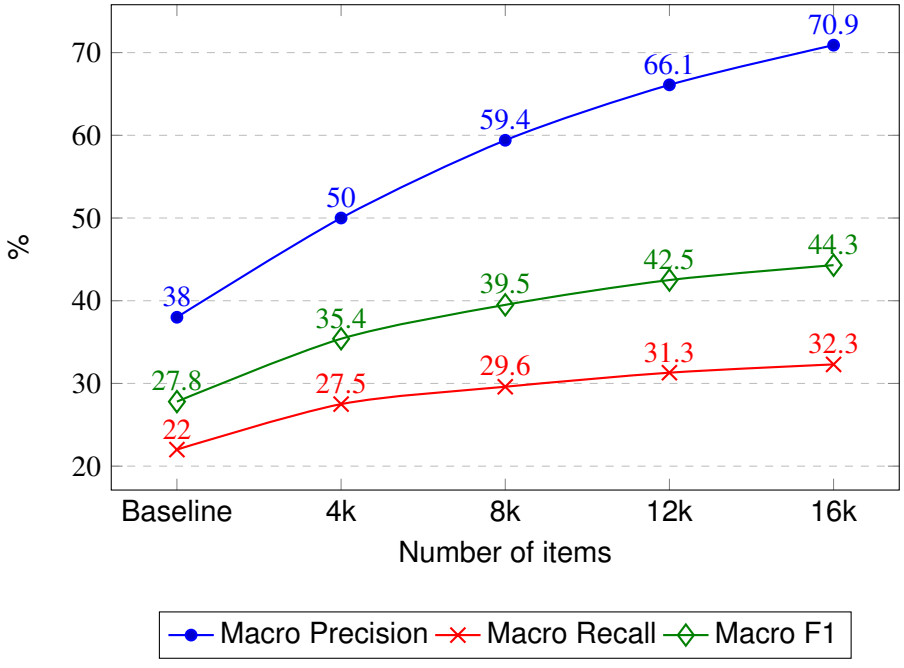


Figure 7.8 – Concept drift macro-averaged values

7.5.4/ Adapt to Feature evolution Experiments

Feature evolution performance is evaluated by comparing the classification model with a dynamic rule set, dynamic hierarchical relations, a static set of labels, while (i) being adapted with a dynamic feature space and (ii) with the same classification model holding a static feature space for t_i. The number of terms used to describe the text items increases as presented in Table 7.7.

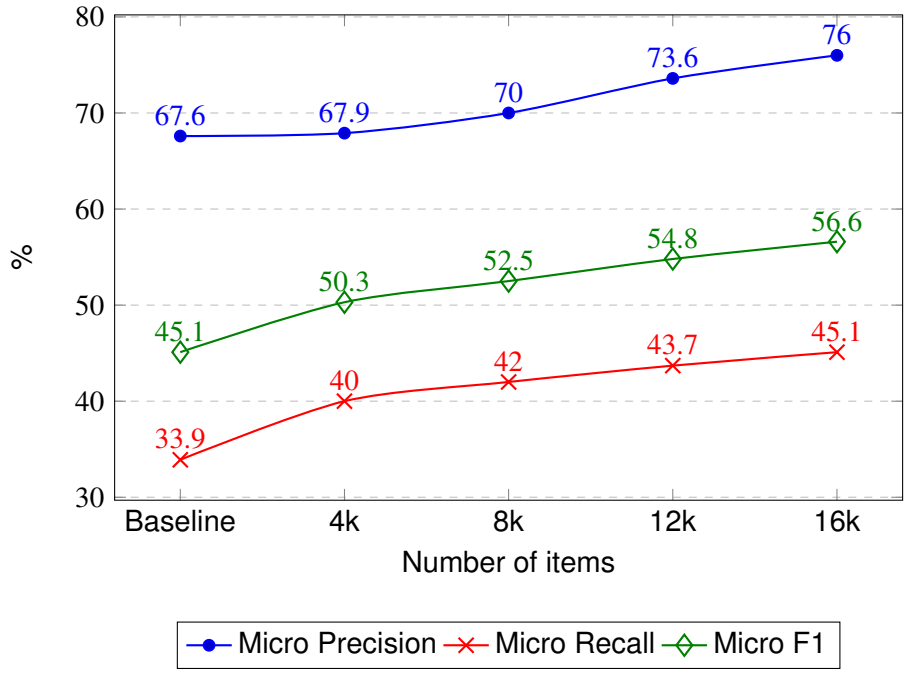


Figure 7.9 – Concept drift micro-averaged values

Table 7.8 – concept drift performance gain comparatively to the baseline

	Macro-Averaged				Micro-Averaged			
	4K	8K	12K	16K	4K	8K	12K	16K
Precision	12,0%	21,4%	28,1%	32,9%	0,3%	3,4%	6,0%	8,4%
Recall	5,5%	7,6%	9,3%	10,3%	6,1%	8,1%	9,8%	11,2%
F1	7,6%	11,6%	14,6%	16,5%	5,2%	7,6%	9,7%	11,5%

During the execution of the adaptive process with concept drift adaptation and feature evolution, in Fig. 7.10 and Fig. 7.11 the reader can observe a global increase of: precision, recall and consequently F1 for micro and macro averages. With feature evolution, a gain is observed in macro precision, between 2,0% (for the 4K) and 2,3% (for the 16K) as well as a recall gain between 0,5% (for the 4K) and 3,9% (for the 16K) (Table 7.9). There is also a gain in micro precision ranging between 0,3% (for the 4K) and 0,9% (for the 16K), and a recall gain between 1,2% (for the 4K) and 4,3% (for the 16K) as presented in Table 7.9. The total number of terms and the classification performance increase with the feature evolution adaptation.

7.5.5/ Adapt to Concept evolution Experiments

Concept evolution performance is evaluated by comparing the classification model with a dynamic rule set, dynamic hierarchical relations, dynamic feature space and by (i) being adapted with a dynamic set of labels and (ii) sharing the same classification model with a static set of label for t_i. The total number of labels used to classify the items decreases according to the number of documents to be maintained, as presented in Table 7.7.

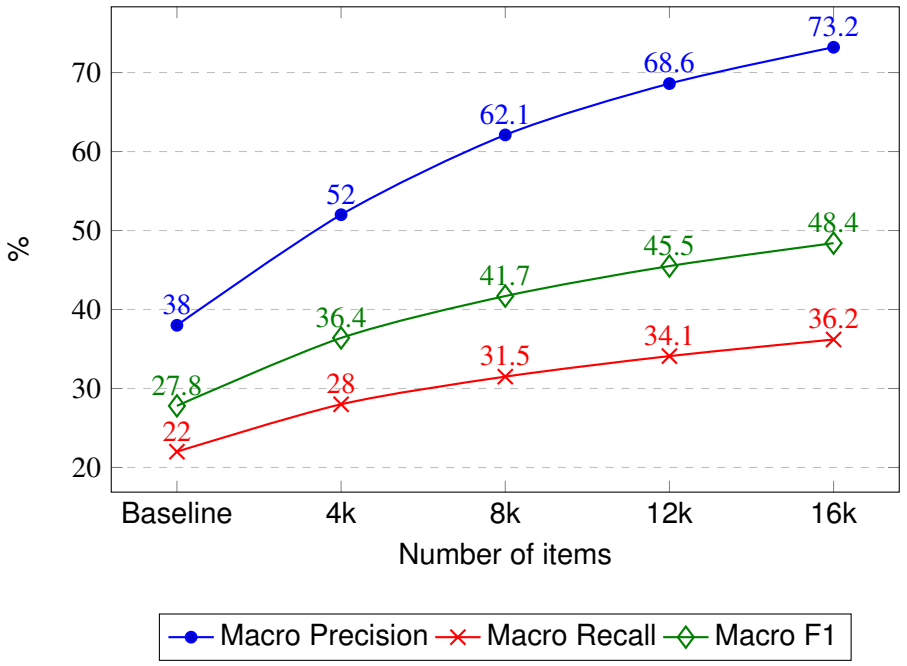


Figure 7.10 – Feature evolution macro-averaged values

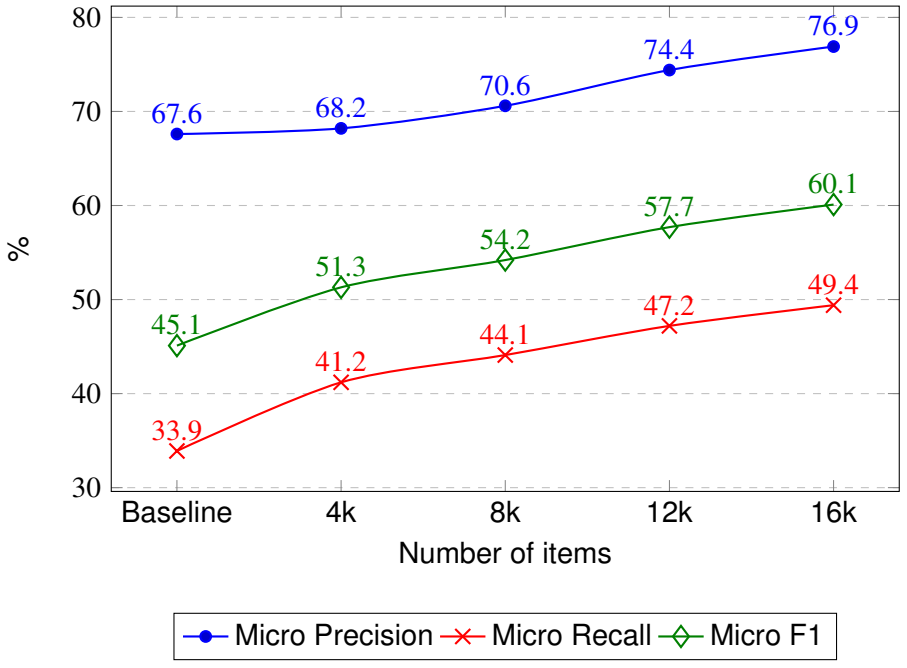


Figure 7.11 – Feature evolution micro-averaged values

However, during the execution of the adaptive process with concept drift adaptation, feature evolution and concept evolution, in Fig. 7.12 and Fig. 7.13, the reader can observe a global increase of: precision, recall and consequently F1 for micro and macro averages.

With concept evolution, a gain is observed in macro precision up to 0,1% plus a recall gain up to 0,1% as presented in Table 7.10. There is also a gain in micro precision up to 0,9% and a recall gain up to 2,2% (Table 7.10).

Table 7.9 – Feature evolution adaptation performance gain to the concept drift adaptation

	Macro-Averaged				Micro-Averaged			
	4K	8K	12K	16K	4K	8K	12K	16K
Precision	2,0%	2,7%	2,5%	2,3%	0,3%	0,6%	0,8%	0,9%
Recall	0,5%	1,9%	2,8%	3,9%	1,2%	2,1%	3,5%	4,3%
F1	0,9%	2,3%	3,1%	4,1%	1,0%	1,8%	2,9%	3,5%

The total number of labels decrease and the classification performance increases with the feature evolution adaptation.

