//

Semantic Analysis Using Map Reduce

Language: Java, Map Reduce
Author: Ankur Pandey, Nisha Choudhary

//

Table of Contents

1. MOTIVATION

Semantic analysis mainly focuses on finding a match between two people when a set of preferences in their various facets of life is known.

In the field of artificial intelligence, there are various ways to represent knowledge. Our project, semantic analysis attempts to be one of the various ways of knowledge representation.

Semantic Analysis uses input as an Entity with a set of associated tags. This aims basically to create a graphically structure where the nodes are representation of some concepts or entity. The entities are connected with each other with the help of arcs. These arcs are links that represent concepts or the relationship between them.

This can be used to analyze user's category likeness on a larger scale like on Wikipedia, Facebook, Twitter, LinkedIn, and many others. To analyzes different user likeness.

2. RELATED WORKS AND DIFFERENCES

The work based on the paper of 'Element level semantic matching using WordNet', is the one that is related to our project. Here the aim is to find semantic correspondences between elements of two graphs which search from parent to child to find match between two categories.

However, we proposed an algorithm that find a match or similarity between two categories by storing parent pointer to every child node. By that common point can be reached by following the parent pointer from two child node till the matched has been found or root has been found. Also, we implement this idea using the Map Reduce algorithm by parsing Xml Data. We start this from the two Objects which are the child nodes and then we find the shortest possible path between the two nodes i.e. lowest common ancestor node from that tree which will be faster to analyses score is data link is big rather than searching from parent.

3. PROBLEM DEFINATION

Our project focusses mainly on getting three things together:

SUBJECT (S) + PREDICATE (P) + OBJECT (O)

Subject and object can be of various examples like people, things, etc. Predicate here basically implies any verb that can bring similarity between the Subject and the Object.

Once this is identified, we then calculate the total score of the match between the two entities. The score would be directly dependent on the matches. Greater the number of matches, greater would be the score.

Example:

a. Mark likes San Francisco in California, in USA.
   Subject= Mark
   Predicate= Likes
   Object= USA

b. Juliette likes Seattle in Washington, in USA.
   Subject= Juliette
   Predicate= Likes
   Object= USA

The role of Semantic Analysis here is to bring together all the similarities and note the levels of differences by giving score between a subject and an object using in memory tree creation for the Data model.

4. PROPOSED METHODOLOGY

Our project, Semantic Analysis uses the Map Reduce to parse Xml files and get Subject and Object in our case UserID and link liked. This will create in memory tree. And search semantic score by below algorithm. We start this from the two Objects which are the child nodes and then we find the shortest possible path between the two nodes i.e. lowest common ancestor node from that tree which will be faster to analyses score is data link is big rather than searching from parent. The shorter the path the better would be the match.

Role of Mapper and Reducer:

Mapper: Input: <Key::UserID , Values::Link>

1. The mapper basically deals with parsing the XML file and getting User ID and link Searched.
2. Once the XML file is parsed and based on various categories a tree is created.
3. Into the mapper function which will be used by reducer to generate score, we have defined different levels, which deals with score allocation.

Reducer: Input:

1. Once all the levels are allocated, the reducer's job is to generate the final scores.

2. This will generate output === UserID1::USERID2::Score

   Eg. 16010000: 10025000: 100

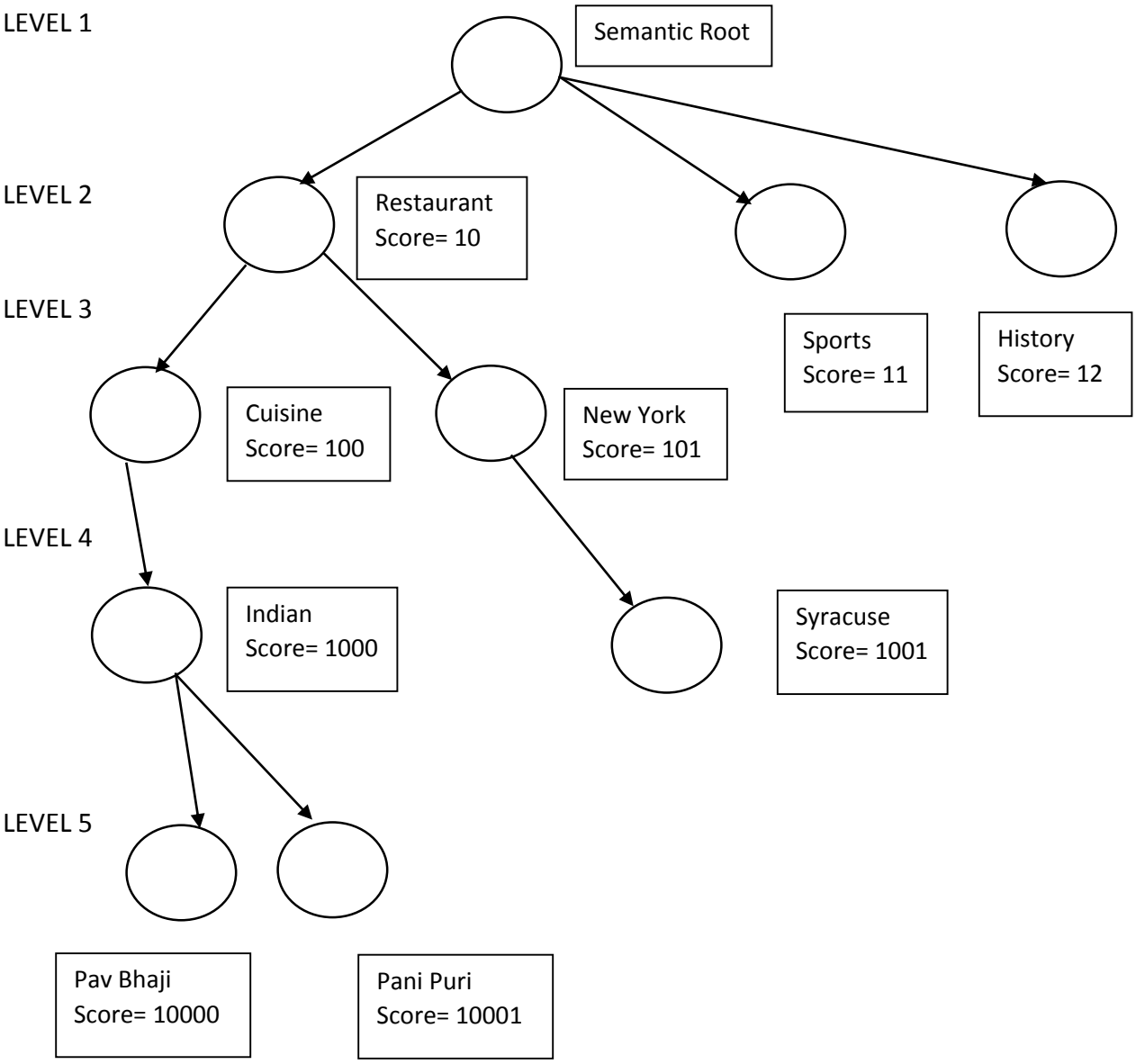
Semantic Score Calculation Algorithm

//

```
//----<     Search For Semantic Score   >-------------------------------------------------
public Integer semanticScoreSearch(String nodeSearch1, String nodeSearch2){
    Iterator<Node> depthIterator = tree.iterator("SemanticRoot");

    // Found node for specific Category 1
    Node nodeSearchforID1 = null;
    while (depthIterator.hasNext()) {
        nodeSearchforID1 = depthIterator.next();
        if (nodeSearchforID1.getIdentifier().contentEquals(nodeSearch1)){
            break;
        }
        System.out.println(nodeSearchforID1.getParent());
    }

    // Found node for specific Category 2
    Node nodeSearchforID2 = null;
    while (depthIterator.hasNext()) {
        nodeSearchforID2 = depthIterator.next();
        if (nodeSearchforID2.getIdentifier().contentEquals(nodeSearch2)){
            break;
        }
        System.out.println(nodeSearchforID2.getParent());
    }

    // Calculating semantic score
    return scoreCalculate(nodeSearchforID1, nodeSearchforID2);
}

//----<     Calculate Semantic Score   >-------------------------------------------------
public Integer scoreCalculate(Node nodeSearchforID1, Node nodeSearchforID2){

    // Loop till match not found or till root node in oour case SemanticRoot not found
    while ((nodeSearchforID1.getIdentifier() != "SemanticRoot" || nodeSearchforID2.getIdentifier() != "SemanticRoot"
        && (nodeSearchforID1.getIdentifier() != nodeSearchforID2.getIdentifier())){
        nodeSearchforID1 = nodeSearchforID1.getParent();
        nodeSearchforID2 = nodeSearchforID2.getParent();
    }
    if (nodeSearchforID1.getIdentifier() == nodeSearchforID2.getIdentifier()){
        return nodeSearchforID1.getLevel(); // Returning Score if matches node or category found
    }
    return 1; // Returning Score 1 if not found any matching node or category found
}
```

In Memory Graph Structure

//



If the graph link is huge then searching from bottom and passing to reducer will be faster.

