Semantic Web Mining

A project report submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Engineering

By

Leon Suraj Dsouza (Roll No. 7058)

Bryceleen Benjamin Dsouza (Roll No. 7057)

Mugesh Tilkraj Nadar (Roll No. 7087)

Under the guidance of

Prof. Swati Ringe

DEPARTMENT OF COMPUTER ENGINEERING

Fr. Conceicao Rodrigues College of Engineering, Bandra (W),

Mumbai - 400050

University of Mumbai

April 28, 2017

This work is dedicated to my family, friends and teachers.

I am very thankful for their motivation and support.

Internal Approval Sheet

CERTIFICATE

This is to certify that the project entitled **"Semantic Web Mining"** is a bonafide work of **Leon Suraj Dsouza (Roll No. 7058), Bryceleen Benjamin Dsouza (Roll No. 7057), Mugesh Tilkraj Nadar (Roll No. 7087)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Engineering**

(Prof. Swati Ringe)

Supervisor/Guide

(Prof. Sunil Surve)	(Dr. Srija Unnikrishnan)
Head of Department	Principal

Approval Sheet

Project Report Approval

This project report entitled by **Semantic Web Mining** by **Leon Suraj Dsouza, Bryceleen Benjamin Dsouza, Mugesh Tilkraj Nadar** is approved for the degree of Bachelor of Engineering in Computer Engineering.

Examiners

1.______________

2.______________

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Bryceleen Dsouza (Roll no. 7057)(sign)______________________

Leon Suraj Dsouza (Roll no. 7058)(sign)______________________

Mugesh Tilkraj Nadar (Roll no. 7087)(sign)______________________

Date :

Acknowledgments

We have great pleasure in presenting the report on **"Semantic Web Mining"**.

We take this opportunity to express our sincere thanks towards the guide Prof. Swati Ringe, C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and the suggestions regarding the line of this work. We enjoyed discussing the work progress with her during our visits to department.

We again would like to thank Prof. Sunil Surve, Head of Computer Dept., Principal and the management of C.R.C.E., Mumbai for the encouragement and providing necessary infrastructure for pursuing the project.

We also thank all non-teaching staff for their valuable support, to complete our project.

We would also like to appreciate the support we got from our families and friends which is one of the main factors which kept us motivated while working on this project.

Abstract

Linked Data is used in the Web to create typed links between data from different sources. Connecting diffused data by using these links provides new data which could be employed in different applications. Association Rules Mining (ARM) is a data mining technique which aims to find interesting patterns and rules from a large set of data. In this project, the problem of applying association rules mining using Linked Data in centralization approach has been addressed -i.e. arranging collected data from different data sources into a single dataset and then apply ARM on the generated dataset. Firstly, a number of challenges in collecting data from Linked Data have been presented, followed by applying the ARM using the dataset of connected data sources. Preliminary experiments have been performed on this semantic data showing promising results and proving the efficiency, robust, and useful of the used approach.

With the introduction and standardization of the semantic web as the third generation of the Web, this technology has attracted and received more human attention than ever and thus the amount of semantic web data is constantly growing. These semantic web data are a rich source of useful knowledge for feeding data mining techniques. Semantic web data have some complexities, such as the heterogeneous structure of the data, the lack of exactly-defined transactions, the existence of typed relationships between entities etc. One of the data mining techniques is association rule mining, the goal of which is to find interesting rules based on frequent item-sets. In this paper we propose a novel method that considers the complex nature of semantic web data and, without end-user involvement and any data conversion to traditional forms, mines association rules directly from semantic web datasets at the instance level. This method assumes that data have been stored in triple format (*Subject*, *Predicate*, and *Object*) in a single dataset.

Keywords: Semantic Web, Association Rules, Semantic Web Mining, Data Mining, Information Retrieval, SWApriori, Data Mining, Association Rules Mining, Linked Data Mining, Frequent Itemset Mining, Linked Data Query.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

ARM - Association Rule Mining

ARs - Association Rules

SWApriori - Semantic Web Apriori

ILP - Inductive logic programming

FP - Frequent Pattern

LiDDM - Linked Data Data Miner

RDF - Resource Description Format

SPARQL - SPARQL Protocol and RDF Query Language

Chapter 1

Introduction

Linked Data and Association rule mining is a practical technique used to achieve Semantic Web goals. Linked Data is defined as "a set of best practices for publishing and connecting structured data on the Web". Hence many natural features of semantic web data, exists also in Linked Data. Extending the scope, we can say that data mining research from traditional data to semantic web data along with its Linked Structure could help in discovering and mine richer and more useful knowledge.

ARM, one of the main data mining techniques, tries to find frequent itemsets and based on these frequent itemsets, generates interesting association rules (ARs).

In trying to apply ARM to semantic web data, problems to face and differences compared with traditional data are as follows:

Heterogeneous data structure: Traditional data mining algorithms work with homogeneous datasets in which instances are stored in a well-ordered sequence and each instance has predefined attributes. But in semantic web data, data are heterogeneous. This means that specific category/domain instances (such as people, cars, drugs and etc.) based on one ontology or multiple ontologies may have different attributes.

No exact definition of transactions: In conventional information systems, data is stored in databases using predetermined structures, and by using these structures it’s possible to recognize transactions and thus extract them from the dataset. Then traditional association rule mining algorithms work on these transactions.

Multiple relations between entities: Traditional ARM algorithms, in order to generate large itemsets, consider only entities' values and suppose there is only one type relation between entities (for example bought together). But in semantic web data, there are multiple relations between entities. In fact predicates are relations between two entities or between one entity and one value. These different relations must be considered in the ARM process.

In an algorithm, named SWApriori, has considered the above challenges and without the end user involvement, mines ARs directly from a single semantic web dataset. So we could state that the mentioned problem has been solved. The problems and challenges of collecting desired data, from different Linked Data sources, and the ARM of it has been investigated. Thus the suggested approach collects data from various data sources and connects them so that all data appear as a single and central dataset and then uses existing methods to mine ARs from the generated dataset.

1.1 Objective

An algorithm, named SWApriori, has been proposed which has considered the above challenges and without the end user involvement, mines ARs directly from a single semantic web dataset. The problem statement includes collecting and connecting them so that all data appear as a single and central dataset and then using existing methods to mine ARs from the generated dataset.

1.2 Scope

•Preliminary experiments have been performed on this semantic data showing promising results and proving the efficiency, robust, and usefulness of the used approach.

•Knowledge representation of the web.

•Integration of databases in the knowledge web.

•Scalability of knowledge-intensive web services

Chapter 2

Literature Survey

2.1 Semantic Web

Web 3.0 is a Read/Write/Execute Web. In Semantic Web, the data is presented in such a manner that it is not only human readable but also machine processable. The problem which the world was facing before the Web 3.0 was as follows :

a. Whenever users wanted to search anything on the internet say a user wants to see the shoes within the range of 2000-5000 She enters “SHOES WITHIN 2000-5000RS” in the text box of a search engine. On a traditional Web, the search engine will display the links of all those pages in which only the keywords entered in the query will be present. From there, the user has to explore every link and see whether the information is related to his context

b. In the past many machine-learning algorithms have been successfully applied to traditional datasets in order to discover useful and previously unknown knowledge. Although these machine learning algorithms are useful, the nature of semantic web data is quite different from traditional data. The majority of previous semantic web data mining work focus on clustering and classification. Some of these work are based on inductive logic programming or ILP which uses ontology encoded logics.

The Semantic Web when necessary will try to put in formalism like RDF rules, OWL etc. to solve complex problems, puts a great emphasis on describing data and the relationship between data and is less focused on the user experience part

<u>Characteristics of Web 3.0</u>

1. Intelligence: Using various Artificial Intelligence techniques like neural networks, machine learning, rule mining etc. applications can work intelligently with the data available to them
2. Personalization: Individual preferences are considered when doing tasks like information processing or search.
3. Interoperability: Deals with reuse of information that facilitates exchange of new knowledge.

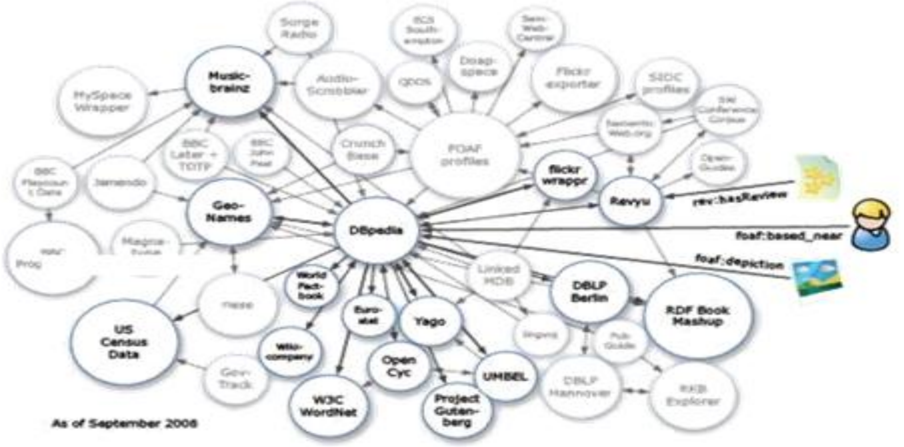


Figure 2.1: Ontologies on the Web

2.2 Introduction

The ARM was first introduced with the aim of finding frequent itemsets and generating rules based on these frequent itemsets. Many ARM algorithms have been proposed which deal with traditional datasets. These algorithms are classified into two main categories: Apriori based and FP-Tree based. These algorithms usually work with discretized values, but later an evolutionary algorithm was introduced for mining quantitative ARs from huge databases without any need to data discretization.

As will be seen, semantic web dataset contents are convertible to graph. Other related approaches in ARM are the use of frequent sub-graph and frequent sub-tree techniques for pattern discovery from graph structured data. The logic behind these algorithms is to generate a tree/graph based on existing transactions and then mine the generated tree/graph. Although these methods are interesting, they are not appropriate for our work, because in semantic web data there is no exact definition of transactions, and also after converting dataset contents to graph, each vertex of the graph, independent of its incoming link, is not replicated in the whole graph more than once. On the other hand, graph vertices are unique and thus discovering sub-graph/sub-tree redundancy is not possible.