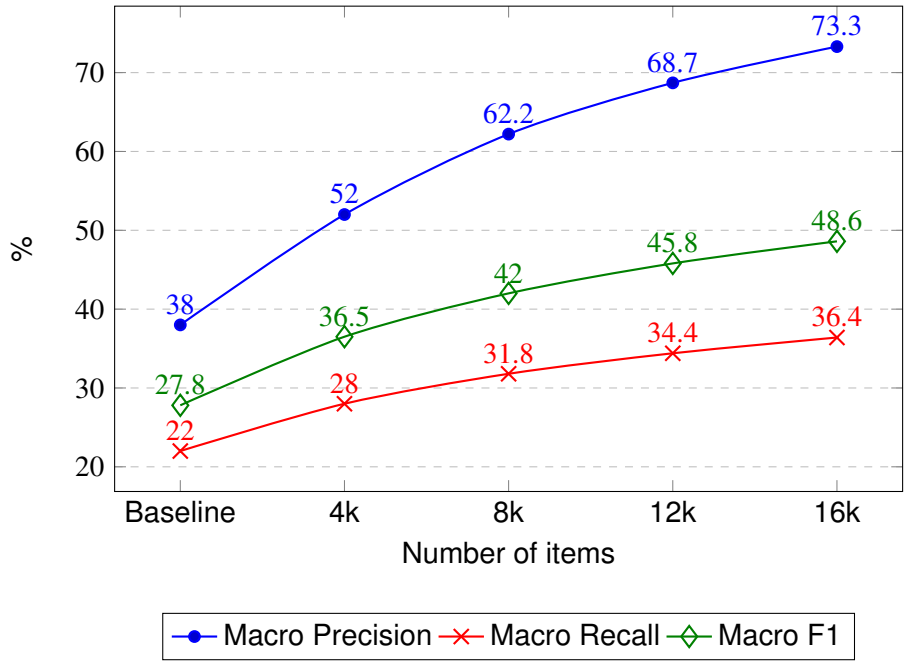


Figure 7.12 – Concept evolution macro-averaged values

Table 7.10 – Concept evolution adaptation performance gain comparatively to the feature evolution adaptation

	Macro-Averaged				Micro-Averaged			
	4K	8K	12K	16K	4K	8K	12K	16K
Precision	0,0 %	0,1 %	0,1 %	0,1 %	0,9 %	0,7 %	0,3 %	0,6%
Recall	0,0 %	0,3 %	0,3 %	0,3 %	0,0 %	2,0 %	2,2 %	2,2%
F1	0,0 %	0,3 %	0,3 %	0,3 %	0,3 %	1,7 %	1,7 %	1,8%

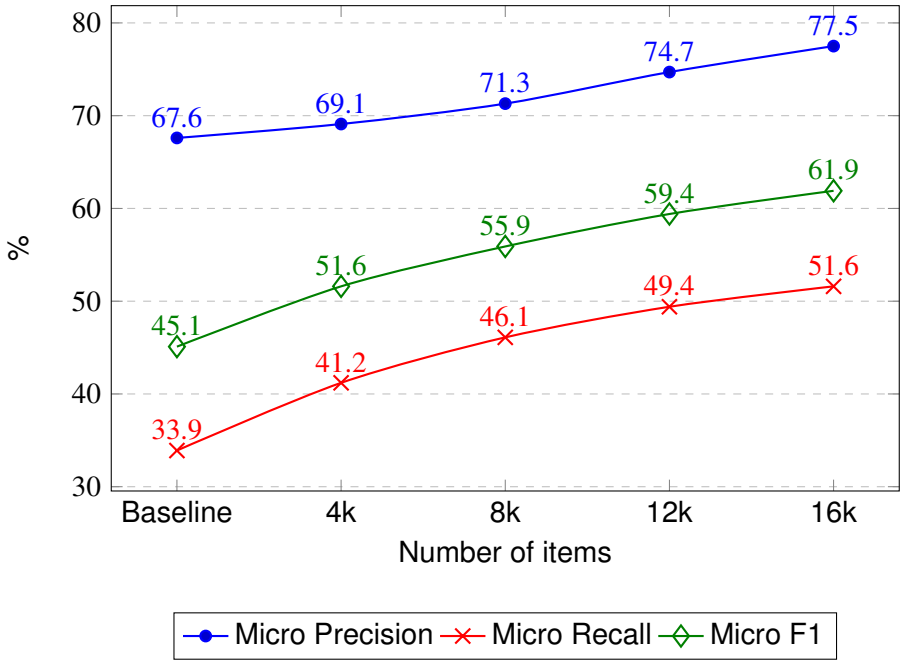


Figure 7.13 – Concept evolution micro-averaged values

7.6/ SUMMARY

In order to adapt the Semantic HMC process to Big Data context, this chapter proposes a new adaptive process for the ontology-described classification model according to non-stationary data streams, regarding feature evolution, concept evolution and concept drift. The proposed approach is the first to propose an adaptive learning process that consistently evolves an ontology-described classification model used for hierarchical multi-label classification regarding a nonstationary stream of unstructured text data in Big Data context.

Four steps comprise the proposed conceptual maintenance process: (1) Input Change Detection, (2) Modification Request Derivation, (3) Model Changes Translation and (4) Classification Model Evolution. This process suggests capturing the semantics of the process in two core entities:

— Input Sensors, which detect the input changes in the input (either data stream or previously applied modifications).

— Modification Request Types, that define:

- the necessary and sufficient conditions to derive each Modification Request according to the input changes;
- the (ontology) Model Changes necessary to carry out the derived Modification Request.

The implementation using Big Data technologies allows the process distribution for multiple machines to achieve high scalability.

The results show that the proposed adaptive process, instantiated with the necessary and sufficient conditions proposed, significantly increases the accuracy (i.e. precision, recall) of the Classification Model. Furthermore, feature evolution and concept evolution slightly improve accuracy in comparison to systems featuring only concept drift.

IV

Discussion and conclusions

8

CONCLUSIONS:THROUGH A MAINTAINABLE SEMANTIC CLASSIFICATION

The research question that guided the research presented in this thesis is: “Is a hierarchical multi-label classifier used to automatically analyse and describe data items based on ontologies and web reasoning maintainable over a large amount of data with great variety in constant generation (i.e. in Big Data context)?".

This research question was motivated by the fact that a semantic analysis of data items from Big Data might be able to provide relevant description to exploit value from Big Data. The maintainability of such process in Big Data was the crucial point because even simple traditional classification processes that deal with a high number of labels can easily become computationally infeasible. The problem of describing items in Big Data context according to a domain model (e.g. an ontology) is a crucial point to exploit value from Big Data. This problem can be addressed at different levels of abstraction, ranging from a theoretical analysis to low level implementation. For example, from a theoretical point of view it is important that new algorithms semantically describe the items better. From a low level point of view, questions are addressed related to new technologies to store and process Big Data. For example, a more efficient distributed storage to store and retrieve the described items or more efficient technologies to process high level algorithms. To this end, in order to solve the problem of semantic description of data items in Big Data context, extensive research must be conducted at every level in order to provide a general solution.

This thesis' approach is on a middle abstraction level that builds the bridge between highly abstract approaches and low level technologies. It researches data item description by combining high level approaches to describe data items (i.e. hierarchical multi-label classification and ontologies) with low level technologies to process and store Big Data (MapReduce to process batch data, Storm topologies to process data in real time and NoSQL databases to store and query data). More specifically, it researches the maintainability of a hierarchical multi-label classification algorithm based on ontologies and web reasoning for Big Data.

In order to facilitate the research in this middle abstraction level, it is important to abstract the outcome of this specific work into more generic principles to be used at different levels. This abstraction process is useful since it contributes to addressing similar problems in a semantic hierarchical multi-label classification for Big Data.

To that end, a number of practical conclusions based on empirical findings learned from experience are enumerated in this chapter. The conclusions are divided into three topics:

— Ontology-described Classification Model learning

— Hierarchical multi-label classification using web reasoning

— Ontology-described adaptive Classification Model

Later in this chapter, these practical conclusions are used to discuss the maintainability of a hierarchical multi-label classification process based on ontologies and web reasoning.

The remaining of this chapter is structured into five sections. Section 8.1 presents the conclusions about learning of the Classification Model from huge batch of unstructured data. Section 8.2 presents the conclusions about classify data items using web reasoning in Big Data. Section 8.3 presents the conclusions about adapting the Classification Model according to a non-stationary stream of unstructured data items. Section 8.4 discusses the conclusions of this thesis presented in the previous sections.

8.1/ Ontology-described Classification Model Learning

In order to perform Hierarchical Multi-Label classification in Big Data without overloading the users, an automatic process to learn a classification model from huge volumes of data items is proposed.

Chapter 4 describes an unsupervised process to automatically learn a ontology-described label hierarchy from unstructured text. The process is based on a hybrid approach between statistical and lexical approaches to learn the labels. Furthermore, in Chapter 5 the labels are described using their related terms. Based on those terms, classification rules are created to classify data items with labels. The created Classification Rules define the necessary and sufficient terms for an item to be classified with a label.

The Classification Model learning process was successfully implemented in a scalable and distributed platform to process Big Data. A basic information-retrieval approach is used to select the labels and a highly scalable subsumption method to learn the hierarchical relations between labels. The rules are serialized in a horn clause language to reduce the reasoning effort in relation to translating the rules into logical constraints of an ontology captured in Description Logics, thus improving the scalability and performance of the system.

The results in Chapter 4 show that the label hierarchy is automatically learned from large datasets of unstructured text data in a scalable way.

The evaluation in Chapter 5 highlights two aspects of the learned classification model:

— The classification rules can be learned from huge amount of data using simple but highly scalable techniques;

— The performance of the learned classification model to classify unstructured text documents is comparable to the performance of algorithms from the state of the art in the machine-learning field.

Therefore, the first conclusion is that an ontology-described classification model can be automatically learned from huge volumes of data.

8.2/ Hierarchical Multi-label Classification using Web Reasoning

To classify the data items according the ontology-based classification model, web reasoning is used. In order to do so, the learned ontology representing the classification model is populated with the data items to be classified, and the web reasoner is then applied so as to classify the items with the labels according to the classification rules.

Therefore, two types of classification were proposed in Chapter 5. The first consists of classifying all items with all labels before query-time, so as to derive all implicit information. In this case, items are classified using a forward-chaining web reasoner. The reasoner creates an inference closure and queries are then answered using technologies developed in the field of data management. The second consists of classifying the items directly at query time in order to limit the derivation to information that is related to the query. In this case, items are classified using a backward-chaining web reasoner that applies only the necessary and sufficient rules to answer the performed query.

The evaluation results described in Chapter5 prove that the classification performance of the Semantic HMC process that uses learned rule-enriched ontology and rule-based reasoning to classify unstructured text documents is comparable to the performance of algorithms from the state of the art in the machine-learning field.

In summary, the proposed ontology learning process that uses web reasoners, specially those based on rules, can be used for hierarchical multi-label classification in Big Data context.

8.3/ Ontology-described Adaptive Classification Model

In order to adapt the ontology-described classification model to Big Data context, a new adaptive process is proposed for the ontology-described classification model according to non-stationary data streams, regarding feature-evolution, concept-evolution and concept-drift (Chapter 7).

The proposed adaptive learning process is the first to consistently evolve an ontology-described classification model used for hierarchical multi-label classification regarding a non-stationary stream of unstructured text data. The adaptive process is adaptable to different types of ontology-described classification models used in Big Data classification processes (e.g. either considering words, images, sounds, GPS coordinates). This process was successfully adopted in the specific case of the Semantic HMC. Its implementation using Big Data technologies allows the process distribution for large sets of machines to achieve high scalability in terms of Volume and Velocity.

The results in Chapter 7 show that the proposed adaptive process instantiated with the proposed necessary and sufficient conditions increases the classification quality performance of the classification model. The concept-drift adaptation significantly improves the precision and the recall of the classification model. Furthermore, feature evolution and

concept evolution slightly improve the precision and recall compared to the concept-drift adaptation only.

In summary, an ontology-based classification model can be adapted to non-stationary streams of data in order to capture changes in the domain and consequently improve classification quality.

8.4/ DISCUSSION

While the evaluation presented in this thesis shows that a hierarchical multi-label classification based on ontologies and web reasoning for Big Data is indeed possible, only a part of the general problem was addressed in this thesis. This section discusses the conclusions and envisioned future work based on the identified achievements and limitations.

The first practical conclusion is that an ontology-described classification model can be automatically learned from huge volumes of data. In the Semantic HMC process, methods from the state of the art for learning the classification model are used. The term detection is based on statistical methods, the label selection is based on an information retrieval method, the hierarchy-generation process is based on an existing subsumption method and the rules are created by exploiting term co-occurrence using two thresholds (i.e. Alpha and Beta).

Additionally, the parameters (i.e. threshold values) used for learning the classification model have an high impact in the classification process performance. In fact, wrong values can struggle with the process (e.g. too high Alpha and Beta thresholds can lead to a low number of rules and consequently no (or a low number) of classification(s) are performed). The experiments performed in this thesis adopted the default values from the technologies used, and others where assigned without an extensive study of their impact. However, to achieve the best possible performance, the thresholds must be calculated for each data set and maintained during the process of data streams.

Despite those, the evaluation results (Section 5.5.2) prove that the classification performance of the Semantic HMC process that uses ontologies and rule-based reasoning to classify unstructured text documents is comparable to the performance of algorithms from the state of the art in the machine-learning field.

Regarding the state of the art, the term detection process can be improved by capturing the semantics behind the words (i.e. trough advanced Natural Language Processing such as word sense disambiguation and named entity detection). Moreover, a study of the relevancy and pertinence of other methods in label selection and creation of classification rules must be considered in further work. For example, studying the adaptability of recent multi-label classification techniques [Zhang et al., 2014], to horn clause rules that can be processed by web reasoners.