As we observe from the graph above, the link of
Semantic Root/ Restaurant/ Cuisine/ Indian/Pav Bhaji
Semantic Root/ Restaurant/ Cuisine/ Indian/ Pani Puri
Here for two user score is: 1000

```
////////////////////////////////////////////////////////////////////////////
/////////// Reducer
////////////////////////////////////////////////////////////////////////////
public static class Reduce extends Reducer<Text, Text, Text, Text> {

    @Override
    protected void setup(Context context) throws IOException,
            InterruptedException {
    //  context.write(new Text("<configuration>"), null);
    }


    @Override
    protected void cleanup(Context context) throws IOException,
            InterruptedException {
    //  context.write(new Text("</configuration>"), null);
    }

    private Text outputKey = new Text();

    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        for (Text value : values) {
            outputKey.set(scoreCalcualte(key, value));
            context.write(outputKey, null);
        }
    }
////////////////////////////////////////////////////////////////////////////
/////////  Mapper
////////////////////////////////////////////////////////////////////////////
while (reader.hasNext()) {
    int code = reader.next();
    switch (code) {
    case XMLStreamConstants.START_ELEMENT: // START_ELEMENT:
        currentElement = reader.getLocalName();
        break;
    case XMLStreamConstants.CHARACTERS: // CHARACTERS:
        if (currentElement.equalsIgnoreCase("UserID")) {
            propertyID += reader.getText();
            System.out.println("propertName" + propertyID);
        } else if (currentElement.equalsIgnoreCase("link")) {
            propertylink += reader.getText();

            String TREETOKEN= "/";
            StringTokenizer TreeToken = new StringTokenizer(propertylink, TREETOKEN);
            String parent="";
            if(TreeToken.hasMoreTokens())
                parent = TreeToken.nextToken();
            if(parent!=""){
                tree.addNode(parent,"SemanticRoot");
            while (TreeToken.hasMoreTokens()) {
                String child = TreeToken.nextToken();
               // System.out.println(" child= " + child);
                tree.addNode(child,parent);
                parent = child;
               // System.out.println(" parent= " + parent);
                propertylink = child;
             }
            }
            System.out.println("propertyValue" + propertylink);
        }
        break;
    }
}
reader.close();
context.write(new Text(propertyID.trim()), new Text(
        propertylink.trim()));
```

5. SIMULATION/ EXPERIMENTAL RESULTS

INPUT: XML File

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <property>
        <UserID>10120000</UserID>
        <UserName>Ramesh</UserName>
        <link>Restaurant/Cuisine/Thai/USA/FastFood Huai</link>
  </property>
  <property>
        <UserID>10020000</UserID>
        <UserName>John</UserName>
        <link>Sports/Game/Football/Team/Chelsea</link>
  </property>
  <property>
        <UserID>10320000</UserID>
        <UserName>Ankur</UserName>
        <link>Restaurant/Cuisine/Thai/USA/FastFood Huai</link>
  </property>
  <property>
        <UserID>10025000</UserID>
        <UserName>Dave</UserName>
        <link>Sports/Game/Football/Team/Liverpool</link>
  </property>
  <property>
        <UserID>16010000</UserID>
        <UserName>Jack</UserName>
        <link>Sports/Game/Criket/Team/India</link>
  </property>
  <property>
        <UserID>10120000</UserID>
        <UserName>Nisha</UserName>
        <link>Restaurant/Cuisine/Indian/Rice</link>
  </property>
</Configuration>
```

OUTPUT:

Format= ID1 : ID2 : Score

10120000 : 10120000 : 100000
10120000 : 10020000 : 0
10120000 : 10320000 : 100000
10120000 : 10025000 : 0
10120000 : 16010000 : 0
10120000 : 11120000 : 100

10020000 : 10120000 : 0
10020000 : 10020000 : 100000
10020000 : 10320000 : 0
10020000 : 10025000 : 100000
10020000 : 16010000 : 100
10020000 : 11120000 : 0
10320000: 10120000 : 100000
10320000: 10020000 : 0
10320000: 10320000 : 100000
10320000: 10025000 : 0
10320000: 16010000 : 0
10320000: 11120000 : 100
10025000: 10120000 : 0
10025000: 10020000 : 100000
10025000: 10320000 : 0
10025000: 10025000 : 100000
10025000: 16010000 : 100
10025000: 11120000 : 0
16010000: 10120000 : 0
16010000: 10020000 : 100
16010000: 10320000 : 0
16010000: 10025000 : 100
16010000: 16010000 : 100000
16010000: 11120000 : 0
11120000: 10120000 : 100
11120000: 10020000 : 0
11120000: 10320000 : 100
11120000: 10025000 : 0
11120000: 16010000 : 0
11120000: 11120000 : 100000

6. ANALYSIS OF THE RESULTS

Above output shows higher the scores higher the similarity between two subjects.

From this, we can find the similarity of different subjects like in our case User ID and Objects of different category. We can conclude the following things from this.

1. The output shows value according to deeper level of matches of knowledge between two users.
2. From this data we can recommend category one level below the match level from proposed common Ancestor node search from child node.
3. This can be used to analyze user's category likeness on a larger scale like on Wikipedia, Facebook, Twitter, LinkedIn, and many others. To analyzes different user likeness.

7. INDIVIDUAL CONTRIBUTIONS

1. Ankur Pandey
   a. Project Idea
   b. Analysis of the Project Scope
   c. Analysis of the Methodology of the Project
   d. Implementation of the Project

2. Nisha Choudhary
   a. Designing of the Project
   b. Implementation of the Project
   c. Testing of the Project

8. REFERENCS

1. "Element level semantic matching using WordNet" by Mikalai Yatskevich and Fausto Giunchiglia.
2. http://www.w3schools.com/xml/xml_whatis.asp
3. http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html