Not all graph-based approaches are based on sub-graph techniques. An algorithm has been introduced that inputs data into a graph structure and then by a novel approach without the use of sub-graph redundancy, mines ARs from these data. This work is not useful for our problem because the algorithm finds only maximal frequent itemsets instead of all frequent itemsets and also, like other traditional ARM algorithms, this algorithm works only with well-defined transactions.

2.3 Arm Algorithm

An algorithm has been introduced that by using a mining pattern which the end-user provides, mines ARs from semantic web data. This algorithm uses dynamic and graph-based structure data that must be converted to well-structured and homogeneous datasets so that traditional ARM methods can use them. To convert data, users must state the target concept of analysis and their involved features by a mining pattern following an extended SPARQL syntax. This work, similarly to other related approaches in mining ARs from semantic web data, requires that the end-user be familiar with ontology and dataset structure.
A recent work on semantic data mining is LiDDM which is a piece of software that enables to do data mining on linked data. LiDDM working process is as follows:

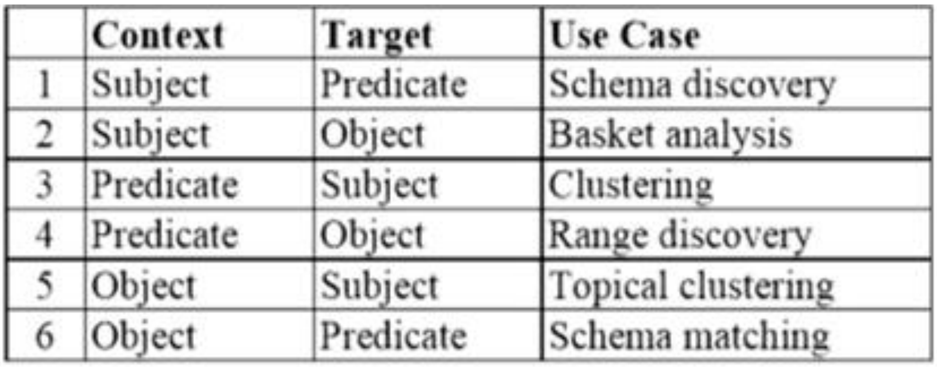
1. The software requires semantic web datasets obtained by using user-defined SPARQL queries. Then clubs the results and converts them in tabular format.
2. Next pre-processing has to be done on these data and then the traditional data mining algorithms is applied.

The actual problem with the LiDDM is the end user involvement during the mining process. Thus one needs to be aware of the ontologies and dataset structure. Thus he needs to guide the mining process step by step.

Another approach similar to the LiDDM is the RapidMiner semweb plugin. This technique applied the data mining technique on the semantic web and linked data along with reformatting set-valued data, such as converting multiple values of a feature into a simple nominal feature to decrease the number of generated features and thus the approach scales well. Even in this process the end user involvement is required.
In RDF structure, a triple defines every data statement names and is identified with three values: subject, predicate and object. In order to generate transactions, it is possible to use one of these three values to group transactions (transaction identifier) and use one of the remaining values as transaction items. Six different combinations of these values along with

Thei’r usage are shown in Table 1. For example, grouping triples by predicate and using objects for generating transactions has usage in clustering. This approach eliminates one part of triples parts and doesn't consider it in mining process that isn't interested.

SPARQL-ML [4] is another approach to mining semantic web data that provides special statement as an extension to SPARQL query language to create and learn a model for specific concept of retrieved data. It applies classification and regression techniques to data, but other data mining techniques such as clustering and ARM are not covered by this approach. Another limitation is that this technique is applicable only on those datasets for which the SPARQL endpoints support SPARQL-ML, which is currently not very widespread. Our proposed algorithm can deal with all kinds of datasets and ontologies.



	Context	**Target**	**Use Case**
1	Subject	Predicate	Schema discovery
2	Subject	Object	Basket analysis
3	Predicate	Subject	Clustering
4	Predicate	Object	Range discovery
5	Object	Subject	Topical clustering
6	Object	Predicate	Schema matching

Figure 2.2: Combinations of Triple parts

Chapter 3

Preliminaries

3.1 Association Rules

Frequent itemset mining and association rule induction are powerful methods for so-called market basket analysis, which aims at finding regularities in the co-occurred items, such as sold products or prescript biomedical drugs. The problem of mining association rules was first introduced in 1993.

Let us denote each item with I_i, thus $I = \{I_1,I_2,\dots,I_m\}$ is set of all items which sometimes called the item base. Each transaction T_i is a subset of I and based on transactions we define database as collection of transactions denoted by $D = \{T_1,T_2,\dots,T_n\}$. Each itemset (S) is a non-empty subset of I and an association rule (R) is a rule in the form of X→Y which both X and Y are itemsets. This rule means that if in a transaction the itemset X occurs, with certain probability the itemset Y will appears in the same transaction too. We call this probability as confidence and call X as rule antecedent and Y as rule consequent

3.2 Support of an itemset:

The absolute support of the itemset S is the number of transactions in D that contain S.. Likewise, the relative support of S is the fraction (or percentage) of the transactions in D which contain S.

More formally, let S be an itemset and U the collection of all transactions in D that contain all items in S. Then

$$Sup_{abs}(S) = |U|$$

$$Sup_{rel} = (|U|/|D|)*100\%$$

For brevity we call $Sup_{rel}(S)$ as Sup(S).

3.3 Confidence of an Association Rule:

The confidence of an association rule R = X→Y is the support of the set of all items that appear in the rule divided by the support of the antecedent of the rule. That is,

$$Conf(R)=(Sup(\{X \cup Y\}) / Sup(X)) * 100\%$$

Rules are reported as strong association rules if their confidence reaches or exceeds a given lower limit (minimum confidence, to be specified by a user). In this paper, we call this association rules as strong association rules.

3.4 Support of an Association Rule:

The support of the rule is the (absolute or relative) number of cases in which the rule is correct. For example in the association rule R: A,B→C, the support of R is equal to support of {A,B,C}.

The support of the rule is the (absolute or relative) number of cases in which the rule is correct.

For example in the association rule : , → , the support of is equal to support of { , , }.

3.5 Frequent Itemsets:

Itemsets with greater Support than a certain threshold, so-called minimum support are frequent itemsets. The goal of frequent itemset mining is to find all frequent itemsets.

3.6 Maximal Itemsets

A frequent itemset is called maximal if no superset is frequent, that is, has a support exceeding the minimum support.

Chapter 4

Proposed System

4.1 Mining Association Rules from Linked Data

In order to mine ARs from Linked Data, desired data has been collected from a few datasets (DBPedia,Factbook etc.) and then converts them to a singled and central dataset and finally mines ARs by using the proposed algorithm. Next sections show the datasets used and acquired results.

This part describes the used methodology of collecting desired data.

4.1.1 Selecting Domain

The generated ARs quality depends on the input dataset quality and how much the provided data is being specialized in a domain, the generated rules are more specific and more useful respectively. Also because generalized data are diffused, the MinSup value needs to be low so that these generalized rules appear in the result. Currently we selected Country as data domain.

4.1.2 Selecting Datasets

In next step we select suitable datasets that have appropriate data about countries. As mentioned earlier, there are two approaches to automatically select suitable datasets. Our strategy for selecting datasets is to consider a dataset as the start point and extract desired data from it, afterward select another dataset based on the extracted data and finally traverse these new selected datasets. This operation will be continued until a special criterion is being satisfied (for e.g. traverse at most 5 datasets). The selected dataset for start point is very important. In addition to have many links to other datasets, the start point dataset must have suitable data about selected domain. DBPedia has been selected as start point dataset since it has many links to other datasets.

4.1.3 Connecting Datasets

There are two ways for connecting related data of different datasets. The first way is to acquire suitable data independently and then append them to each other or connect them by using common attribute. The second way is to collect data from a dataset and then traverse another datasets by using objects from triples of collected data. The latter method requires an explicit relation among datasets- i.e. object of a triple refers to an entity (a subject) that exists in another dataset. These explicit relations are grouped into two relations' groups: more information relations and equivalent relations. More information relations are those relations that state one attribute of an entity which has been defined and exist in another dataset. Equivalent relations are those relations that state two entities in two different datasets are equivalent. The most common equivalent relation is owl:sameAs. It's clear that using more information relations don't help to collect suitable data about a domain since these relations

only show external attributes causing irrelevant entities in the collection process. This is why only equivalent relations have been used to collect suitable and relevant data after collecting primary data from start point dataset.

4.1.4 Ontology Mapping

As mentioned before, due to existence of multiple data sources, ontology mapping must be applied on data so data become coherent. There are many approaches that perform ontology mapping [7-9]. For this project, ontology mapping will been done manually so that data become suitable for ARM by integrating subjects and predicates- i.e. detecting identical subjects or predicates with different names, and also to scale numerical objects.

4.1.5 Duplicated Data

This problem can be removed by selecting the best and the most valid data and eliminate other duplicates. Thus, the issue can be solved by selecting data from specialized dataset and eliminating other duplicated data.

4.1.6 Last step

After collecting desired data and mapping ontology and removing duplicated data, the data have to be placed in a single and central dataset with a unified ontology.

4.2 Data Structure

The algorithm inputs a set of triples (subject, predicate and object).

4.2.1 The Brute Force Structure

For the purpose of storing data in main memory, the simplest and the most efficient way is to use a cuboid (3D array) as data structure, in such a way that the first dimension stores source (subject), the second stores destination (object) and the third stores relation (predicate) between source and destination. Each cuboid entry value is 0 or 1. If the (i,j,k)th entry value is equal to 1, this means there is a relation with k type from ith entity (as subject) to jth entity (as object). Although a cuboid structure is very fast and easy to use it requires a large amount of memory space.