One must be aware that the Semantic HMC process has specific semantics. Even if the Semantic HMC's main concepts are captured by the ontology used to describe the classification model, changes in process semantics (e.g. new method to create rules) can impact the way in which the web reasoner is used to classify the items (e.g. new types of classification rules). This leads to the second practical conclusion: hierarchical multi-label classification can be performed using web reasoning.

To classify data items in Big Data context using web reasoning, a translation of the clas-

sification rules into simple Horn Clause rules is proposed. The use of a larger number of simple Horn Clause rules instead of a smaller number of more complex rules reduces the reasoning effort, thus improving the scalability and performance of the classification process. More complex types of rules generated from other approaches can compromise the performance of the classification process. For example, in [Werner et al., 2014] the authors translate the classification rules into complex logical constraints of an ontology captured in Description Logic with SHOIN(D) expressivity. In order to perform the classification, they use out-of-the-box reasoners to classify text documents. However, the system's scalability is limited and they cannot be used with large datasets as it is required in a Big Data context.

The current state of the art in large/web scale reasoners has limited application scope. To evaluate the classification process, a backward chaining rule-based web reasoner is used with some adaptations (i.e. rule selection) (Chapter 5). Even if the proposed approach scales to a large amounts of triples, it is limited to one machine. The use of other web reasoners (e.g. the ones using MapReduce [Mutharaju et al., 2010, Urbani et al., 2011, Urbani et al., 2012, Wu et al., 2013]) must be studied in order to improve and optimize classification performance.

The third practical conclusion is that an ontology-based classification model can be automatically adapted to non-stationary streams of data. A general adaptive process is proposed in Section 7.2 and then instantiated to the specific case of the Semantic HMC process in Section 7.3. The adaptive classification model process must be specifically instantiated so as to capture the specific semantics of each classification process.

If the approach to learn the classification model changes (e.g. new algorithm to detect terms, labels or create rules), the adaptive process can be impacted. Ergo, the adaptability of each new approach used to learn the classification model must be studied and the adaptive process must be instantiated accordingly.

Section 7.5 proves that the adaptive approach process improves classification performance. However, the process is based on simple necessary and sufficient conditions that can be improved by adopting/adapting other methods from the state of the art of concept drift [Gama et al., 2014]. Also in future work, the extension of the maintenance process must be studied beyond the adaptation of a stream of data, by supporting semantic expressive and controlled corrections from external actors.

Beyond previous considerations, other limitations were identified motivating further research.

First, Big Data is uncertain, erroneous, imprecise and incomplete (e.g. user-generated data). In this thesis, the Veracity dimension that refers to inconsistency and data quality problems was not considered. In unsupervised classification processes, such as the Semantic HMC, even minor errors can accumulate without proper data quality management, resulting in revenue loss and process inefficiency. In this specific context, it can cause quality problems in the learned classification model.

Secondly, some assumptions where made about the scalability of the used technologies to process Big Data. The technologies used to implement the prototypes are proved in literature as highly scalable through huge sets of not-so-expensive machines. Based on that assumption, the evaluation is performed using a small cluster of four machines for proof of concept and it is assumed that the processes can be used in a larger set of machines. Yet process scalability using a cluster with a higher number of machines, and

the impact of the process parameters (i.e. thresholds) in process performance need to be experimentally proved. Likewise, the process performance improvement by automatically adapting the process parameters for each data set, as well as to taking the stream of data into consideration, requires further research.

Based on the conclusions made in the previous sections, the methods presented in the previous chapters (Part II and Part III of this thesis) were able to limit the problems outlined in Chapter 1 and demonstrate an automatic data item description through a maintainable hierarchical multi-label classification process in Big Data context.

BIBLIOGRAPHY

[AL-Nabi et al., 2013] AL-Nabi, D. L. A., et Ahmed, S. S. (2013). **Survey on Classification Algorithms for Data Mining:(Comparison and Evaluation)**. *Computer Engineering and Intelligent Systems*, 4(8):18–24.

[Amandeep Kaur Mann, 2013] Amandeep Kaur Mann, N. K. (2013). **Review Paper on Clustering Techniques**. *Gjcst*, 13(5):43–47.

[Anil et al., 2010] Anil, R., Owen, S., Dunning, T., et Friedman, E. (2010). **Mahout in Action**. Manning Publications Co. Sound View Ct. #3B Greenwich, CT 06830.

[Antoniou et al., 2009] Antoniou, G., et van Harmelen, F. (2009). **Web ontology language: OWL**. In *Handbook on ontologies*, pages 91–110. Springer.

[Aparicio et al., 2013] Aparicio, R., et Acuna, E. (2013). **Using Ontologies To Improve Document Classification With Transductive Support Vector Machines**. *International Journal*, 3(3):1–15.

[Asuncion et al., 2009] Asuncion, A., Welling, M., Smyth, P., et Teh, Y. W. (2009). **On smoothing and inference for topic models**. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 27–34. AUAI Press.

[Baader, 2003] Baader, F. (2003). **Basic Description Logics**. *The description logic handbook*, pages 43—-95.

[Baader et al., 2009] Baader, F., Horrocks, I., et Sattler, U. (2009). **Description Logics**. In Staab, S., et Studer, R., editors, *Handbook on Ontologies SE - 1*, International Handbooks on Information Systems, pages 21–43. Springer Berlin Heidelberg.

[Baader et al., 2006] Baader, F., Lutz, C., et Suntisrivaraporn, B. (2006). **Efficient reasoning in EL**. In *2006 International Workshop on Description Logics DL'06*, page 15.

[Baader et al., 2001] Baader, F., et Sattler, U. (2001). **An Overview of Tableau Algorithms for Description Logics**. *Studia Logica*, 69(1):5–40.

[Bachmair et al., 2001] Bachmair, L., et Ganzinger, H. (2001). **Resolution Theorem Proving.** *Handbook of automated reasoning*, 1:19–99.

[Bao et al., 2011] Bao, J., Braines, D., et Mott, D. (2011). **A Distributed Tableau Algorithm for the ALC Description Logic**.

[Baraniuk, 2011] Baraniuk, R. G. (2011). **More is less: signal processing and the data deluge**. *Science*, 331(6018):717–719.

[Baumgartner et al., 1996] Baumgartner, P., Furbach, U., et Niemelä, I. (1996). **Hyper Tableaux**.

[Ben-Ari, 2012a] Ben-Ari, M. (2012a). **Mathematical logic for computer science**. Springer.

[Ben-Ari, 2012b] Ben-Ari, M. (2012b). **Propositional Logic: Deductive Systems**. In *Mathematical Logic for Computer Science SE - 3*, pages 49–73. Springer London.

[Ben-David et al., 2010] Ben-David, D., Domany, T., et Tarem, A. (2010). **Enterprise data classification using semantic web technologies**. In *The Semantic Web–ISWC 2010*, ISWC'10, pages 66–81, Berlin, Heidelberg. Springer-Verlag.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., et Lassila, O. (2001). **The Semantic Web**. *Scientific American*, 284(5):34–43.

[Bi et al., 2011] Bi, W., et Kwok, J. (2011). **Multi-label classification on tree-and DAG-structured hierarchies**. *Yeast*, pages 1–8.

[Bifet et al., 2014] Bifet, A., et Morales, G. D. F. (2014). **Big Data Stream Learning with SAMOA**. *2014 IEEE International Conference on Data Mining Workshop*, pages 1199–1202.

[binti Oseman et al., 2010] binti Oseman, K., Haris, N. A., et bin Abu Bakar, F. (2010). **Data Mining in Churn Analysis Model for Telecommunication Industry**. *Journal of Statistical Modeling and Analytics Vol*, 1(19-27).

[Bobadilla et al., 2013] Bobadilla, J., Ortega, F., Hernando, A., et Gutiérrez, A. (2013). **Recommender systems survey**. *Knowledge-Based Systems*, 46(0):109–132.

[Brickley et al., 2004] Brickley, D., et Guha, R. (2004). **RDF Vocabulary Description Language 1.0: RDF Schema**.

[Cambria et al., 2014] Cambria, E., et White, B. (2014). **Jumping NLP curves: A review of natural language processing research**. *IEEE Computational Intelligence Magazine*, 9(2):48–57.

[Caraballo, 1999] Caraballo, S. A. (1999). **Automatic Construction of a Hypernym-labeled Noun Hierarchy from Text**. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 120–126, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Cerri et al., 2014] Cerri, R., Barros, R. C., et de Carvalho, A. C. (2014). **Hierarchical multi-label classification using local neural networks**. *Journal of Computer and System Sciences*, 80(1):39–56.

[Chen et al., 2014] Chen, M., Mao, S., et Liu, Y. (2014). **Big data: A survey**. *Mobile Networks and Applications*, 19(2):171–209.

[Cherman et al., 2011] Cherman, E. A., Monard, M. C., et Metz, J. (2011). **Multi-label problem transformation methods: A case study**. *CLEI Electronic Journal*, 14(1):4.

[Cilibrasi et al., 2007] Cilibrasi, R. L., et Vitanyi, P. M. B. (2007). **The Google similarity distance**. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383.

[Cimiano et al., 2003] Cimiano, P., Staab, S., et Tane, J. (2003). **Automatic Acquisition of Taxonomies from Text: FCA meets NLP**. *Proceedings of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining*, pages 10–17.

[Costa et al., 2013] Costa, R., Lima, C., Sarraipa, J., et Jardim-Gonçalves, R. (2013). **Semantic enrichment of building and construction knowledge sources using a domain ontology for classification**. In *AIP Conference Proceedings*, volume 1558, pages 1381–1384.

[Cui et al., 2014] Cui, W., Liu, S., Member, S., Wu, Z., et Wei, H. (2014). **How Hierarchical Topics Evolve in Large Text Corpora**. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2281–2290.

[Cukier, 2010] Cukier, K. (2010). **Data, data everywhere: A special report on managing information**. Economist Newspaper.

[De Knijff et al., 2013] De Knijff, J., Frasincar, F., et Hogenboom, F. (2013). **Domain taxonomy learning from text: The subsumption method versus hierarchical clustering**. *Data and Knowledge Engineering*, 83:54–69.

[Dean et al., 2008] Dean, J., et Ghemawat, S. (2008). **MapReduce : Simplified Data Processing on Large Clusters**. *Communications of the ACM*, 51(1):1–13.

[Ditzler et al., 2015] Ditzler, G., Roveri, M., Alippi, C., et Polikar, R. (2015). **Learning in Nonstationary Environments: A Survey**. *IEEE Computational Intelligence Magazine*, 10(4):12–25.

[Dunning, 1993] Dunning, T. (1993). **Accurate Methods for the Statistics of Surprise and Coincidence**. *Computational Linguistics*, 19:61–74.

[Elberrichi et al., 2012] Elberrichi, Z., Amel, B., et Malika, T. (2012). **Medical Documents Classification Based on the Domain Ontology MeSH**. *The International Arab Journal of e-Technology*, 2(4):210–215.

[Fang et al., 2010] Fang, J., Guo, L., et Niu, Y. (2010). **Documents classification by using ontology reasoning and similarity measure**. In *Proceedings - 2010 7th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010*, volume 4, pages 1535–1539.

[Farias et al., 2015] Farias, T. M., Roxin, A., et Nicolle, C. (2015). **FOWLA, A federated architecture for ontologies**. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9202, pages 97–111.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., et Smyth, P. (1996). **From data mining to knowledge discovery in databases**. *AI magazine*, 17(3):37.

[Feldman et al., 1998] Feldman, R., Fresko, M., Kinar, Y., Lindell, Y., Liphstat, O., Rajman, M., Schler, Y., et Zamir, O. (1998). **Text Mining at the Term Level**. In *Second European Symposium, PKDD '98*, number September, pages 65–73. Springer-Verlag.

[Flouris et al., 2007] Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., et Antoniou, G. (2007). **Ontology Change: Classification and Survey**. *Knowl. Eng. Rev.*, 23(2):117–152.

[Galinina et al., 2013] Galinina, A., et Borisov, A. (2013). **Knowledge modelling for ontology-based multiattribute classification system**. *Applied Information and Communication Technologies*, pages 103–109.

[Gama et al., 2014] Gama, J., Bifet, A., Pechenizkiy, M., et Bouchachia, A. (2014). **A Survey on Concept Drift Adaptation**. *CM Computing Surveys (CSUR)*, 46(4).

[Gantz et al., 2011] Gantz, J., et Reinsel, D. (2011). **Extracting value from chaos**. *IDC iview*, (June):1–12.

