Distributed Multi-Relational Association Rule Mining for Ontological Data using Evolutionary Algorithms

Saikat Roy[1] and Vijayesh Kumar Das[1]

Institute for Informatic, University of Bonn, Germany
saikatroy@uni-bonn.de, s6vidass@uni-bonn.de

Abstract

This report was created as part of the Distributed Big Data Analytics Lab in the Winter Semester, 2018-19 at the University of Bonn. The aim of the project was to explore Distributed Multi-Association Rule Mining on Ontological Data using Spark and Scala. This report discusses methods explored and the results of the project.

1 Introduction

This report was created as part of the Distributed Big Data Analytics Lab in the Winter Semester, 2018-19 at the University of Bonn. The aim of the project was to explore Distributed Multi-Association Rule Mining using Evolutionary Algorithms. We discover hidden knowledge patterns in the form of multi-relational association rules by utilizing the evidence coming from evolving assertional data. An ontology may be incomplete, noisy and sometimes inconsistent, due to which ontologies and assertions may be out of sync. Hence, we use evolutionary algorithms to compute multi relational association rules.

Evolutionary algorithms typically suffer from the a prohibitive time complexity on fitness function evaluations. With the massive size of ontological databases such as dbPedia or Freebase for example, the issue of speeding up an association rule discovery algorithm becomes paramount. To achieve this, the existing evolutionary rule mining algorithm used in this work was augmented with a distributed fitness function calculator with the aim of reducing the time taken to train the algorithm for association rule discovery in large ontological datasets.

2 Problem Statement

The problem is defined as association rule mining in ontological knowledge base (KB) using an evolutionary algorithm. Formally the problem may be defined as:
Given an ontological KB κ, a frequency threshold θ_f, a head coverage threshold θ_{hc} and a confidence improvement threshold θ_{ic}, find a set of association rules, w.r.t. $_f$.

Association rules are defined as Sematic Web Rule Language (SWRL) rules in this work, which may be defined as a logical implication between an antecedent and a consequent of the form $B_1 \wedge B_2 \wedge ... \wedge B_N \rightarrow H$ where B_i represents body predicates and H represents the head.

2.1 Challenges

1. **Language Bias:** The language bias refers to maintaining properties of *safety*, *transitive connectivity* and *non-redundancy* in rules. These must be preserved throughout the course of the algorithm. We implement a *3-list binding priority* mechanism to enforce language bias of safety and transitive connectivity in rules.

2. **Distributed Nature:** The reference algorithm presented in [1] is essentially a non-distributed evolutionary algorithm. The major challenge of this project was to implement distributability while retaining the core strengths of the algorithm. We implement *Spark* based RDD-transformations and broadcasting (shared variables) to distribute the rule construction in a scalable fashion.

3 Approach

An association rule is defined as a conjunction on elementary atoms in the KB. Atoms are of 2 types: Concept and Role atoms, of the type $C(?x)$ and $R(?x, ?y)$. These are, simplistically, antonyms of subjects/objects and predicate as in relational data format (RDF) terminology. Understandably variables in Concept and Role atoms are 1 and 2 respectively. The main algorithm is an evolutionary technique designed to work on SWRL rules [2] by enforcing a Language Bias and redefining Crossover and Mutation operators for association rules in ontological KBs.

The main function is composed of the following functions:

1. **CreateNewPattern:** New random patterns are generated by selecting an atom for the head from a frequent atom list and then adding them to the body until a rule length, defined at the start of the function, is reached.

2. **Recombination:** Target lengths of patterns defined, followed by adding atoms (from pool of input atoms) until the lengths are satisfied. Language bias is enforced by maintaining transitive dependency during recombination.