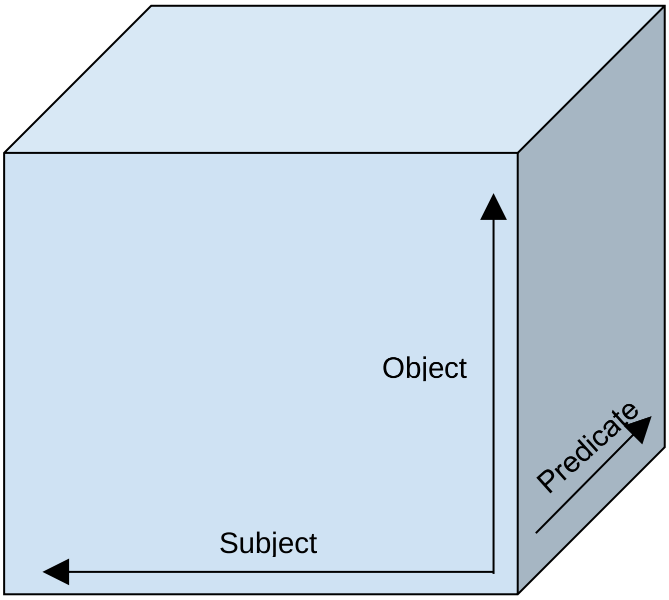


Figure 4.1: Initial Cube Data Structure for Triples

4.2.2 The Modified LinkedList-like Structure

With static memory allocation not an option, since the number of triples in a file given by the user might exceed the size we have allocated on each dimension of the cube,an alternative is to use a linked list data structure with slight modifications.Thus, we make a trade-off between the runtime taken and the space occupied. To store each object scheme (predicates and subjects that are connected to the object), there is an ObjectInfo class with these attributes:

1- Object ID: Object identifier

2- A Linked List whose entries have two parts:

  a. Predicate ID: Predicate identifier

  b. Subjects List: pointer to a list that contains subjects which refer to this Object ID with this Predicate ID.

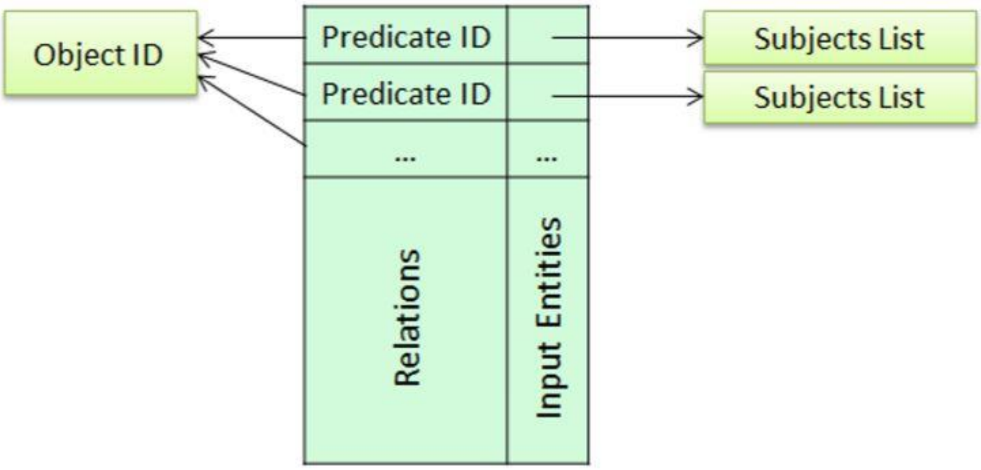


Figure 4.2: Structure of ObjectInfo Class

Using this data structure policy, triples are grouped based on objects and the algorithm defines an ObjectInfo instance and then specifies them based on each predicate, what other subjects refer to this object. The need for grouping is to increase the mining process speed based on the proposed algorithm. Finally there is a list that has entries equal to the objects count. Each entry of this list refers to one of the ObjectInfo instances. This algorithm, in addition to entity values, considers relations between entities in the ARM process. Thus here each Item not only is equal to an entity but also each Item consists of an Entity (Object) and a Relation (Predicate) that is connected to that object. To store each Item there is an Item class that has ObjectID and PredicateID attributes. The figure below shows the structure of the Item class.

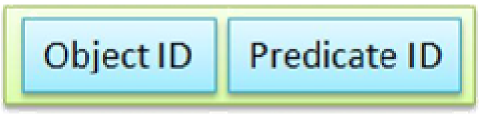


Figure 4.3 : Item Structure

Generating ARs is based on large itemsets. Each itemset is non-empty set of Items. In order to storing generated (candidate/large) itemsets, there is an Itemset class that contains these attributes:

1- List of Items: that holds L items (≥ 2).

2- Support: number of subjects that refer to all Items via correspond predicates.

The Itemset is large if Support is equal to or greater than MinSup value. The figure below shows the structure of the Itemset class

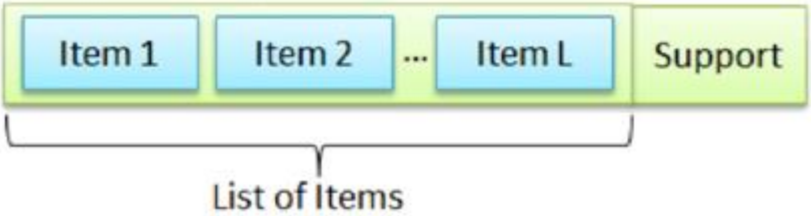


Figure 4.4 : Itemset Structure

It was said that ARs are constructed from Items and each rule has only one Item in the consequent part. To store generated ARs, there is a Rule class that contains these attributes:

1- List of Items as Antecedent

2- An Item as Consequent

3- Rule Confidence

4- Rule Support

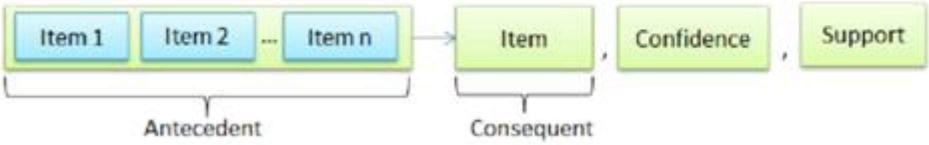


Figure 4.5 : Rule Structure

4.3 Algorithm

The proposed algorithm name is SWApriori. The algorithm workflow is as follows. After traversing triples, discretizing data and eliminating triples with less frequent subject, predicate or object, all triples parts (subjects, predicates and objects) must be converted to numerical IDs. This conversion is done to increase the mining process speed, because this algorithm focuses on comparing entities and relations and clearly comparing two numbers is faster than comparing two literals. After converting data into numerical values, the Generate2LargeItemsets algorithm is called by SWApriori algorithm and generates 2-large itemsets and feeds them to the main algorithm. Then the SWApriori algorithm launches to make larger itemsets. Finally the GenerateRules algorithm generates ARs based on these large itemsets. These algorithms are as follows:

Algorithm 1 (SWApriori) is the main algorithm that after calling Generate2LargeItemsets and generating 2-large itemsets, launches to generate L-large itemsets (≥ 3) and finally calls GenerateRules to generate ARs.

```
Algorithm 1. Mining association rules from semantic web data
SWApriori(DS, MinSup, MinConf)
Input:
   DS: Dataset that consists of triples (Subject, Predicate, and Object)
   MinSup: Minimum support
   MinConf: Minimum confidence
Output:
   AllFIs: Large itemsets
   Rules: Association rules
Variables:
   FIs9, Candidates: List of Itemsets
   IS10, IS1, IS2, IS3: Itemset (multiple items)
   ObjectInfoList: List of ObjectInfo
Begin
   Traverse triples and discretize objects
   Delete triples which their subject, predicate or object has frequency less than MinSup value
   Convert input dataset's data to numerical values
   Store converted data into ObjectInfo instances
   ObjectInfoList = ObjectInfo instances
   FIs = AllFIs = Generate2LargeItemsets(ObjectInfoList, MinSup)
   L = 1
   Do
      L = L + 1
      Candidates = null;
      For each IS1, IS2 in FIs
         If IS1[1..L-1].ObjectID = IS2[1..L-1].ObjectID and
         IS1[1..L-1].PredicateID = IS2[1..L-1].PredicateID Then
            IS3 = CombineAndSort(IS1,IS2)
            Candidates = Candidates ∪ IS3
         End If
      End For
```

```
32.     FIs = null;
33.     For each IS in Candidates
34.       If Support(IS) $\geq$ MinSup AND all subsets of IS are large Then
35.         FIs = FIs $\cup$ IS
36.       End If
37.     End For
38.     AllFIs = AllFIs $\cup$ FIs
39.   While (FIs.Lenght > 0)
40.   Rules = GenerateRules(AllFIs, MinConf)
41.    Return AllFIs, Rules
42. End
```

Let us explain the SWApriori algorithm in detail. This algorithm accepts a dataset that contains triples along with minimum support (MinSup) and minimum confidence (MinConf) values as input parameters. The preprocess step is done in lines 15 to 19. In line 20 all 2-large itemsets are generated by calling Generate2LargeItemsets algorithm. The loop between lines 22 to 39 generates all large itemsets and will continue until generating larger itemsets is no longer possible. In each iteration of this loop, all large itemsets with length of L are verified and new candidate itemsets with length of L+1 are generated. Each loop iteration (lines 25-31), uses previous loop iteration results which have been stored in FIs. Line 25 states that all large itemsets with length of L must be compared two by two, and this comparison is done in lines 26 and 27. If two large itemsets with length of L are combinable (their L-1 first items are equal) they will be combined by the CombineAndSort function and will generate a new candidate itemset with length of L+1. The items of this new candidate itemset are sorted by Object ID and then by Relation ID. In line 29 the new candidate itemset is added to the candidate itemsets collection. After generating all candidate itemsets with length of L+1, in lines 33 to 35 all large itemsets are selected from the candidate itemsets collection and then added to the large itemsets collection (FIs). Finally line 37 adds generated large itemsets with length of L+1 to the collection of all frequent itemsets (AllFIs). After generating all possible large and frequent itemsets, the ARs are generated by calling GenerateRules in line 40.

Calculating the exact time complexity of SWApriori algorithm is not easy, because as the number of L increases, the number of generated frequent itemsets first is increased and then is decreased. In the worst case SWApriori is in the order of O(I2L3), if I is the number of large itemsets and L is the length of the largest itemset.