[Gao et al., 2007] Gao, J., Fan, W., et Han, J. (2007). **On appropriate assumptions to mine data streams: Analysis and practice**. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 143–152.

[Griffiths et al., 2004] Griffiths, T. L., et Steyvers, M. (2004). **Finding scientific topics**. *Proceedings of the National academy of Sciences*, 101(1):5228–5235.

[Gruber, 1993] Gruber, T. R. (1993). **A Translation Approach to Portable Ontology Specifications by A Translation Approach to Portable Ontology Specifications**. *Knowledge Creation Diffusion Utilization*, 5:199–220.

[Haase et al., 2005] Haase, P., et Stojanovic, L. (2005). **Consistent Evolution of OWL Ontologies**. In Gómez-Pérez, A., et Euzenat, J., editors, *The Semantic Web: Research and Applications SE - 13*, volume 3532 of *Lecture Notes in Computer Science*, pages 182–197. Springer Berlin Heidelberg.

[Hearst, 1992] Hearst, M. a. (1992). **Automatic Acquisition of Hyponyms ftom Large Text Corpora**. *Proceedings of the 14th conference on Computational Linguistics*, 2:23–28.

[Hitzler et al., 2013] Hitzler, P., et Janowicz, K. (2013). **Linked Data, Big Data, and the 4th Paradigm**. *Semantic Web*, 4(3):333–335.

[Hohpe et al., 2004] Hohpe, G., et Woolf, B. (2004). **Enterprise integration patterns: Designing, building, and deploying messaging solutions**. Addison-Wesley Professional.

[Hulten et al., 2001] Hulten, G., Spencer, L., et Domingos, P. (2001). **Mining time-changing data streams**. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 97–106.

[Johnson et al., 2010] Johnson, I., Abécassis, J., Charnomordic, B., Destercke, S., et Thomopoulos, R. (2010). **Making ontology-based knowledge and decision trees interact: An approach to enrich knowledge and increase expert confidence in data-driven models**. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6291 LNAI, pages 304–316. Springer.

[Jones, 2012] Jones, M. T. (2012). **Process real-time big data with Twitter Storm- An introduction to streaming big data**. *IBM Technical Library*, pages 1–9.

[Katakis et al., 2006] Katakis, I., Tsoumakas, G., et Vlahavas, I. (2006). **Dynamic feature space and incremental feature selection for the classification of textual data streams**. *Knowledge Discovery from Data Streams*, pages 107–116.

[Khan et al., 2014] Khan, M. A.-u.-d., Uddin, M. F., et Gupta, N. (2014). **Seven V's of Big Data understanding Big Data to extract value**. *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, pages 1–5.

[Klein et al., 2003] Klein, M., et Noy, N. F. (2003). **A component-based framework for ontology evolution**. In *Proceedings of the IJCAI*, volume 3. Citeseer.

[Kocev et al., 2007] Kocev, D., Vens, C., Struyf, J., et Džeroski, S. (2007). **Ensembles of multi-objective decision trees**. In *European Conference on Machine Learning*, pages 624–631. Springer.

[Kotsiantis, 2007] Kotsiantis, S. B. (2007). **Supervised Machine Learning : A Review of Classification Techniques**. *Informatica*, 31(3):249–268.

[Kumar et al., 2012] Kumar, R., et Verma, R. (2012). **Classification Algorithms for Data Mining: A Survey**. *International Journal of Innovations in Engineering and Technology (IJIET)*.

[Lance, 1967] Lance, G. N. (1967). **A general theory of classificatory sorting strategies: II. Clustering systems**. *The Computer Journal*, 10(3):271–277.

[Lin, 2013] Lin, J. (2013). **Monoidify! monoids as a design principle for efficient mapreduce algorithms**. *arXiv preprint arXiv:1304.7544*.

[Liu et al., 2012a] Liu, C., Qi, G., Wang, H., et Yu, Y. (2012a). **Reasoning with large scale ontologies in fuzzy pD* Using MapReduce**. *Computational Intelligence Magazine, IEEE*, 7(2):54–66.

[Liu et al., 2014] Liu, L., Zhang, P., Fan, R., Zhang, R., et Yang, H. (2014). **Modeling ontology evolution with SetPi**. *Information Sciences*, 255:155–169.

[Liu et al., 2012b] Liu, X., Song, Y., Liu, S., et Wang, H. (2012b). **Automatic taxonomy construction from keywords**. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1433–1441.

[Lomotey et al., 2013] Lomotey, R. K., et Deters, R. (2013). **Terms extraction from unstructured data silos**. *Proceedings of 2013 8th International Conference on System of Systems Engineering: SoSE in Cloud Computing and Emerging Information Technology Applications, SoSE 2013*, pages 19–24.

[Lops et al., 2011] Lops, P., de Gemmis, M., et Semeraro, G. (2011). **Content-based recommender systems: State of the art and trends**. In *Recommender Systems Handbook*, pages 73–105. Springer.

[Ma et al., 2014] Ma, C., Zhang, H. H., et Wang, X. (2014). **Machine learning for Big Data analytics in plants**. *Trends in Plant Science*, 19(12):798–808.

[Madjarov et al., 2012] Madjarov, G., Kocev, D., Gjorgjevikj, D., Džeroski, S., Madjarov, G., Gjorgjevikj, D., Kocev, D., Gjorgjevikj, D., et Džeroski, S. (2012). **An extensive experimental comparison of methods for multi-label learning**. *Pattern Recognition*, 45(9):3084–3104.

[Maedche et al., 2001] Maedche, A., et Volz, R. (2001). **The ontology extraction & maintenance framework Text-To-Onto**. *Workshop on Integrating Data Mining and Knowledge Management*, pages 1–12.

[Maimon et al., 2010] Maimon, O., et Rokach, L., editors (2010). **Data Mining and Knowledge Discovery Handbook, 2nd ed**. Springer.

[Manola et al., 2004] Manola, F., Miller, E., et McBride, B. (2004). **RDF primer**. *W3C recommendation*, 10:1–107.

[Mashat et al., 2012] Mashat, A. F., Fouad, M. M., Philip, S. Y., et Gharib, T. F. (2012). **A Decision Tree Classification Model for University Admission System**. *International Journal of Advanced Computer Science and Applications*, 3(10):17–21.

[Masud et al., 2010] Masud, M. M., Qing, C., Khan, L., Aggarwal, C., Jing, G., Jiawei, H., et Thuraisingham, B. (2010). **Addressing concept-evolution in concept-drifting data streams**. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 929–934.

[Medelyan et al., 2013] Medelyan, O., Manion, S., Broekstra, J., Divoli, A., Huang, A. L., et Witten, I. H. (2013). **Constructing a focused taxonomy from a document collection**. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7882 LNCS:367–381.

[Meijer et al., 2014] Meijer, K., Frasincar, F., et Hogenboom, F. (2014). **A semantic approach for extracting domain taxonomies from text**. *Decision Support Systems*, 62:78–93.

[Milner, 1999] Milner, R. (1999). **Communicating and mobile systems: the pi calculus**. Cambridge university press.

[Moller et al., 2009] Moller, R., et Haarslev, V. (2009). **Tableau-Based Reasoning**. *Handbook on Ontologies*, page 654.

[Motik et al., 2006] Motik, B., et Sattler, U. (2006). **A comparison of reasoning techniques for querying large description logic aboxes**. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 227–241. Springer.

[Mutharaju et al., 2010] Mutharaju, R., Maier, F., et Hitzler, P. (2010). **A MapReduce Algorithm for SC**. In *23rd International Workshop on Description Logics DL2010*, page 456.

[Nardi et al., 2003] Nardi, D., et Brachman, R. J. (2003). **An Introduction to Description Logics.** In *Description Logic Handbook*, pages 1–40.

[Noy et al., 2004] Noy, N. F., et Klein, M. (2004). **Ontology Evolution: Not the Same as Schema Evolution**. *Knowledge and Information Systems*, 6(4):428–440.

[Obrst, 2003] Obrst, L. (2003). **Ontologies for semantically interoperable systems**. In *Proceedings of the twelfth international conference on Information and knowledge management - CIKM '03*, pages 366–369. ACM Press.

[Papanikolaou et al., 2015] Papanikolaou, Y., Rubin, T. N., et Tsoumakas, G. (2015). **Improving Gibbs Sampling Predictions on Unseen Data for Latent Dirichlet Allocation**. *arXiv preprint arXiv:1505.02065*.

[Pittet et al., 2014] Pittet, P., Cruz, C., et Nicolle, C. (2014). **An ontology change management approach for facility management**. *Computers in Industry*, 65(9):1301–1315.

[Qin, 2014] Qin, S. J. (2014). **Process data analytics in the era of big data**. *American Institute of Chemical Engineers (AIChE) Journal*, 60(9):3092–3100.

[Read et al., 2011] Read, J., Pfahringer, B., Holmes, G., et Frank, E. (2011). **Classifier chains for multi-label classification**. *Machine learning*, 85(3):333–359.

[Ricci et al., 2010] Ricci, F., Rokach, L., Shapira, B., et Kantor, P. B. (2010). **Recommender Systems Handbook**. Springer, 1st edition.

[Robinson, 1965] Robinson, J. A. (1965). **Automatic deduction with hyper-resolution**. *International Journal of Computer Mathematics*, 1(1):227–234.

[Rokach, 2007] Rokach, L. (2007). **Data mining with decision trees: theory and applications**, volume 69. World Scientific.

[Rudolph, 2011] Rudolph, S. (2011). **Foundations of Description Logics**. In *Reasoning Web 2011, Lecture Notes in Computer Science, volume 6848*, pages 76–136. Springer.

[Salton et al., 1988] Salton, G., et Buckley, C. (1988). **Term-weighting approaches in automatic text retrieval**. *Information Processing & Management*, 24(5):513–523.

[Sanderson et al., 1999] Sanderson, M., et Croft, B. (1999). **Deriving concept hierarchies from text**. *22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 206–213.

[Santos et al., 2009] Santos, A. P., et Rodrigues, F. (2009). **Multi-label Hierarchical Text Classification using the ACM Taxonomy**. *14th Portuguese Conference on Artificial Intelligence EPIA 2009*, pages 553–564.

[Schlicht et al., 2008] Schlicht, A., et Stuckenschmidt, H. (2008). **Distributed Resolution for ALC.** In *Description Logics*.

[Shearer et al., 2009] Shearer, R., et Horrocks, I. (2009). **Hypertableau Reasoning for Description Logics**. *Journal of Artificial Intelligence Research*, 36(June 2008):165–228.

[Sheth, 2014] Sheth, A. (2014). **Transforming Big Data into Smart Data: Deriving value via harnessing Volume, Variety, and Velocity using semantic techniques and technologies**. *2014 IEEE 30th International Conference on Data Engineering*, pages 2–2.

[Shvachko et al., 2010] Shvachko, K., Kuang, H. K. H., Radia, S., et Chansler, R. (2010). **The Hadoop Distributed File System**. *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*.

[Sidorov et al., 2014] Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., et Chanona-Hernández, L. (2014). **Syntactic N-grams as machine learning features for natural language processing**. *Expert Systems with Applications*, 41(3):853–860.

[Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., et Katz, Y. (2007). **Pellet: A practical owl-dl reasoner**. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51 – 53. Software Engineering and the Semantic Web.

[Smadja et al., 1990] Smadja, F., et McKeown, K. R. (1990). **Automatically extracting and representing collocations for language generation**. *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, pages 252–259.

[Smullyan, 1995] Smullyan, R. M. (1995). **First-order logic**. Courier Dover Publications.

[Spinosa et al., 2008] Spinosa, E. J., de Leon F de Carvalho, A. P., et Gama, J. (2008). **Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks**. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 976–980.

[Stojanovic et al., 2002] Stojanovic, L., Maedche, A., Motik, B., et Stojanovic, N. (2002). **User-driven ontology evolution management**. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, volume 2473 of *LNCS*, pages 133–140, Heidelberg. Heidelberg, Springer-Verlag.

[Studer et al., 1998] Studer, R., Benjamins, V. R., et Fensel, D. (1998). **Knowledge engineering: Principles and methods**. *Data & Knowledge Engineering*, 25(1-2):161–197.

[Sucar et al., 2014] Sucar, L. E., Bielza, C., Morales, E. F., Hernandez-Leal, P., Zaragoza, J. H., et Larrañaga, P. (2014). **Multi-label classification with Bayesian network-based chain classifiers**. *Pattern Recognition Letters*, 41:14–22.

[Syed et al., 2013] Syed, A. R., Gillela, K., et Venugopal, C. (2013). **The Future Revolution on Big Data**. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(6):2446–2451.