3. **Mutate:** Applied with small probability. If head coverage of a pattern is higher than threshold, it is specialized. Otherwise it is generalized (last atom removed)

4. **Specialize:** Essentially adds a frequent concept or role atom to the pattern based on a random number generated in the range [0,1). Language Bias is enforced by controlling the variable bindings of the newly created atoms, based on predefined criteria

During the course of this project, several objects and functions were implemented to aid with the implementation of a distributed evolutionary algorithm. The major components of the algorithm are as follows:

1. `Atom(..) object:` Basic container object representing an atom in SWRL rules. Contains the properties of atom name, type, and variable bindings.

2. `Rule(..) object:` Object representing a Rule in SWRL format. Contains a list of atoms and provides for a wide variety of functions to enforce *Language Bias*.during rule growth or trimming.

3. `Operations object:` Object representing the major operations (recombination, mutate and so on) of the algorithm defined previously. Also contains the set of frequent role and concept atoms. This object is broadcasted during the run of the main algorithm to all partitions and provides a number of wrappers to access the bias maintainence function of the `Rule(..)` objects

4 Implementation

The project was programmed in Scala while the distributed components were implemented in Apache Spark. SANSA OWL was used for parsing the Owl file (Functional Syntax) used in the project. We started with defining separate classes for Atom and Rule objects. Parsing a composite data structure was defined for SWRL rules compliant with the ontology using scala.collections and Atom data structures by manually parsing the RDD axioms generated by the SANSA OWL framework.

4.1 Language Bias Maintenance

Language Bias was one of the main concern of mining rules from our algorithm. An example of a mined rule with Language Bias maintained is given below.

```
1 http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product(1) <-
2 http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product_Product_Line(1, 2)and
3 http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product(2) and
4 http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product_Manufacture_Location(2, 3)
5 and http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product(3) and
6 http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product_Manufacture_Location(2, 4)
7 and http://www.workingontologist.org/Examples/Chapter3/Product.owl#Product_Product_Line(4, 5)
```

Primarily, the implementation went according to the algorithmic descriptions in [1]. However, a major challenge was the enforcement of a Language Bias for the atoms in the SWRL rules. This was done by defining 3 ListBuffer objects in each Rule object populated by a simple integer values for the following:

1. Available: This was initially populated by elements in the range of 0 20. These represented bindings for new Atom objects with non-bound variables to be added into the rule. Values are removed every time when new objects use one of them when added into the Rule object.

2. Non-Available: This was defined to represent bindings that are non-available for free variables. This list is initialized empty and populated any time an Available variable is used.

3. Priority: These represent variables that are a priority to be used typically these are dangling Atom objects with unbound variables which risk the containing Rule object losing transitive dependence among all the atoms. The `Recombine(..)` and `Specialize(..)` functions used a combination of the above to maintain Language Bias in the implementation. Infact, each `Rule(..)` object contains its own set of Binding Variable Priority lists to help each of them independently maintain their language bias. Algorithm 1 gives a brief idea of the binding mechanism used in the project.

Data: Fresh Atom (a) with default bindings, Rule r
Result: Atom with rule language-bias preserving bindings
binding ← ***null***;
if *fresh binding variable not required* **then**
  read *priorityList*;
  if *priorityList is not* ***empty*** **then**
    binding ← random variable from *priorityList*;
    add variable to *usedVariableList*;
  else
    check *usedVariableList*;
    if *usedVariableList is not empty* **then**
      binding ← random variable from *usedVariableList*;
    end
  end
end
if *binding is* ***null*** **then**
  binding ← random variable from *freshVariableList*;
  add variable to *priorityList*;
end

Algorithm 1: Basic Mechanism for 3-List Binding Priority

4.2 Distributed Evolutionary Algorithm for Association Rule Mining

Our primary concern was transforming the non-distributed algorithm from [1] to a distributed implementation in Spark. We achieved the same by replacing modules of the main algorithm with distributed counterparts. Starting with a *RuleRDD* which is an RDD of `Rule` objects, they are as follows:

- **Fitness calculation and sorting:** The fitness calculation and sorting is rather straightforward given the broadcasted `Operations` object and is implemented below. It corresponds to lines 5-7 in the reference algorithm in [1].

```
\\ RuleRDD is an RDD of Rule Objects
\\ opRDDBroadcast is a broadcasted Operations(..) object
ruleRDD.map(r => opRDDbroadcast.value(0).calculateHeadCoverage(r))
ruleRDD.sortBy(_.hC)
```

- **Selecting pairs of Rule Objects for recombination and mutation:** Corresponding to lines 8-14 in its non-distributed counterpart, the following code selects pairs of rules and using the broadcasted `Operations` object, does the recombination and mutation operations on them while being distributed on the workers

```
var topRulesRDD = ruleRDD.zipWithIndex().filter(_._2<tp).map(_._1)
var bottomRulesRDD = ruleRDD.zipWithIndex().filter(_._2<N-tp).map(_._1)
var jointRulesRDD = bottomRulesRDD.zipWithIndex().filter(t=>(t._2+1)%2!=0).map(_.swap)
    .join(bottomRulesRDD.zipWithIndex().filter(t=>(t._2+1)%2==0).mapValues(f=>f-1)
    .map(_.swap)).map(_._2)
```

```
6
7   var processedRulesRDD = jointRulesRDD.map(x => {
8       val rand = new Random()
9       var r1 = x._1
10      var r2 = x._2
11      if(rand.nextFloat()<op.pcross){
12        var k = opRDDbroadcast.value(0).recombine(x._1, x._2)
13        r1 = k._1
14        r2 = k._2
15      }
16      if(rand.nextFloat()<op.pmut)
17        r1 = opRDDbroadcast.value(0).mutate(r1)
18      if(rand.nextFloat()<op.pmut)
19        r2 = opRDDbroadcast.value(0).mutate(r2)
20      ListBuffer(r1,r2)
21      }).flatMap(x => x)
22
23  processedRulesRDD.collect()
24  ruleRDD = topRulesRDD.union(processedRulesRDD)
```

4.3 Further Thoughts on Scalability:

One of the major restrictions of the project was that instead of using RDF Triples, we were restricted to using OWL Ontologies. A significant advantage of using RDF Triples is that our technique allows for the discovery of rules on a subset of the dataset. In combination with a RDF based Inference Engine possibly like that provided by Apache Jena, this algorithm would be able to discover rules in a scalable fashion across a large triple stores. This would be made possible by resampling the evaluation dataset periodically and recalculating the fitness function on the rules available. Although, this is purely a hypothesis on our part at this point.

5 Project Timeline

A brief description of the timeline of the project is given in Table 1.

6 Conclusion

In conclusion, we worked on the implementation of a distributed multi-association rule mining algorithm for ontological data. We worked on a Apache Spark based implementation of the algorithm and identified and tackled various algorithmic challenges during the course of this work. The issues from data-parsing to rule discovery in ontologies to exposure to the challenges of a Big Data implementation posed an engaging experience in this lab.

References

[1] Claudia dAmato, Andrea GB Tettamanzi, and Tran Duc Minh. Evolutionary discovery of multi-relational association rules from ontological knowledge bases. In *European Knowledge Acquisition Workshop*, pages 113–128. Springer, 2016.

Id	Dates	Project Work
1	8 November	- Project Assignment
2	9- 27 November	- Understanding the research paper - Preparation for Presentation
3	28 November	- Presentation
4	29 November to 31 December	- Code Inception - Owl Ontology using Sansa Owl - Implementation of Fitness Function - Implementation of Genetic Algorithms
5	1 January to 24 January	- Implementation of Apache Spark - Parallelising the whole Algorithm
6	25 January	Project Meeting
7	26 January - 21 February	- Code Integration - SWRL Rules - Results -Report
8	22 February	Project Submission

Table 1: Table describing the timeline of the project

[2] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21(79), 2004.