Algorithm2

(Generate2LargeItemsets) is called by SWApriori and by traversing all ObjectInfo instances generates all possible object sets that have length two. Finally if many subjects by two arbitrary predicates refer to both objects of the generated object set, the object set along with these two predicates are identified as a 2-large itemset. The pseudocode of this algorithm is shown in below:

```
Algorithm 2. Generating 2-Large itemsets from ObjectInfo instances
Generate2LargeItemsets(ObjectInfoList, MinSup)
Input:
  ObjectInfoList: List of ObjectInfo instances
  MinSup: Minimum support value
Output:
  LIS: List of Itemsets with two in length
Variables:
  Ob1, Ob2: ObjectInfo
  SS111, SS2: Subject Set //subjects that refer to an object via special predicate
  R112, R2: Value corresponds to RelationID //refers to predicates
Begin
For each Ob1, Ob2 in ObjectInfoList
  For each R1 in Ob1.Relations
    For each R2 in Ob2.Relations
      SS1 = R1.SubjectsList
      SS2 = R2.SubjectsList
      IntersectionCount = IntersectCount(SS1, SS2)
      If IntersectionCount $\geq$ MinSup Then
        LIS = LIS $\cup${(Ob1.ObjectID + R1), (Ob2.ObjectID + R2)}
      End If
    End For
  End For
End For
Return LIS
End
```

This algorithm accepts all ObjectInfo instances and minimum support value as input parameters. ObjectInfo instances store objects information as it was shown in Figure 3. Each ObjectInfo instance is related to an object and reveals what subjects by what predicates refer to the object. This algorithm generates all possible 2-large itemsets. In line 13 all ObjectInfo instances are traversed and compared two by two. In lines 14 and 15 all input relations (Relation attribute of ObjectInfo class) of these two instances are traversed and compared two by two. In line 16 the list of all subjects that refer to object Ob1 by predicate R1 is extracted from Ob1.R1.SubjectsList and then added to SS1 list. This operation will be repeated for Ob2 and R2 and the result is added to SS2 in line 17. In line 18 an intersection is taken from SS1 and SS2. This intersection reveals what subjects refer to both objects by both predicates. If the intersection count is equal to or greater than MinSup value, both objects along with their corresponding predicates generate a 2-large itemset. This algorithm finishes when all objects, for each their incoming predicates, are compared to each other.

The complexity of Generate2LargeItemsets is in the order of $O(B^2R^2S)$, if B is the number of large entities (large ObjectInfo instances), R is the maximum number of relations of ObjectInfos and S is the maximum number of subjects concerned to an ObjectInfo (S is the required time for intersecting by using hash set)

Algorithm3 (GenerateRules) traverses all generated large itemsets and proceeds to generate candidate rules with one item in the consequence part. If the candidate rule confidence is equal to or greater than MinConf value, the rule is identified as strong rule. The pseudocode of this algorithm is shown below:

```
1. Algorithm 3. Generating association rules based on large itemsets
2. GenerateRules(AllFIs, MinConf)
3. Input:
4.     AllFIs: All large itemsets
5.     MinConf: Minimum confidence
6. Output:
7.     Rules: Association rules
8. Variables:
9.     IS13: Itemset
10.    Itm: Item
```

```
11.   Consequent: Item that is appeared in rule consequent part
12.   Antecedent: List of Items that are appeared in rule antecedent part
13. Begin
14. For each IS in AllFIs
15.   For each Itm in IS
16.     Consequent = Itm
17.     Antecedent = IS – Consequent
18.     Confidence = Support(IS) ÷ Support(Antecedent)
19.     If Confidence ≥ MinConf Then
20.       Rules = Rules ∪ (Antecedent, Consequent)
21.     End If
22.   End For
23. End For
24.   Return Rules
25. End
```

This algorithm accepts frequent and large itemsets and a minimum confidence value as input parameters. In line 14, the large itemsets are selected one by one. In line 15 all Items of the selected large itemset are traversed. Line 16 and 17 construct a rule body based on the selected large itemset and selected item, and then line 18 calculates the confidence of this new rule. Line 19 verifies the rule confidence. If the confidence value is equal to or greater than MinSup value, that is this rule is a strong rule and then it is added to the strong rules collection in line 20. Notice that the algorithm in line 16 selects only one Item as consequent part.

The complexity of GenerateRules is in the order of O(IL), if I is the number of all large itemsets and L is the length of the largest itemset.

4.4 Project Design

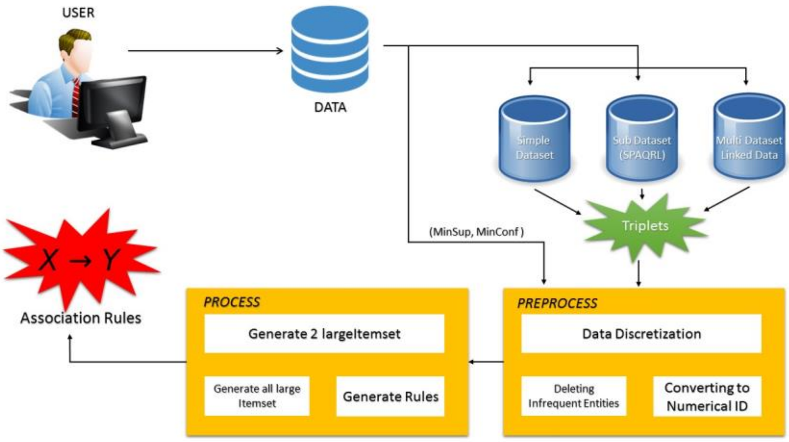


Figure 4.6 : Flow structure

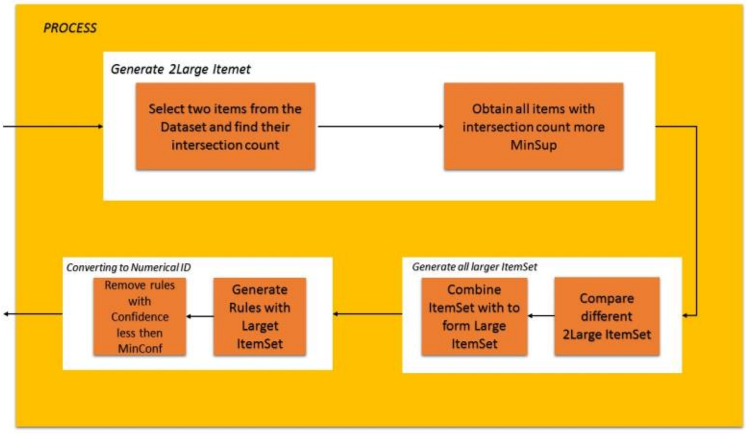

PROCESS
Generate 2Large Itemet
Select two items from the Dataset and find their intersection count
Obtain all items with intersection count more MinSup
Converting to Numerical ID
Remove rules with Confidence less then MinConf
Generate Rules with Larget ItemSet
Generate all larger ItemSet
Combine ItemSet with to form Large ItemSet
Compare different 2Large ItemSet

Figure 4.7 : Preprocess

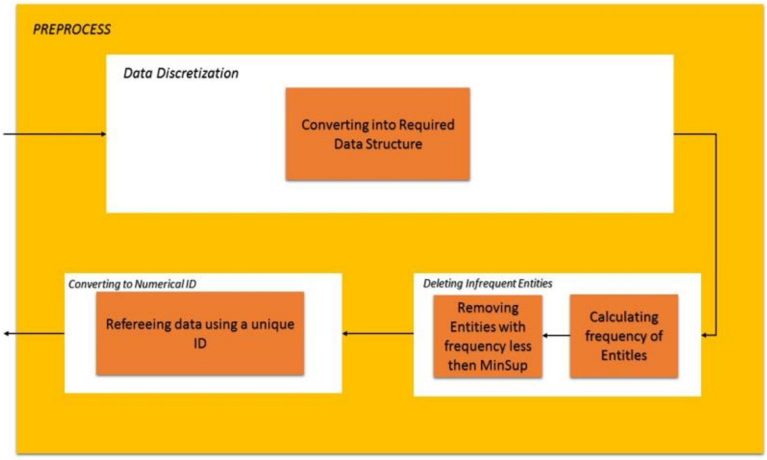

PREPROCESS
Data Discretization
Converting into Required Data Structure
Converting to Numerical ID
Refereeing data using a unique ID
Deleting Infrequent Entities
Removing Entities with frequency less then MinSup
Calculating frequency of Entitles