[ter Horst, 2005] ter Horst, H. J. (2005). **Completeness, decidability and complexity of entailment for {RDF} schema and a semantic extension involving the {OWL} vocabulary**. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2–3):79 – 115. Selcted Papers from the International Semantic Web Conference, 2004 ISWC, 20043rd. International Semantic Web Conference, 2004.

[The Apache Software Foundation, 2012] The Apache Software Foundation (2012). **Hadoop**.

[The Apache Software Foundation, 2013] The Apache Software Foundation (2013). **Mahout**.

[The Apache Software Foundation, 2014] The Apache Software Foundation (2014). **Apache Hadoop**.

[Toshniwal et al., 2014] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et Others (2014). **Storm@ twitter**. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM.

[Toutanova et al., 2000] Toutanova, K., et Manning, C. D. (2000). **Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger**. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 63–70.

[Troester, 2012] Troester, M. (2012). **Big data meets big data analytics**. *Cary, NC: SAS Institute Inc*.

[Tsai et al., 2015] Tsai, C. W., Lai, C. F., Chao, H. C., et Vasilakos, A. V. (2015). **Big data analytics : a survey**. *Journal of Big Data*, pages 1–32.

[Tsarkov et al., 2006] Tsarkov, D., et Horrocks, I. (2006). **FaCT++ Description Logic Reasoner: System Description**. In Furbach, U., et Shankar, N., editors, *Proceedings of the Third International Joint Conference (IJCAR)*, volume 4130, pages 292–297. Springer Berlin / Heidelberg.

[Tsoumakas et al., 2007] Tsoumakas, G., Katakis, I., et Overview, A. (2007). **Multi-Label Classification : An Overview**. *International Journal of Data Warehousing and Mining*, 3(September):1–13.

[Tsoumakas et al., 2008] Tsoumakas, G., Katakis, I., et Vlahavas, I. (2008). **Effective and efficient multilabel classification in domains with large number of labels**. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, pages 30–44.

[Tsoumakas et al., 2010] Tsoumakas, G., Katakis, I., et Vlahavas, I. (2010). **Mining multi-label data**. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer.

[Tsymbal, 2004] Tsymbal, A. (2004). **The problem of concept drift: definitions and related work**. *Computer Science Department, Trinity College Dublin*, 4(C):2004–15.

[Tzitzikas et al., 2013] Tzitzikas, Y., Kampouraki, M., et Analyti, A. (2013). **Curating the Specificity of Ontological Descriptions under Ontology Evolution**. *Journal on Data Semantics*.

[Urbani, 2010] Urbani, J. (2010). **Scalable and parallel reasoning in the Semantic Web**. In *The Semantic Web: Research and Applications*, pages 488–492. Springer.

[Urbani, 2013] Urbani, J. (2013). **Three Laws Learned from Web-scale Reasoning**. In *2013 AAAI Fall Symposium Series*, pages 76–79.

[Urbani et al., 2010] Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., et Bal, H. (2010). **OWL reasoning with WebPIE: Calculating the closure of 100 billion triples**. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6088, pages 213–227. Springer Berlin Heidelberg.

[Urbani et al., 2012] Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., et Bal, H. (2012). **WebPIE: A Web-scale Parallel Inference Engine using MapReduce**. *Journal of Web Semantics*, 10:59–75.

[Urbani et al., 2009] Urbani, J., Kotoulas, S., Oren, E., et Harmelen, F. (2009). **Scalable Distributed Reasoning Using MapReduce**. In Bernstein, A., Karger, D., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., et Thirunarayan, K., editors, *The Semantic Web - ISWC 2009 SE - 40*, volume 5823 of *Lecture Notes in Computer Science*, pages 634–649. Springer Berlin Heidelberg.

[Urbani et al., 2013] Urbani, J., Piro, R., van Harmelen, F., et Bal, H. (2013). **Hybrid reasoning on OWL RL**. *Semantic Web*.

[Urbani et al., 2011] Urbani, J., Van Harmelen, F., Schlobach, S., et Bal, H. (2011). **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)*, volume 7031 of *ISWC'11*, pages 730–745, Berlin, Heidelberg. Springer-Verlag.

[Vens et al., 2008] Vens, C., Struyf, J., Schietgat, L., Džeroski, S., et Blockeel, H. (2008). **Decision trees for hierarchical multi-label classification**. *Machine Learning*, 73(2):185–214.

[Vogrincic et al., 2011] Vogrincic, S., et Bosnic, Z. (2011). **Ontology-based multi-label classification of economic articles.** *Comput. Sci. Inf. Syst.*, 8(1):101–119.

[Vora, 2011] Vora, M. N. (2011). **Hadoop-HBase for large-scale data**. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 601–605. IEEE.

[W3C OWL Working Group et al., 2009] W3C OWL Working Group, Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., et Lutz, C. (2009). **OWL 2 Web Ontology Language**.

[Wang et al., 2003] Wang, H., Fan, W., Yu, P., et Han, J. (2003). **Mining concept-drifting data streams using ensemble classifiers**. *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2(1):226—-235.

[Wang, 2006] Wang, J. (2006). **A framework for on-demand classification of evolving data streams**. *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589.

[Wang et al., 2013] Wang, X., Liu, S., Song, Y., et Guo, B. (2013). **Mining evolutionary multi-branch trees from text streams**. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, page 722.

[Wenerstrom et al., 2006] Wenerstrom, B., et Giraud-Carrier, C. (2006). **Temporal Data Mining in Dynamic Feature Spaces**. *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 1141–1145.

[Werner et al., 2014] Werner, D., Silva, N., Cruz, C., et Bertaux, A. (2014). **Using DL-reasoner for hierarchical multilabel classification applied to economical e-news**. In *Proceedings of 2014 Science and Information Conference, SAI 2014*, pages 313–320.

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

[Witten et al., 2005] Witten, I. H., et Frank, E. (2005). **Data Mining: Practical machine learning tools and techniques**. Morgan Kaufmann.

[Wu et al., 2013] Wu, H., Liu, J., Ye, D., Zhong, H., et Wei, J. (2013). **A distributed rule execution mechanism based on MapReduce in sematic web reasoning**. *Proceedings of the 5th Asia-Pacific Symposium on Internetware - Internetware '13*, pages 1–7.

[Wu et al., 2012] Wu, K., et Haarslev, V. (2012). **A Parallel Reasoner for the Description Logic ALC.** In Kazakov, Y., Lembo, D., et Wolter, F., editors, *Description Logics*, volume 846 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Wu et al., 2004] Wu, S.-T. W. S.-T., Li, Y. L. Y., Xu, Y. X. Y., Pham, B. P. B., et Chen, P. C. P. (2004). **Automatic Pattern-Taxonomy Extraction for Web Mining**. *IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*.

[Zablith et al., 2013] Zablith, F., Antoniou, G., D'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., et Sabou, M. (2013). **Ontology evolution: a process-centric survey**. *The Knowledge Engineering Review*, pages 1–31.

[Zhang et al., 2007] Zhang, M.-L., et Zhou, Z.-H. (2007). **ML-KNN: A lazy learning approach to multi-label learning**. *Pattern recognition*, 40(7):2038–2048.

[Zhang et al., 2014] Zhang, M. L., et Zhou, Z. H. (2014). **A review on multi-label learning algorithms**. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.

List of Figures

List of Tables

A

ADAPTIVE PROCESS IMPLEMENTATION DETAILS

This appendix deeply details some implementation parts of the adaptive process proposed in Section 7.4. For clarity proposes, the algorithms presented in this chapter are described using pseudo-code instead of a real language. The presented pseudo-code omits details that are not essential for the algorithm understanding, such as variable declarations, datatypes etc.

Three sections compose the rest of this appendix. The first section A.1 describes in detail the storage of data structures used in the adaptive process using Big Data technologies. The second section A.2 describes in detail some bolts of the Input Change Detection topology. Finally section A.3 present the algorithms used to derive the modification requests in the Modification Derivation Topology.

A.1/ ADAPTIVE PROCESS DATA STORAGE

The specific data used in the Semantic HMC process is stored in a distributed and scalable Big Data store Apache HBase (Fig. A.1). HBase is a non-relational column-oriented database that is built on top of HDFS (Hadoop Distributed File System).

Due to its distributed nature it can handle the data across a distributed environment required to process the stream using high scalable distributed technologies such as Storm or Kafka. Even more, the column-oriented architecture of HBase is a good choice to fit the co-occurrence matrix as it provides efficient random access to each table cell. Beyond the co-occurrence matrix, the HBase database also stores control tables used to control the messages already processed from the KafKa brokers (i.e. the Input Changes, Modification Requests, Model Changes and Applied Modifications) and some cache tables with relevant information of the Classification Model to not overload the Triple-store (i.e. List of beta terms, list of alpha terms, list of hierarchical relations, list of labels and list of terms). One of the disadvantages of using a column-oriented database is getting the high latency when required a batch of data across multiple rows and columns (e.g. the co-occurrence matrix diagonal). The document frequency of all terms is used to detect changes in the labels for each new document from the data stream. To improve the performance, redundant table was created to cache the document frequency (co-occurrence main diagonal) in order to lower the latency access. Also, some in-memory caches are used to store highly used data in short periods of time.

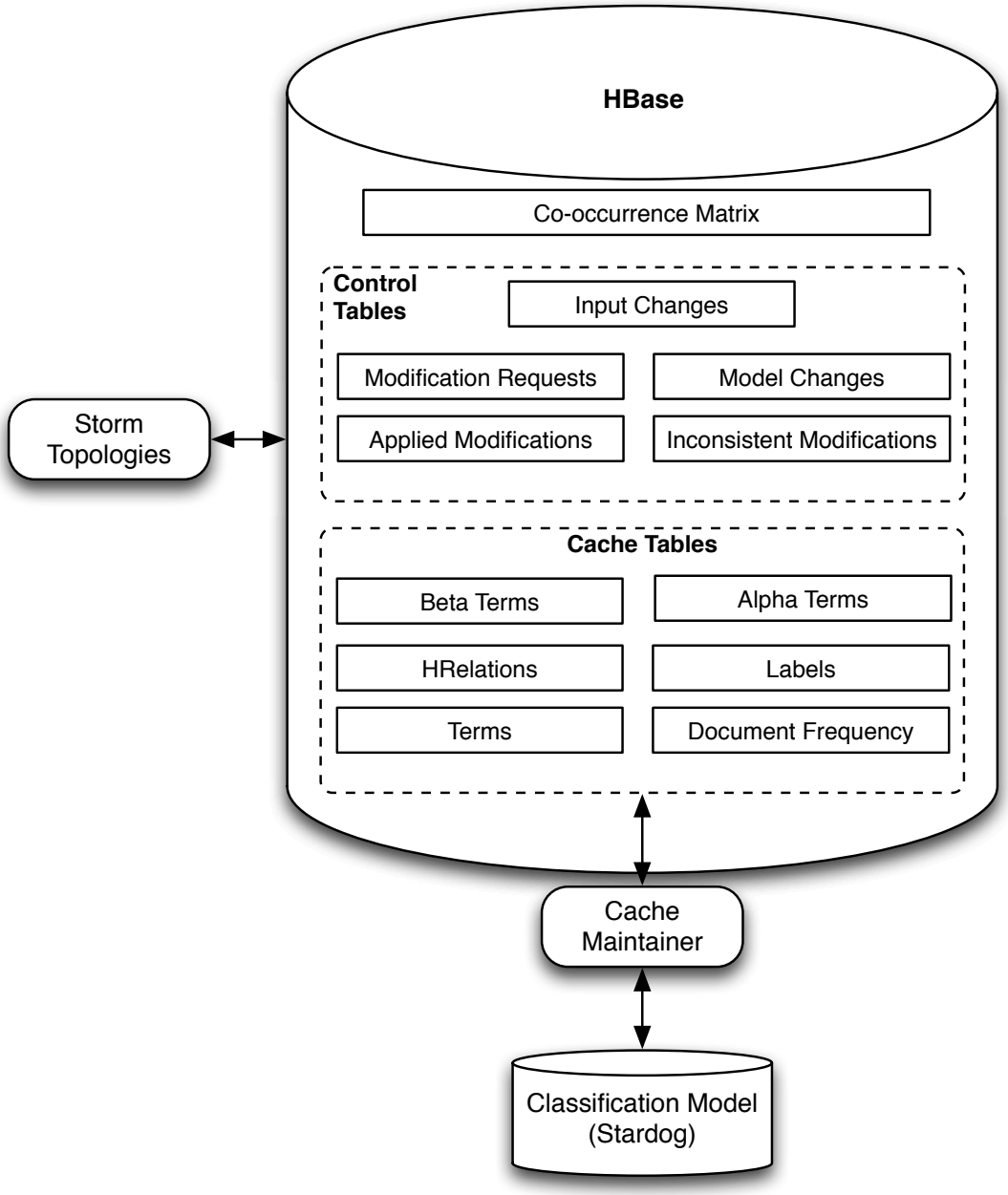


Figure A.1 – Adaptive Process Data Storage

A.2/ Input Change Detection Topology

This section describes in detail two of the bolts from the Input Change Detection topology. They are: (1) The term detection bolt and (2) the Matrix Update bolt.

A.2.1/ TERM DETECTION BOLT

The term detection bolt detects relevant words (terms) to describe a new document from the stream of documents (Fig.A.2). First is verified if that document has been already in the inverted index (i.e. by using a unique ID for each document). If the document doesn't exist in the index it is indexed. At indexation time a parser and a tokenizer are used to extract terms from text documents. Term extraction includes stop-word and synonym detection and lemmatization. The vector of terms of that document is then retrieved from the index and emitted to the Update matrix bolt in order to update the co-occurrence matrix.

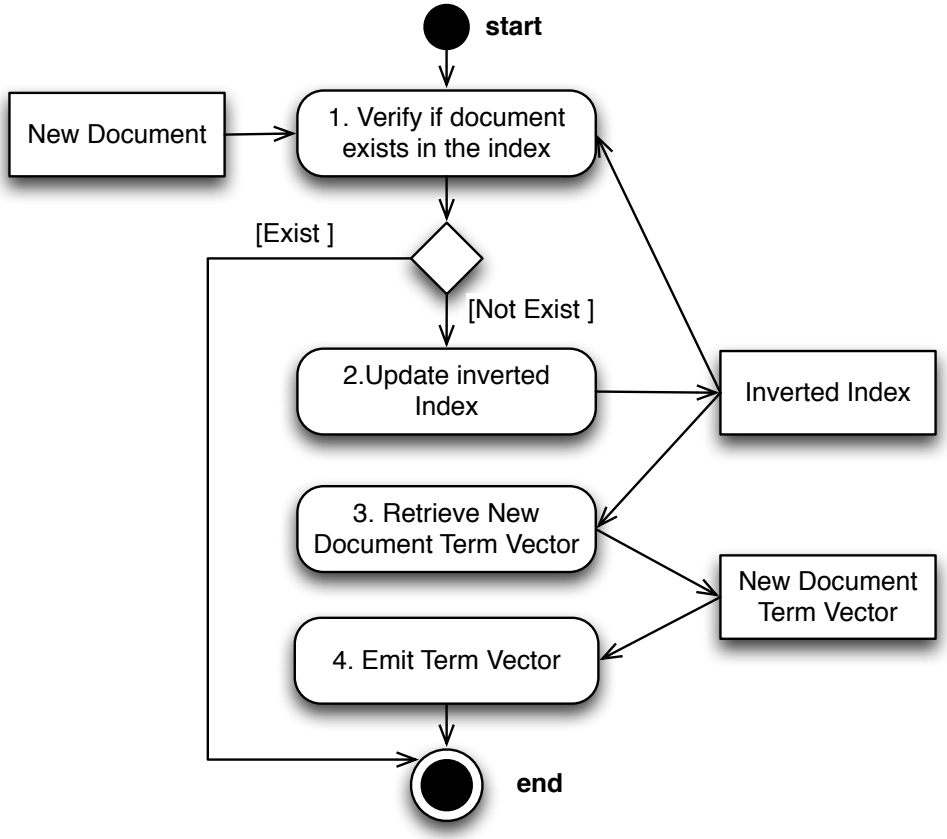


Figure A.2 – Term Detection Bolt

A.2.2/ MATRIX UPDATE BOLT

The matrix update bolt updates the co-occurrence matrix according to each new document doc' emitted from the term detection bolt (Fig. A.3).

Let the set of terms $term_i$ in the new document doc' is defined as:

$$setTerms_{doc'} = \{term_i \in Term : term_i \in doc'\} \tag{A.1}$$

First, for all possible combination of terms in pairs $(term_i,\ term_j)$ where $term_j, term_i \in setTerms_{doc'}$ and $term_i$ can be equals to $term_j$:

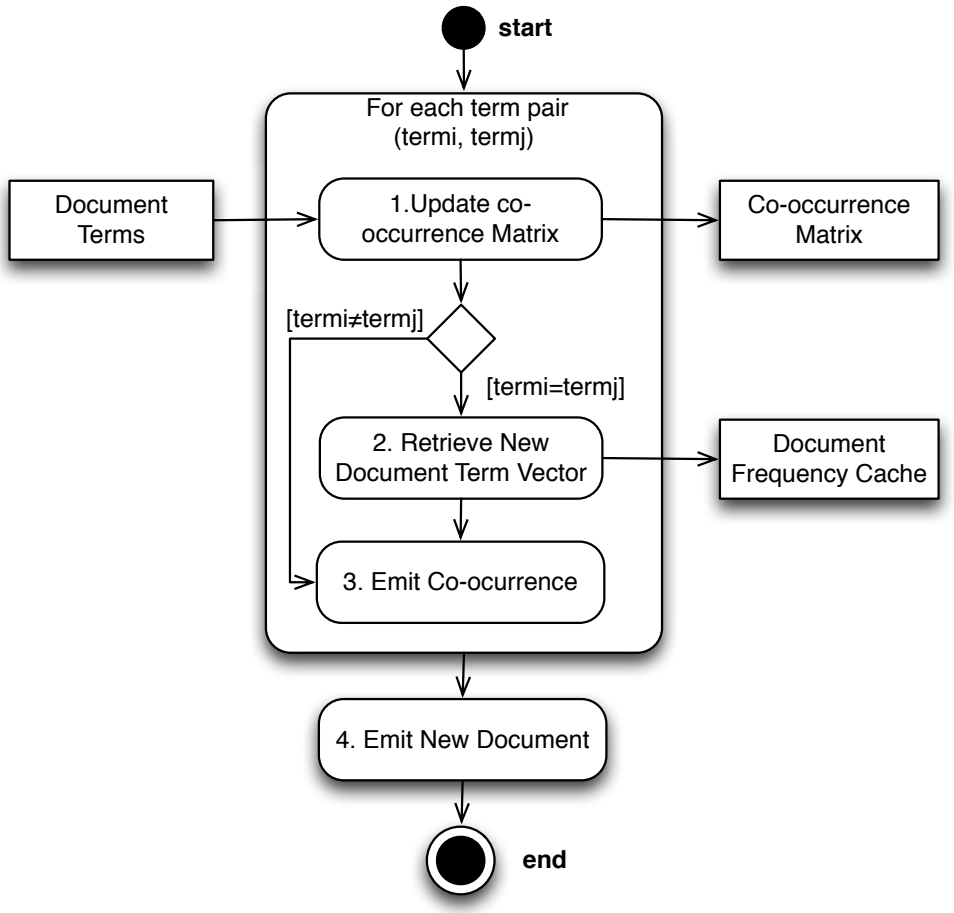


Figure A.3 – Matrix Update Bolt

1. the co-occurrence matrix is updated such as:

$$cfm_{C'}\left(term_i, term_j\right) = cfm_C\left(term_i, term_j\right) + 1 \tag{A.2}$$

2. In the case of $(term_i,\ term_j)$ be equals, the document frequency cache that represents the co-occurrence main diagonal is updated such as:

$$dfCache\left(term_i\right) = dfCache\left(term_i\right) + 1 \tag{A.3}$$

3. The pair of terms $(term_i,\ term_j)$ is emitted to the Matrix Frequency sensor.

After all pairs of terms are processed, the input document with including the term vector is emitted to (1) the New Document sensor and (2) the term detection sensor. Using the term vector emitted by the matrix update bolt instead of using the term vector emitted by the term detection bolt,is assured: (ii) that the matrix is already updated according that document and (ii) is ready to be used by subsequent components (i.e. bolts and topologies).

A.3/ MODIFICATION REQUEST DERIVATION ALGORITHMS

The modification request derivation bolt (Fig. A.4) starts by verify if the input change is already processed. For that, the control table *Input Changes* stored in HBase is used. The *Input Changes* table stores the input change ID of each input in/or already processed and its type. If the input change ID exists in the *Input Changes* table than the input change is already processed. If it does not exist, then the input change ID and its type are added to the *Input Changes* table and the modification requests for that input change are then derived.

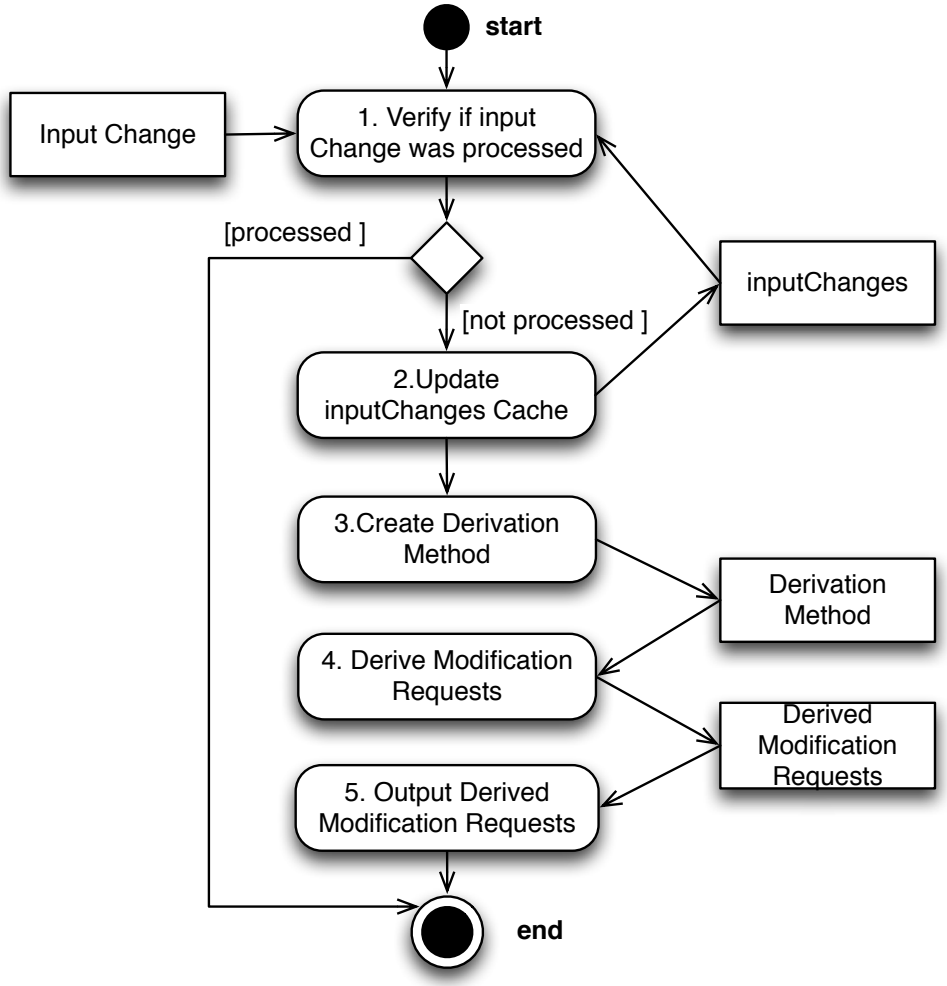


Figure A.4 – Modification Request Derivation Bolt

To derive the modifications request, subclasses of the *DerivationMethod* are used. The *DerivationMethod* class is has a method called *deriveModificationRequest* that given the *inputChange* it derives the modification requests necessary to adapt the classification model. For each input change, one or more *DerivationMethod* objects are created on runtime. The objects are created by relying on the type of the input change as described in Table A.1.

Example, for a input change of the type *newDocument* a *DeriveAddLabel* and a *De-*

Table A.1 – Modification Request Derivation Methods

Derivation Method	Input change type	Derived Modification Request Types
DeriveNewTerm	newTerm	AddTerm
DeriveAddLabel	newDocument	AddLabel
DeriveDeleteLabel	newDocument	DeleteLabel
DeriveAddHRelation	coOcurrence	AddHRelation
DeriveDeleteHRelation	coOcurrence	DeleteHRelation
DeriveAddAlphaTerm	coOcurrence	AddAlphaTerm
DeriveDeleteAlphaTerm	coOcurrence	DeleteAlphaTerm
DeriveAddBetaTerm	coOcurrence	AddBetaTerm
DeriveDeleteBetaTerm	coOcurrence	DeleteBetaTerm
AppliedModificationImpact	appliedModification	Multiple Types

riveDeleteLabel Derivation Methods are created (Fig. A.5). The modification requests are derived for all created derivation methods and outputted to the modification request broker.