Figure 4.8 : Process

4.5 Project Architecture

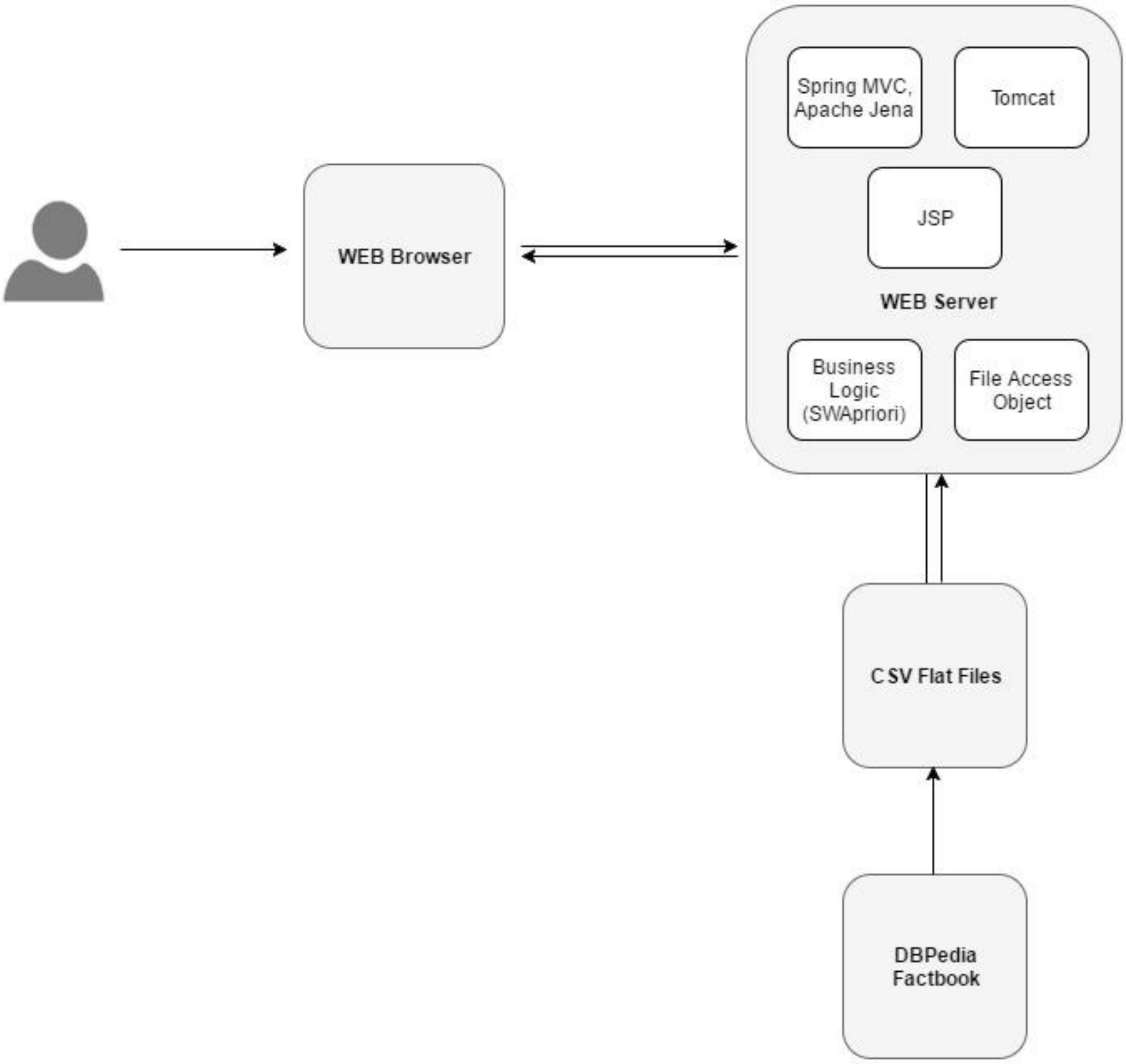


Figure 4.9 : Architecture

4.6 Sequence Diagram

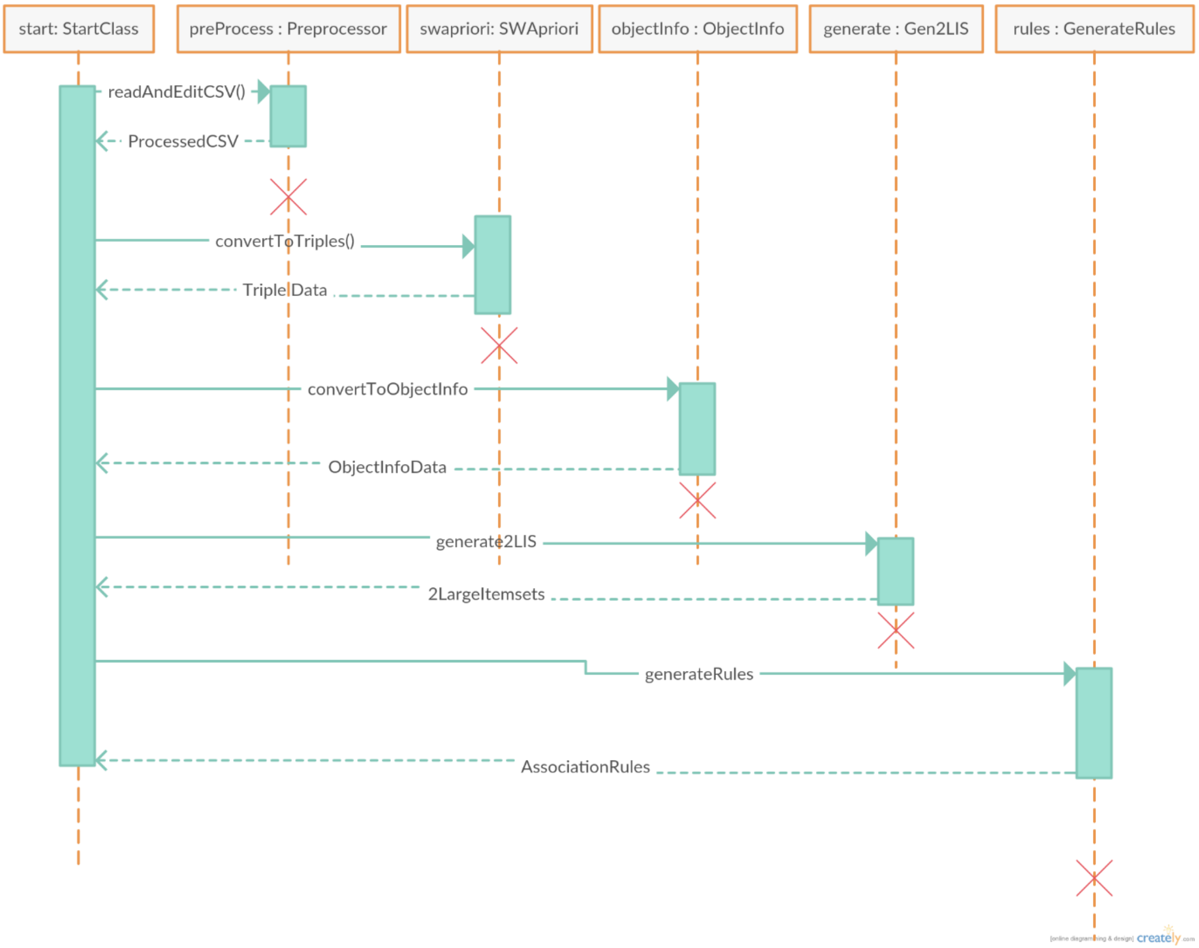


Figure 4.10: Sequence

Chapter 5

Implementation Details

5.1 Software Requirements

1. Netbeans IDE 7.0 or higher
2. Apache Jena Framework
3. Protégé.
4. Owl Web Ontology Language
5. Java (and Spring MVC)
6. Maven
7. Web browser

5.2 Minimum Software Requirements

Inter Side Processor	RAM	Disk Space
Intel Celeron or AMD Athlon 64	2 GB	10GB

5.3 Recommended Software Requirements

Inter Side Processor	RAM	Disk pace
Intel Core i3 or AMD - 1.7 GHz	6 GB	20GB

IMPLEMENTATION

For implementing the proposed solution, we will first collect the datasets from the various data sources. These sources include DBpedia, Factbook etc.Once the data is obtained they need to be converted into the structure format as per the proposed solution mentioned in the section 4

The first step mentioned is extracting the data and developing the dataset from it. Then we apply the SWApriori algorithm and obtain the Association rules. This rules in then used for obtaining output.

Flow:

1. The user uploads a CSV file containing the data amongst which he wants the system to find association rules through an upload button on the browser and clicks on the submit button.
2. Once the user clicks the submit button, the CSV file undergoes preprocessing in order to remove infrequent triples and descriptive attributes and data types of various entities.
3. The remaining triples are then converted into the ObjectInfo structure mapping both, predicates and subjects to their respective objects.
4. The ObjectInfo structure is then passed to a class which gives us a list of LargeItemsets of size 2 by using support as a category to classify the given Itemset as large where the support is the intersection count of the list of subjects of that object for a given predicate.
5. From the 2LIS, the algorithm keeps generating larger itemsets until no larger itemsets are possible.
6. Finally the last large itemset gives us the rules and corresponding confidence is found out.
7. The rules are written out into another CSV file along with the confidence and then displayed to the user on the browser in the form of a neatly arranged table containing the columns, Antecedent, Consequent and Confidence.

Chapter 6

Results

1. Large Itemsets Count results

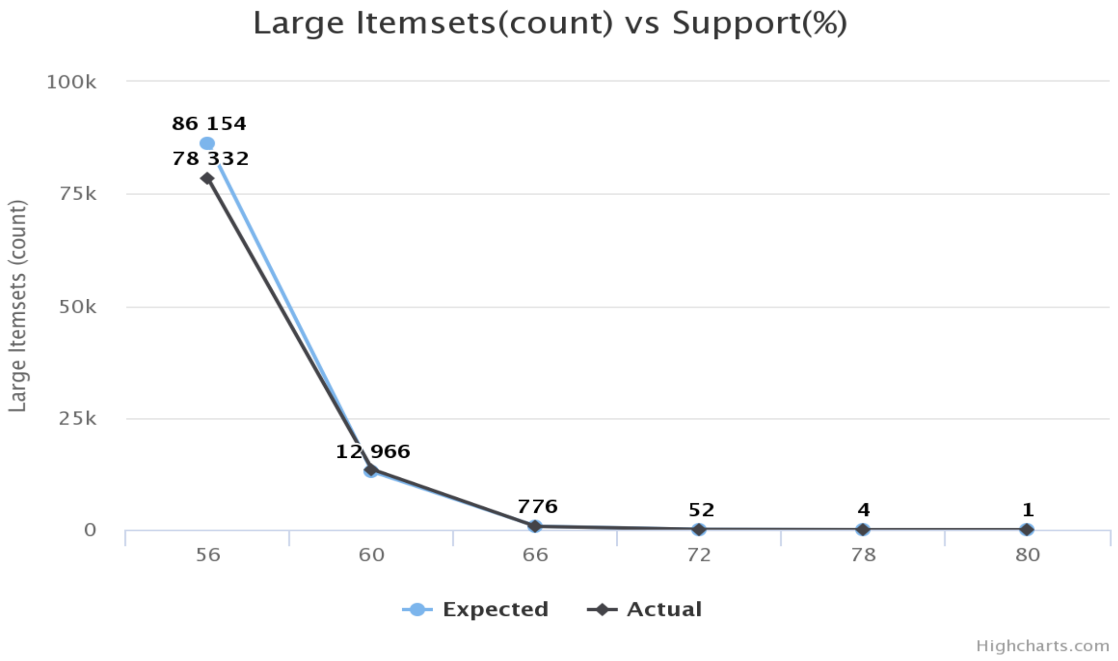


Figure 6.1: Large Itemset vs Support

2. Avg Confidence vs Support

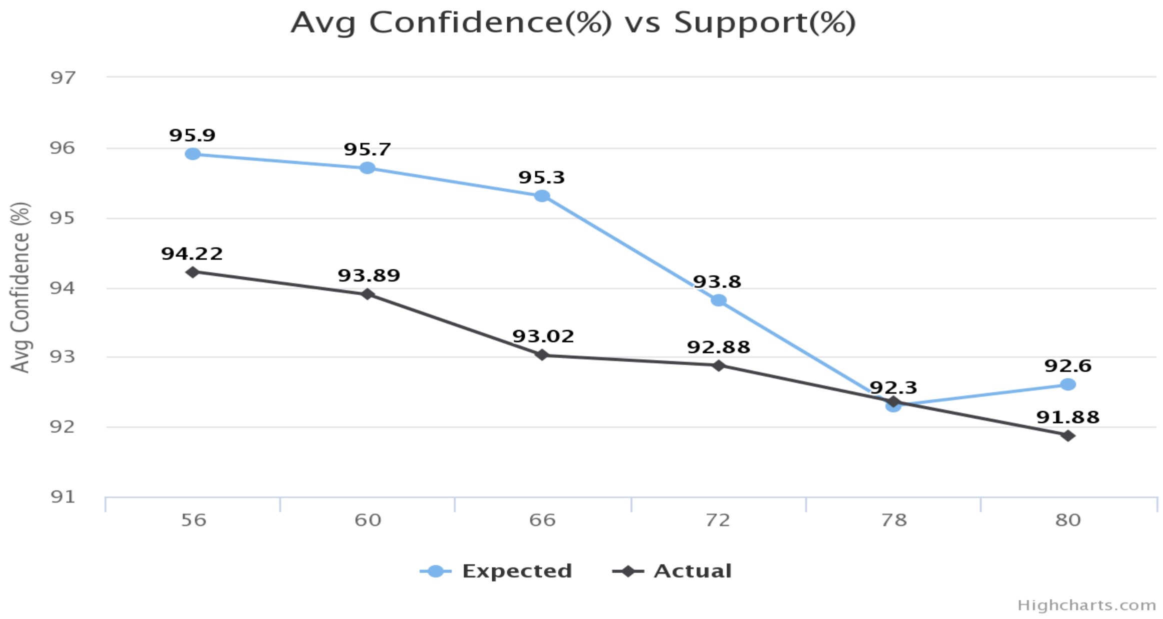


Figure 6.2: Average Confidence v/s Support

Instead of showing the results in a tabular format, we’ve chosen to display it in the form of graph in this report in order to make it visually appealing. As can be deduced from the graphs, our results are very close to the expected one with really small errors when compared to it.

Screenshots

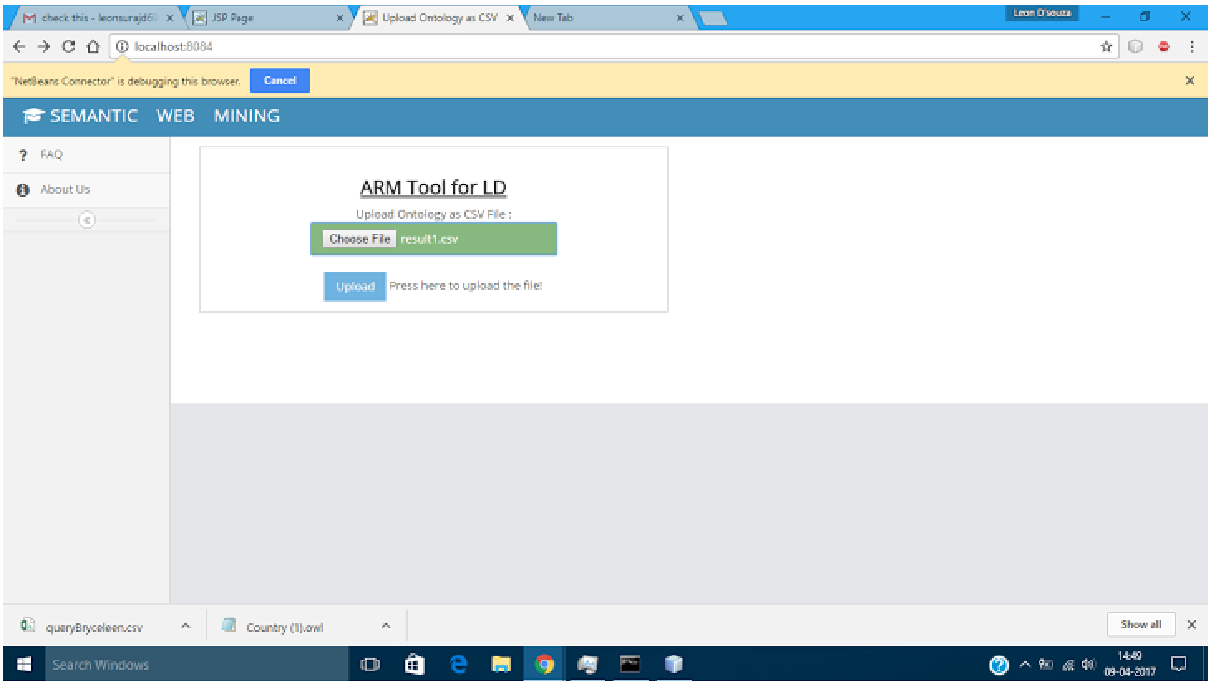
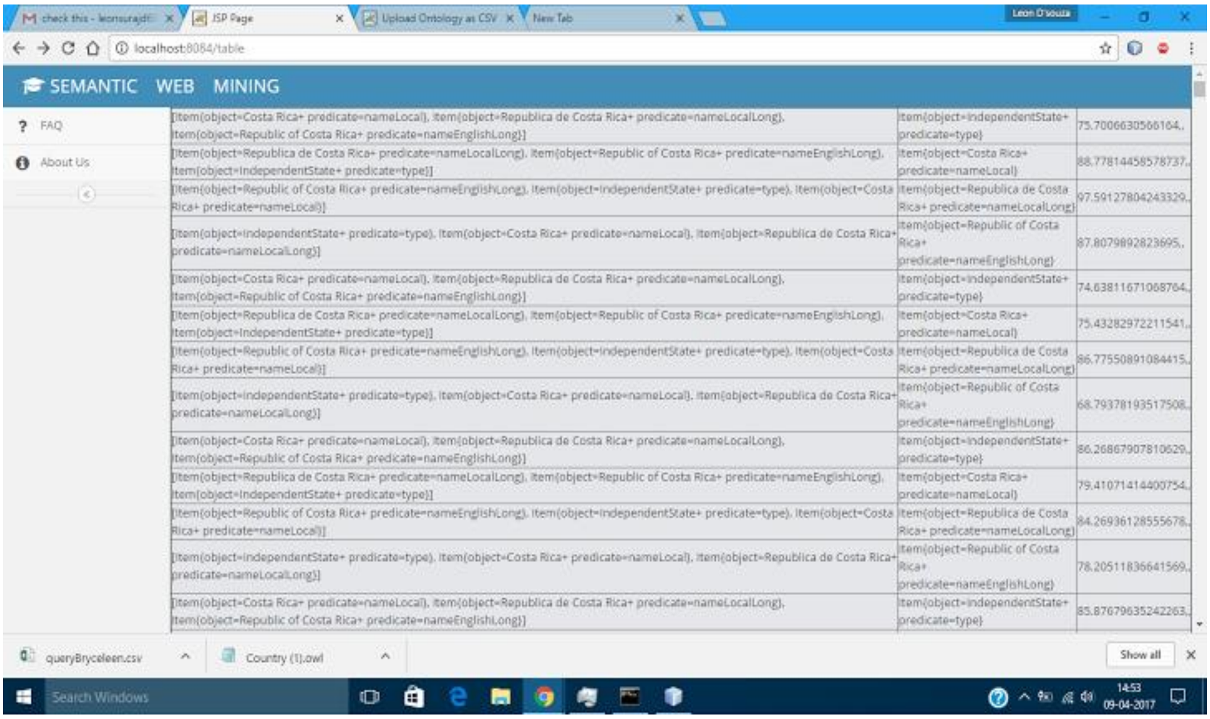


Figure 6.3: GUI



[Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}]	Item{object=IndependentState+ predicate=type}	75.7006630566104,
[Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}, Item{object=IndependentState+ predicate=type}]	Item{object=Costa Rica+ predicate=nameLocal}	88.77814458578737,
[Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}, Item{object=IndependentState+ predicate=type}, Item{object=Costa Rica+ predicate=nameLocal}]	Item{object=Republica de Costa Rica+ predicate=nameLocalLong}	97.59127804243329,
[Item{object=IndependentState+ predicate=type}, Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}]	Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}	87.8079892823695,
[Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}]	Item{object=IndependentState+ predicate=type}	74.63811671068764,
[Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}, Item{object=IndependentState+ predicate=type}]	Item{object=Costa Rica+ predicate=nameLocal}	75.43282972211541,
[Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}, Item{object=IndependentState+ predicate=type}, Item{object=Costa Rica+ predicate=nameLocal}]	Item{object=Republica de Costa Rica+ predicate=nameLocalLong}	86.77550891084415,
[Item{object=IndependentState+ predicate=type}, Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}]	Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}	68.79378193517508,
[Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}]	Item{object=IndependentState+ predicate=type}	86.26867907810629,
[Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}, Item{object=IndependentState+ predicate=type}]	Item{object=Costa Rica+ predicate=nameLocal}	79.41071414400754,
[Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}, Item{object=IndependentState+ predicate=type}, Item{object=Costa Rica+ predicate=nameLocal}]	Item{object=Republica de Costa Rica+ predicate=nameLocalLong}	84.26936128555678,
[Item{object=IndependentState+ predicate=type}, Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}]	Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}	78.20511836641569,
[Item{object=Costa Rica+ predicate=nameLocal}, Item{object=Republica de Costa Rica+ predicate=nameLocalLong}, Item{object=Republic of Costa Rica+ predicate=nameEnglishLong}]	Item{object=IndependentState+ predicate=type}	85.87679635242263,