The Derivation methods are described in detail later on the following sub-sections.

A.3.1/ DERIVE NEW TERM

The "DeriveNewTerm" class derives modification requests of the type (AddTerm) based on input changes of the type "NewTerm". In the case of the Semantic HMC, verifies if a new term is relevant to be considered in the classification model. It is verified according some frequency requirements limited by thresholds. A new term is derived from a word according the Algorithm A.1 where:

— **Input:** The Input Change of the type "newTerm" where the input data is the detected term (inputTerm); the minimum support (minimumSupport); the minimum document frequency (minimumDocumentFrequency); the maximum document frequency (maximumDocumentFrequency) to be considered a term.

— **Output:** List of Modification Requests of the type "AddTerm" (derivedModificationRequestList).

Listing A.1– Derive New Term

```
1 inputTermDf ← Get the document frequency of inputTerm
2 inputTermSupport ← Get the support of inputTerm
3 If: inputTermSupport >= minimumSupport and minimumDocumentFrequency <
      inputTermDf <= maximumDocumentFrequency
4   Then:
5   derivedModificationRequest ← Create the Modification Request of the
        type "AddTerm" with inputTerm as data
6   Add derivedModificationRequest to the list
        derivedModificationRequestList
7 Return the derivedModificationRequestList
```

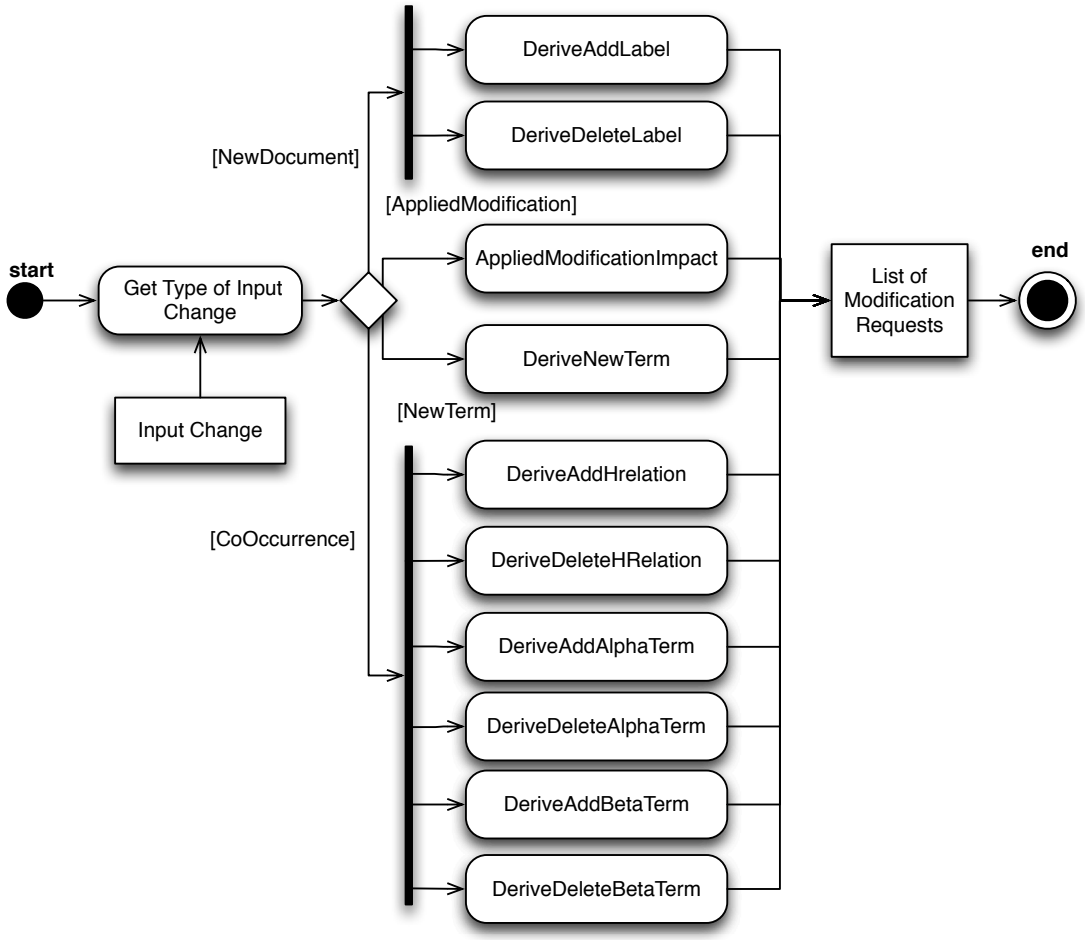


Figure A.5 – Modification Request Derives

A.3.2/ DERIVE ADD LABEL

The "DeriveADDLabel" class verifies if there is any term that meets the requirements to be considered a Label. This task is triggered by a new document in the collection as it impacts the document frequency used to calculate the relevancy of the term to be added to the classification model as a label. The new label is derived according to the Algorithm A.2 where:

— **Input:** The Input Change of the type "newDocument" where the input data is the number of documents (totalNumberOfDocuments) in the collection in that moments; an array with the document frequency for each term (termDocumentFrequencyArray); the threshold to be considered label (labelThreshold).

— **Output:** List of Modification Requests of the type "AddLabel" (derivedModificationRequestList).

Listing A.2– Derive Add Label

```
1 For: termDocumentFrequency in termDocumentFrequencyArray
```

```
    term1 ← get term from termDocumentFrequency
    documentFrequency ← get document frequency from termDocumentFrequency
    If: not exists a Label based on Term term1 and documentFrequency > 0
    then:
      proportion ← documentFrequency / totalNumberOfDocuments
      If:proportion>= labelThreshold
        Then:
        derivedModificationRequest ← Create the Modification Request of
            type "AddLabel" with term1 as parameter
        Add derivedModificationRequest to the list
            derivedModificationRequestList
  Return the derivedModificationRequestList
```

A.3.3/ Derive Delete Label

The "DeriveDeleteLabel" class derives modification requests to remove labels that are no longer relevant. The labels relevancy is calculated based in the proportion of that label in the entire collection. A new document in the collection impacts the proportion value and the label relevancy must be re-evaluated. A label deletion modification request is derived according to the Algorithm A.3 where:

— **Input:** The Input Change of the type “newDocument” where the input data is the number of documents (totalNumberOfDocuments) in the collection in that moments; an array with the document frequency for each label (labelDocumentFrequencyArray); the threshold to be considered label (labelThreshold).

— **Output:** : List of Modification Requests of the type “DeleteLabel” (derivedModificationRequestList)

Listing A.3– Derive Delete Label

```
  For: labelDocumentFrequency in labelDocumentFrequencyArray
    label1 ← get term from labelDocumentFrequency
    documentFrequency ← get document frequency from labelDocumentFrequency
    proportion ← documentFrequency / totalNumberOfDocuments
    If:proportion< labelThreshold
      Then:
      derivedModificationRequest ← Create the Modification Request of type
          "DeleteLabel" with label1 as parameter
      Add derivedModificationRequest to the list
          derivedModificationRequestList
  Return the derivedModificationRequestList
```

A.3.4/ Derive Add Hierarchichal Relation

The "DeriveAddHRelaion" class derives modification requests to add new hierarchical relations between labels. The hierarchical relations are created based in the subsumption method that uses conditional proportions to create the hierarchical relations. Any term detected in a new document will impact the co-occurrence matrix and consequently the proportions used to calculate the hierarchical relations. Modification requests of the type "addHRelation" are derived according to the Algorithm A.4 where:

— **Input:** The Input Change of the type "coOccurrence" where the input data is a pair of terms (term1 and term2); the thresholds used to calculate the hierarchical relations (topTreshold and bottomTreshold)

— **Output:** List of Modification Requests of the type "AddHRelation" (derivedModificationRequestList)

Listing A.4– Derive Add Hierarchichal Relation

```
1  If: term1 equals to term2 and term1 is a label
2    Then:
3    listOfCoOccuredTerms ← get all co-occurred terms with term1
4    For: coOccurredTerm in listOfCoOccuredTerms
5      If: coOccurredTerm is a Label
6        Then:
7        coOcurrenceFrequency ← get coOcurrenceFrequency between term1 and
             coOccurredTerm
8        dfCoOcurredLabel ← get document frequency for coOccurredTerm
9        dfTerm1 ← get document frequency for term1
10       cProportionTerm1CoOccurredTerm ← (coOcurrenceFrequency/
             dfCoOcurredLabel)*100
11       cProportionCoOccurredTermTerm1 ← (coOcurrenceFrequency/ dfTerm1)*
             100
12       If: cProportionTerm1CoOccurredTerm >= topTreshold and
             cProportionCoOccurredTermTerm1 < bottomTreshold and not exist
             hierarchichal relation between term1 and coOccurredTerm
13         Then:
14         derivedModificationRequest Create the Modification Request of
               type "AddHRelation" with term1 and coOccurredTerm as
               parameters
15         Add derivedModificationRequest to the list
               derivedModificationRequestList
16 Return the derivedModificationRequestList
```

A.3.5/ Derive Delete Hierarchichal Relation

The "DeriveDeleteHRelation" class derives modification requests to delete hierarchical relations that are no longer relevant for the classification model. The subsumption method is used to calculate the hierarchical relations between labels as stated in the "DeriveAddHRelaion" class. So any term detected in a new document will impact the co-occurrence matrix and consequently the proportions used to calculate the hierarchical relations. Modification requests of the type "deleteHRelation" are derived according to the Algorithm A.5 where:

— **Input:** The Input Change of the type "coOccurrence" where the input data is a pair of terms (term1 and term2); the thresholds used to calculate the hierarchical relations (topTreshold and bottomTreshold)

— **Output:** List of Modification Requests of the type "DeleteHRelation" (derivedModificationRequestList)

Listing A.5– Derive Delete Hierarchichal Relation

```
1  If: term1 equals to term2
```

```
    Then:
    listOfCoOccuredTerms ← get all co-occurred terms with term1
    For: coOccurredTerm in listOfCoOccuredTerms
      If: coOccurredTerm is a Label and if exist hierarchichal relation
          between term1 and coOccurredTerm
        Then:
        coOcurrenceFrequency ← get coOcurrenceFrequency between term1 and
            coOccurredTerm
        dfCoOcurredLabel ← get document frequency for coOccurredTerm
        dfTerm1 ← get document frequency for term1
        cProportionTerm1CoOccurredTerm ← (coOcurrenceFrequency/
            dfCoOcurredLabel) *100
        cProportionCoOccurredTermTerm1 ← (coOcurrenceFrequency/ dfTerm1) *
            100
        If: cProportionTerm1CoOccurredTerm < topTreshold or
            cProportionCoOccurredTermTerm1 >= bottomTreshold
          Then:
          derivedModificationRequest ← Create the Modification Request of
              type "DeleteHRelation" with term1 and coOccurredTerm as
              parameters
          Add derivedModificationRequest to the list
              derivedModificationRequestList
Return the derivedModificationRequestList
```

A.3.6/ DERIVE ADD ALPHA TERM

The "DeriveAddAlphaTerm" class derives modification requests to add new alpha terms used to calculate alpha rules used to classify the items. The alpha terms are calculated using an alpha threshold that selects the most relevant terms based on their conditional proportion with the corresponding label. So any term detected in a new document that co-occurred with a label will impact the proportions used to calculate alpha terms. Modification requests of the type "addAlphaTerm" are derived according to the Algorithm A.6 where:

— **Input:** The Input Change of the type "coOccurrence" where the input data is a pair of terms (term1 and term2); the threshold used to calculate alpha rules (alphaThreshold)

— **Output:** List of Modification Requests of the type "AddAlphaTerm" (derivedModificationRequestList)