Figure 6.4: Output

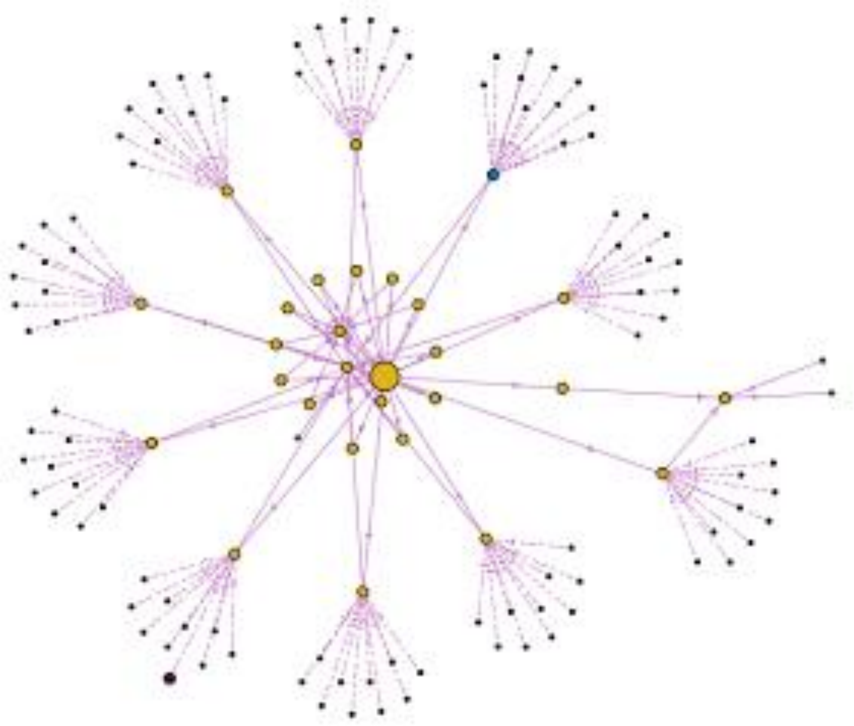


Figure 6.5: Ontology Structure

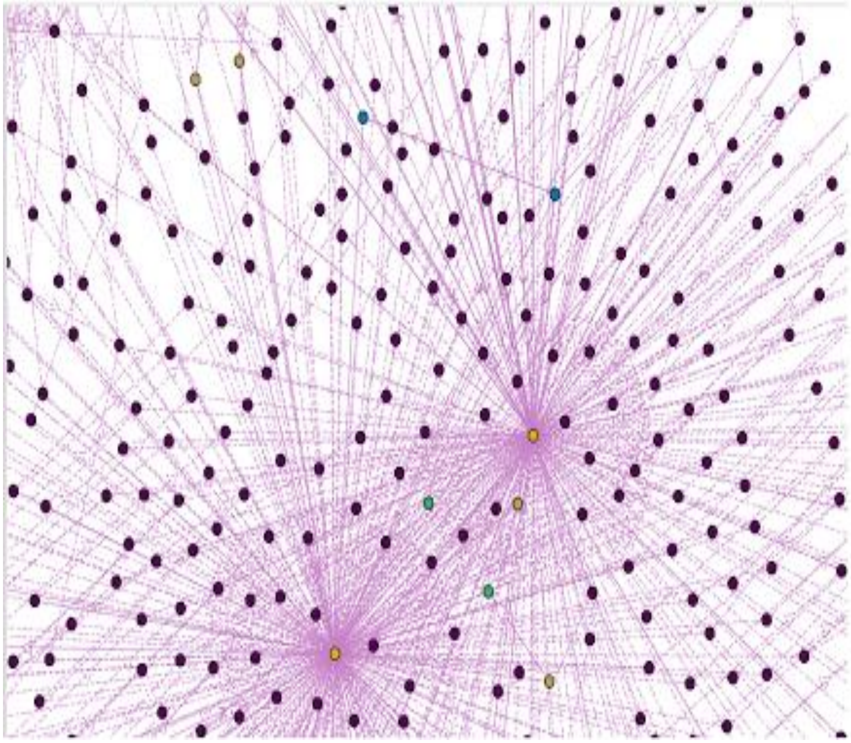


Figure 6.6: Ontology

Chapter 7

Conclusion and Future Enhancements

In this paper the importance of mining association rules from semantic web data and related challenges was discussed and a new algorithm was proposed that can deal with and solve these challenges. The proposed algorithm name is SWApriori. This algorithm can discover ARs from semantic web datasets directly, particularly generalized datasets which do not belong to special domain. On the other hand the algorithm can handle all kinds of datasets and ontologies regardless of the dataset domain. The rationale behind the developed method is that the algorithm after receiving a semantic web dataset, proceeds by applying ontology semantics (if needed), data discretization, infrequent data elimination and finally converting triples to numerical IDs. At the first level of the ARM process, the algorithm identifies large objects and then generates all large objects sets of length two. Afterwards the algorithm generates 2-large itemsets through large objects sets regardless to transactions. Here each itemset consists of multi items and each item consists of an object and a predicate (relation). After generating all 2-large itemsets, the algorithm continues by generating L-large itemsets (□≥3) based on (L-1)-large itemsets. Finally ARs are discovered by using all large itemsets. Discovered rules contain only one item in the consequent part.

The most sensible features of the proposed algorithm are as follow:

- There is no need to convert semantic web data to traditional data. The input data are used in their original format, triple format, by the algorithm.
- Traditional association rule mining algorithms (like Apriori) are not used.
- There is no need for a transactions concept: in fact with semantic web datasets, there is no transaction
- There is no need for user involvement in the mining process: here the main user role is to provide input dataset and the values of MinSup and MinConf. That is the end-user doesn't need to be aware of dataset and ontology structure. But if the end-user wishes, he/she can filter input dataset by SPARQL language or extend the input dataset by assembling linked datasets. Also the end-user can tune the pre-process and the post-process of the proposed algorithm by using ontology concepts for smarter results.

- The algorithm considers different relations between entities: in this algorithm each item consists of an object and a predicate. These items are considered in the mining process.
- The algorithm handles heterogeneous data structures.
- The proposed algorithm can be easily adapted to use other binary combinations of subjects, predicates and objects in generating ARs

And there are some drawbacks in the proposed method as:

- The proposed method is not intelligent enough to involve meaning of data (provided by ontology) in the mining process to guide the process intelligently and generate only interested and useful rules.
- If the content of the input data is general and the end-user does not filter it, the number of generated ARs would be enormous and a large part of them may be uninteresting.
- If we do have commodity hardware, because of the complexity of the algorithm, both space complexity as well as time complexity, it becomes difficult to run once the main memory on the machine is fully consumed.

In fact, this algorithm is very similar to the Apriori algorithm but with different strategies and based on these strategies, the algorithm is able to mine ARs from specialized and generalized semantic web datasets directly. We believe that this kind of learning will become important in the future and will have an effect on the machine learning research area especially the area of semantic web research. The acquired results show the usefulness of the proposed method.

As future work, we intend to apply this method to linked data [24] as the algorithm collects data which are related to an entity from multi datasets automatically, and mine ARs from these connected data. This work require such concepts as ontology alignment, otology mapping, broken links and etc.

Another possible topic for future work is to use encoded knowledge in the ontologies in order to filter the generated association rules.

Other interested possibilities are to cluster entities based on generated frequent itemsets. It is possible to apply this clustering to subjects or objects.

Usually there are hierarchically structure and inheritance rules involved in ontologies. Considering these concepts will lead to a reduction in the generated association rules and improve obtained results quality.

CODES

```
//CSVPreprocesssing
package org.crce.interns.service.impl;

import au.com.bytecode.opencsv.CSVReader;
import au.com.bytecode.opencsv.CSVWriter;
import au.com.bytecode.opencsv.CSVParser;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.stream.Stream;


public class CSVPreprocessor {

    String fileName;

    public CSVPreprocessor(String fileName) {
        this.fileName = fileName;
    }

    public void readAndEditCSV(String fileName) {
        CSVReader csvReader;
        List<String[]> listOfRows = new ArrayList(1000);
        try {
            csvReader = new CSVReader(new FileReader(fileName),
CSVParser.DEFAULT_SEPARATOR, CSVParser.DEFAULT_QUOTE_CHARACTER, 1);
            String[] row;

            while ((row = csvReader.readNext()) != null) {
                        for (String row1 : row) {
                        //remove URLs from the file
                        row = removeURLs(row);
                        }
                        if (!Stream.of(row).anyMatch(x -> x == null || "".equals(x))) {
                        listOfRows.add(row);
                        }
            }
        } catch (FileNotFoundException e) {
            System.err.println(e.getMessage());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }

        writeCSV(listOfRows);

    }

    public String[] removeURLs(String[] row) {
        String remainingString;
        for (int i = 0; i < row.length; i++) {
            //remove URLs
            remainingString = row[i].substring(row[i].indexOf("#") + 1);
            row[i] = remainingString;
            remainingString = row[i].substring(row[i].indexOf("/") + 1);
            row[i] = remainingString;
        }
        return row;
    }

    public void writeCSV(List<String[]> listOfRows) {
        try {
            try (CSVWriter writer = new CSVWriter(new FileWriter(fileName))) {
                        writer.writeAll(listOfRows);
            }
        } catch (IOException ex) {
            Logger.getLogger(CSVPreprocessor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```
//SWapriori
package org.crce.interns.service.impl;
```

```
import au.com.bytecode.opencsv.CSVReader;
import com.google.common.collect.ArrayListMultimap;
import com.google.common.collect.Multimap;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;
import java.util.Map;
import java.util.Set;

public class SWApriori {

    ArrayList<Triple> listOfTriples = new ArrayList(1000);

    public ArrayList<Triple> convertToTriples(String fileName) throws
FileNotFoundException, IOException {
        CSVReader csvReader;
        try {
            csvReader = new CSVReader(new FileReader(fileName));
            String[] row;

            while ((row = csvReader.readNext()) != null) {
                Triple t = new Triple(row[0], row[1], row[2]);
                listOfTriples.add(t);
            }
        } catch (FileNotFoundException e) {
            System.err.println(e.getMessage());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
        return listOfTriples;
    }
    public ArrayList<Triple> convertToNumericalValues(ArrayList<Triple> listOfTriples,
String fileName) throws IOException {
        CSVReader csvReader;
        Set<String> setOfWords = new LinkedHashSet(1000);
        try {
            csvReader = new CSVReader(new FileReader(fileName));
            String[] row;

            while ((row = csvReader.readNext()) != null) {
                for (String row1 : row) {
                    setOfWords.add(row1);
                }
            }
             int count = 0;
            Map<String, Integer> mapOfWords = new LinkedHashMap(1000);
            for (String word : setOfWords) {
                count = count + 1;
                mapOfWords.put(word, count);
            }
                    } catch (FileNotFoundException e) {
            System.err.println(e.getMessage());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
        return listOfTriples;
    }

    public Map<String, Multimap> convertToObjectInfoStructure(ArrayList<Triple>
listOfTriples) {
        Map<String, Multimap> ObjectInfo = new LinkedHashMap(1000);
        LinkedHashSet<String> objectSet = new LinkedHashSet(1000);
           listOfTriples.forEach((t) -> {
            objectSet.add(t.object);
        });
        objectSet.forEach((object) -> {
            Multimap<String, String> multiMap = ArrayListMultimap.create();
            listOfTriples.stream().filter((t1) -> (t1.object.equals(object))).forEachOrdered((t1) ->
{
                multiMap.put(t1.predicate, t1.subject);
            });

            ObjectInfo.put(object, multiMap);
        });
        return ObjectInfo;
    }
}


//Generate2LIS
package org.crce.interns.service.impl;

import org.crce.interns.model.Item;
import com.google.common.collect.ArrayListMultimap;
import com.google.common.collect.Multimap;
import java.util.ArrayList;
import java.util.Collection;
import java.util.LinkedHashSet;
import java.util.Map;
import java.util.Set;

public class Generate2LargeItemSets {

    public Multimap<Item, Item> generate2LargeItemsets(Map<String, Multimap>
ObjectInfo, double minSupport, ArrayList<Triple> listOfTriples) {
        LinkedHashSet<String> objectSet = new LinkedHashSet(1000);
        int yesCount = 0;
        Multimap<Item, Item> largeItemSet = ArrayListMultimap.create();
        listOfTriples.forEach((t) -> {
            objectSet.add(t.object);
        });
        for (String i : objectSet) {
            for (String j : objectSet) {
                if (!i.equals(j)) {
                    Set<String> setOfPredicatesI = ObjectInfo.get(i).keySet();
                    Set<String> setOfPredicatesJ = ObjectInfo.get(j).keySet();
                    for (String i1 : setOfPredicatesI) {
                        for (String j1 : setOfPredicatesJ) {
                            Collection<String> subjectI = ObjectInfo.get(i).get(i1);
                            Collection<String> subjectJ = ObjectInfo.get(j).get(j1);
                            double count = intersectionCount(subjectI, subjectJ);
                            if (count >= minSupport) {
                                yesCount = yesCount + 1;
                                Item item1 = new Item(i, i1);
                                Item item2 = new Item(j, j1);
                                if (largeItemSet.size() <= 28) {
                                    largeItemSet.put(item1, item2);
                                }
                            }
                        }
                    }
                }
            }
        }
        Set<Item> keys = largeItemSet.keySet();
        System.out.println(keys.size());
        return largeItemSet;
    }

    public double intersectionCount(Collection<String> subjectI, Collection<String>
subjectJ) {
        double count = 0;
        for (String subject1 : subjectI) {
            for (String subject2 : subjectJ) {
                if (subject1.equals(subject2)) {
                    count = count + 1;
                }
            }
        }
        return count;
    }

}



//Candidate
package org.crce.interns.service.impl;

import org.crce.interns.model.Item;
import com.google.common.collect.Multimap;
import com.google.common.collect.Sets;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.Map;
import java.util.Set;

public class GenerateCandidateSets {

    public LinkedHashSet<Set<Item>> generateAllPossibilities(Multimap<Item, Item>
largeItemset, Map<String, Multimap> ObjectInfo, double minSupport) {
        LinkedHashSet<Item> setOfItems = new LinkedHashSet(1000);
        Set<Item> objectKeys = largeItemset.keySet();
        setOfItems.addAll(objectKeys);
        Set<Set<Item>> power = Sets.powerSet(setOfItems);
        LinkedHashSet<Set<Item>> powerLinkedSet = new
    LinkedHashSet(Collections.synchronizedSet((power)));
        System.gc();
        ArrayList<Set<Item>> removalItems = new ArrayList(100);
        System.out.println(powerLinkedSet.size());
        for (Iterator<Set<Item>> i = powerLinkedSet.iterator(); i.hasNext();) {
            Set<Item> element = i.next();
            if (element.size() <= 2) {
                removalItems.add(element);
            }
        }
        powerLinkedSet.removeAll(removalItems);
        removalItems.clear();
        for (Iterator<Set<Item>> i = powerLinkedSet.iterator(); i.hasNext();) {
            Set<Item> element = i.next();
            if (getSupport(element, ObjectInfo) < minSupport) {
                removalItems.add(element);
            }
        }
        powerLinkedSet.removeAll(removalItems);
        System.out.println(powerLinkedSet.size());
         return powerLinkedSet;
    }

    public double getSupport(Set<Item> listOfItems, Map<String, Multimap> ObjectInfo) {
        double support = 0;
        for (Item element : listOfItems) {
          support = support + ObjectInfo.get(element.object).get(element.predicate).size();
        }
        return support;
    }

}
```

```
//Generate Rules
package org.crce.interns.service.impl;

import org.crce.interns.model.Item;
import com.google.common.collect.Multimap;
import com.google.common.collect.Sets;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ThreadLocalRandom;

public class GenerateRules {

    public ArrayList<Rule> generateAssociationRules(LinkedHashSet<Set<Item>>
possibilitySet, double minimumConfidence, Map<String, Multimap> ObjectInfo,String
fileName) throws IOException {
        LinkedHashSet<Item> set = new LinkedHashSet(100);
        String file = /*"C:/Users/leons/Desktop/" + */fileName + "1.csv";
        LinkedHashSet<Object> consequent = new LinkedHashSet(100);
        ArrayList<Rule> listOfRules = new ArrayList(100);
        for (Set ele : possibilitySet) {
            set.addAll(ele);
            Set<Set<Item>> power = Sets.powerSet(set);
            for (Set ele1 : power) {
                LinkedHashSet<Item> antecedent = new LinkedHashSet(ele1);
                for (Object i : ele1) {
                    antecedent.remove(i);
                    if (antecedent.size() > 2) {
                        if (getConfidence(antecedent, (Item) i, ObjectInfo) >= minimumConfidence) {
                            double conf = getConfidence(antecedent, (Item) i, ObjectInfo);
                            System.out.println(antecedent + "--->" + i + conf);
                            writeToFile(antecedent, (Item) i, conf, file);
                        }
                    }
                    antecedent.add((Item) i);

                }
            }
        }

        return listOfRules;
    }

    public double getConfidence(LinkedHashSet<Item> antecedent, Item element,
Map<String, Multimap> ObjectInfo) {
        double supportAntecedent = 0, supportConsequent = 0;
        for (Item i : antecedent) {
            System.out.println(ObjectInfo.get(i.object).get(i.predicate).size());
            supportAntecedent = supportAntecedent +
ObjectInfo.get(i.object).get(i.predicate).size();
        }
        supportConsequent = supportConsequent +
ObjectInfo.get(element.object).get(element.predicate).size();
        return (supportConsequent / supportAntecedent);
    }

    public void writeToFile(LinkedHashSet<Item> antecedent, Item i, double confidence,
String filename) throws FileNotFoundException, IOException {
        FileWriter output = null;
        try {
            output = new FileWriter(filename, true);
            try (BufferedWriter writer = new BufferedWriter(output)) {
                System.out.println("writing");
                writer.append(antecedent.toString().replaceAll(",", "^") + "," + i.toString() + "," +
Double.toString(confidence) + "\n");
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        } finally {
            if (output != null) {
                try {
                    output.flush();
                    output.close();
                } catch (IOException e) {

                }
            }
        }
    }
}
```

Chapter 8

References

1. R. B. V.Nebot, "Finding association rules in semantic web data," Knowledge-Based Systems, pp. 51-62, 2012.

2. M. A. Khan, Gunnar Aastrand Grimnes, and Andreas Dengel, "Two pre-processing operators for improved learning from semantic web data," presented at the In First RapidMiner Community Meeting And Conference (RCOMM), 2010

3. F. N. Ziawasch Abedjan, "Context and Target Configurations for Mining RDF Data," presented at the SMER '11 Proceedings of the 1st international workshop on Search and mining entity-relationship data 2011.

4. A. B. Christoph Kiefer, André Locher, "Adding data mining support to SPARQL via statistical relational learning," in ESWC'08 Proceedings of the 5th European semantic web conference on The semantic web: research and applications methods, 2008, pp. 478-492

5. R. Ramezani, "Finding Association Rules in Linked Data," M.Sc. Thesis, Computer & Electerical Engineering Department, Isfahan University of Technology, Isfahan, Iran, 2012, The proposed algorithm for ARM from semantic web data (SWApriori) will apear in a paper titled "A New approach to mining Association Rules from Semantic Web data". This paper is under publication.

6. R. I. V.Narasimha, O.P.Vyas, "LiDDM: A Data Mining System for Linked Data," presented at the Proceedings of the LDOW2011, Hyderabad, India, 2009.

7. L. Huang, Guoxiong Hu, and Xinghe Yang, "Review of ontology mapping," presented at the Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on. IEEE, 2012.

8. I.-Y. S. Namyoun Choi, Hyoil Han, "A Survey on Ontology Mapping," presented at the ACM SIGMOD Record, 2006.

9. M. S. Yannis Kalfoglou, "Ontology mapping: the state of the art," The Knowledge Engineering Review, vol. 18, pp. 1 - 31, 2003.