Listing A.6– Derive ADD Alpha Term

```
If: term1 equals to term2 and term1 is a label
  Then:
  listOfCoOccuredTerms ← get all co-occurred terms with term1
  For: coOccurredTerm in listOfCoOccuredTerms
    If: not exist AlphaRelation relation between term1 and coOccurredTerm
      Then:
      coOcurrenceFrequency ← get coOcurrenceFrequency between term1 and
          coOccurredTerm
      dfCoOcurredLabel ← get document frequency for coOccurredTerm
      cProportionTerm1CoOccurredTerm ← (coOcurrenceFrequency/
          dfCoOcurredLabel) *100
      If: cProportionTerm1CoOccurredTerm > alphaThreshold
```

```
11          Then:
12          derivedModificationRequest ← Create the Modification Request of
                type "AddAlphaTerm" with term1 and coOccurredTerm as
                parameters
13          Add derivedModificationRequest to the list
                derivedModificationRequestList
14  Return the derivedModificationRequestList
```

A.3.7/ DERIVE DELETE ALPHA TERM

The "DeriveDeleteAlphaTerm" class derives modification requests to delete alpha terms that are no longer relevant to calculate alpha rules used to classify the items. As described in the "DeriveAddAlphaTerm", alpha terms are calculated using an alpha threshold that selects the most relevant terms based on their conditional proportion with the corresponding label. Modification requests of the type "deleteAlphaTerm" are derived according to the Algorithm A.7 where:

— **Input:** The Input Change of the type "coOccurrence" where the input data is a pair of terms (term1 and term2); the threshold used to calculate alpha rules (alphaThreshold)

— **Output:** List of Modification Requests of the type "DeleteAlphaTerm" (derivedModificationRequestList)

Listing A.7– Derive Delete Alpha Term

```
1   If: term1 equals to term2 and term1 is a label
2     Then:
3     listOfAlphaTerms ← get all alpha terms of term1
4     For: alphaTerm in listOfAlphaTerms
5       coOccurrenceFrequency ← get co-occurrence frequency between term1 and
            alphaTerm
6       dfAlphaTerm ← get document frequency for alphaTerm
7       cProportionTerm1AlphaTerm ← (coOcurrenceFrequency/dfAlphaTerm)*100
8       If: cProportionTerm1AlphaTerm <= alphaThreshold
9         Then:
10        derivedModificationRequest ← Create the Modification Request of
              type "DeleteAlphaTerm" with term1 and alphaTerm as parameters
11        Add derivedModificationRequest to the list
              derivedModificationRequestList
12  Return the derivedModificationRequestList
```

A.3.8/ DERIVE ADD BETA TERM

The "DeriveAddBetaTerm" class derives modification requests to add new beta terms used to calculate beta rules used to classify the items. The beta terms are calculated using two thresholds (alpha and beta) thresholds that selects the relevant terms based on their conditional proportion with the corresponding label. So any term detected in a new document that co-occurred with a label will impact the proportions used to calculate beta terms. Modification requests of the type "addBetaTerm" are derived according to the Algorithm A.8 where:

— **Input:** The Input Change of the type "coOccurrence" where the input data is a pair of terms (term1 and term2); the thresholds used to calculate beta rules (alphaThreshold and betaThreshold)

— **Output:** List of Modification Requests of the type "AddBetaTerm"(derivedModificationRequestList)

Listing A.8– Derive ADD Beta Term

```
1  If: term1 equals to term2 and term1 is a label
2    Then:
3    listOfCoOccuredTerms ← get all co-occurred terms with term1
4    For: coOccurredTerm in listOfCoOccuredTerms
5      If: not exist BetaRelation relation between term1 and coOccurredTerm
6        Then:
7        coOcurrenceFrequency ← get coOcurrenceFrequency between term1 and
             coOccurredTerm
8        dfCoOcurredLabel ← get document frequency for coOccurredTerm
9        cProportionTerm1CoOccurredTerm ← (coOcurrenceFrequency/
             dfCoOcurredLabel) *100
10       If: cProportionTerm1CoOccurredTerm <= alphaThreshold and
             cProportionTerm1CoOccurredTerm >= betaThreshold
11         Then:
12         derivedModificationRequest ← Create the Modification Request of
               type "AddBetaTerm" with term1 and coOccurredTerm as parameters
13         Add derivedModificationRequest to the list
               derivedModificationRequestList
14 Return the derivedModificationRequestList
```

A.3.9/ DERIVE DELETE BETA TERM

The "DeriveDeleteBetaTerm" class derives modification requests to delete beta terms that are no longer relevant to calculate beta rules used to classify the items. As described in the "DeriveAddBetaTerm", beta terms are calculated using an alpha threshold and a beta threshold that select the most relevant terms based on their conditional proportion with the corresponding label. Modification requests of the type "deleteBetaTerm" are derived according to the Algorithm A.9 where:

— **Input:** The Input Change of the type "coOccurrence" where the input data is a pair of terms (term1 and term2); the thresholds used to calculate beta rules (alphaThreshold and betaThreshold)

— **Output:** List of Modification Requests of the type "DeleteBetaTerm" (derivedModificationRequestList)

Listing A.9– Derive Delete Beta Term

```
1  If: term1 equals to term2 and term1 is a label
2    Then:
3    listOfBetaTerms ← get beta terms of term1
4    For: betaTerm in listOfBetaTerms
5      coOccurrenceFrequency ← get co-occurrence frequency between term1 and
           betaTerm
6      dfBetaTerm ← get document frequency for betaTerm
```

```
7       cProportionTerm1BetaTerm ← (coOcurrenceFrequency/ dfBetaTerm)*100
8       If: cProportionTerm1BetaTerm > alphaThreshold or
            cProportionTerm1BetaTerm < betaThreshold
9         Then:
10        derivedModificationRequest ← Create the Modification Request of
              type "DeleteBetaTerm" with term1 and betaTerm as parameters
11        Add derivedModificationRequest to the list
              derivedModificationRequestList
12  Return the derivedModificationRequestList
```

A.3.10/ APPLIED MODIFICATION IMPACT

The "AppliedModificationImpact" class derives modification requests based on the already applied modification requests. Three types of applied modification requests (AddTerm, AddLabel and DeleteLabel) triggers the modification derivation (Figure A.6). To calculate the impact of that modification requests, three subclasses are used to derivate the modification requests for each type: AddTermImpact, AddLabelImpact and DeleteLabelImpact.

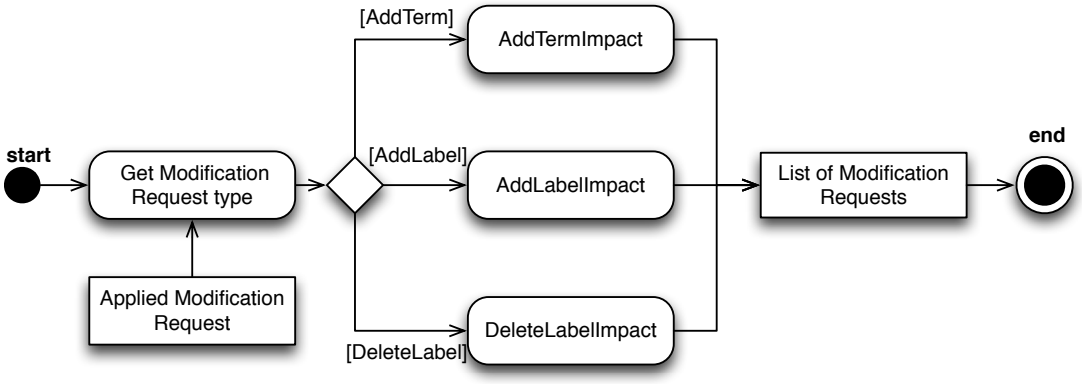


Figure A.6 – Applied Modification Impact

The "AddTermImpact" class derives modification requests of the type "AddLabel" from applied modifications of the type "AddTerm". For that it verifies if the new term has the requirements to be a label. A modification request of the type "AddLabel" is derived according to the Algorithm A.10 where:

— **Input:** The Input Change of the type "appliedModification" where the input data is a modification request (inputModificationRequest) of the type "AddTerm"; the threshold to be considered label (labelThreshold).

— **Output:** List of Modification Requests of the type "AddLabel" (derivedModificationRequestList)

Listing A.10– Add Term Impact

```
1  term1 ← get term from imputModificationRequest
2  documentFrequency ← get document frequency for term1
3  If: not exists a Label based on Term term1 and documentFrequency > 0
```

```
4     Then:
5     proportion ← documentFrequency / totalNumberOfDocuments
6     If: proportion >= labelThreshold
7       Then:
8       derivedModificationRequest ← Create the Modification Request of type
            "AddLabel" with term1 as parameter
9       Add derivedModificationRequest to the list
            derivedModificationRequestList
10  Return the derivedModificationRequestList
```

The "AddLabelImpact" class calculates the modification requests of the type "AddHrelation", "AddAlphaTerm" and "AddBetaTerm" from applied modifications of the type "AddLabel". For that it verifies if the new label has hierarchichal relations using the subsumption method, alpha terms using the Alpha threshold and beta rules using the Alpha and Beta thresholds. Considering that the the input terms (term1 and term2) is the term corresponding to the new label in the applied modification request:

— Modification requests of the type "AddHrelation" are derived according to the Algorithm A.4.

— Modification requests of the type "AddAlphaTerm" are derived according to the Algorithm A.6.

— Modification requests of the type "AddBetaTerms" are derived according to the Algorithm A.8.

The "DeleteLabelImpact" class calculates the modification requests of the type "DeleteHrelation", "DeleteAlphaTerm" and "DeleteBetaTerm" from applied modifications of the type "DeleteLabel". For that it removes all hierarchical relations and all Alpha and Beta terms related with that deleted label.

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

Abstract:

Data generation is exponentially growing, giving increased relevancy to the Big Data phenomenon. Big Data refers to huge datasets characterized by several dimensions (e.g. Volume, Velocity, Variety) that cannot be processed by traditional data processing approaches. Discovering knowledge and value over Big Data is a challenge for business and individual consumers, as it would reduce consumer information overload. While traditional data analysis methods used in the Data Mining field seem to be adapted to process Big Data (e.g. Hierarchical Multi-Label Classification where data items are classified with multiple labels organized in a hierarchy), they often fail to capture the semantics of the data, abstruse understanding and decrease value. Ontologies are a good solution for systems that operate close to the human concept level, and one of the most accepted forms of capturing and describing semantics in the Semantic Web and within corporations. Associating semantic technologies with Big Data processing technologies is a key approach to improve value extraction from Big Data by reducing the gap between the user's perspective and the representation of analysis methods (e.g. a classification model). The aim of this thesis is twofold: (i) to propose a new classification process that hierarchically multi-classifies data items according to a rule-enriched ontology and uses web reasoning, and (ii) to propose approaches to maintain/evolve the classification system in the context of Big Data.

Keywords: Big Data, Ontologies, Semantic Web, Classification

Résumé :

La production de données croit de manière exponentielle donnant plus de crédit au phénomène des "Big Data". Le terme "masses de données" ou "Big Data" fait référence aux jeux de données de très grandes tailles caractérisées par plusieurs dimensions (Volume, Velocité, Variété). Ces données ou flux de données sortent du cadre traditionnel des approches standards de traitement. C'est pourquoi l'extraction de connaissances et leur valeur de cette extraction sont des éléments complexes à traitement à la fois pour les industriels et les consommateurs pour réduire la surcharge d'information. Bien que les méthodes traditionnelles d'analyse de données dans le champ de recherche de la fouille de données semblent être adaptées pour le traitement des données massives (cf. la classification hiérarchique multiétiquette), ces méthodes échouent dans l'identification de la sémantique des données formant une compréhension abstruse et réduisant la valeur. Les ontologies forment une solution pour les systèmes opérant proche des niveaux de conceptualisation humaine, et représente une des formes les plus acceptées pour décrire et capturer la sémantique dans les communautés scientifiques du Web sémantique et les industriels exploitant ces technologies. L'association de ces technologies avec les technologies du traitement des données massives est la clé de l'approche proposée pour améliorer l'extraction de valeur dans le Big Data réduisant ainsi l'écart entre les perspectives des utilisateurs et les méthodes d'analyse (cf. les modèles de classification). Les objectifs de cette thèse sont doubles. D'une part, elle propose une nouvelle méthode de classification pour la multiclassification hiérarchique d'articles selon une ontologie enrichie en règle et l'usage des méthodes du "Web Resoning". D'autre part, elle propose une approche pour maintenir et faire évoluer le système de classification dans le contexte du Big Data et des flux de données non stationnaires.

Mots-clés : Masses de Données, Ontologies, Web Sémantique, Classification

■ École doctorale SPIM - Université de Bourgogne/UFR ST BP 47870 F - 21078 Dijon cedex
■ tél. +33 (0)3 80 39 59 10 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr