Konstantinos Sgontzos

On Achieving High Quality User Reviews Retrieval in the context of Conversational Faceted Search

Software Documentation

Here we describe how to install and use the software through command line, in programmatic environment, as well as a web Service. The examples used are implemented in Java and JavaScript for the latter case.

Requirements:

JDK
Maven
Wordnet
Word2vec Pre-trained Model
Minimum RAM requirement: ***11 GB***

How to compile the jar:

The jar will retrieve the rdf data from the src/main/resources/warehouse/ folder's files and the model will use the properties directly from the src/main/resources/configuration/onFocusRRRConfig.properties file. To change these values replacement of the warehouse and of the onFocusRRRConfig.properties content should be done and rebuild the project. The project can be found in its ***github repository.***

How to run the jar:

Command: java –jar QuestionAnswering-1.2-SNAPSHOT.jar –Xmx10g
Note: *The jar should be in the same directory with the lib folder provided in the "target" folder of the Maven Question Answering Project.*
Note: *The class OnFocousRRR.java should be set as main class from pom.xml.*

How to use the model in a Maven project:

To use the model in maven environment you should first install the jar file in your local maven repository. For example in my environment I use the following cmd command:

```
mvn install:install-file
  -Dfile=C:\Users\Sgo\Documents\NetBeansProjects\QuestionAnswering\target\QuestionAnswering-1.2-SNAPSHOT.jar
  -DgroupId=gr.forth.ics.isl
  -DartifactId=QuestionAnswering
  -Dversion=1.2-SNAPSHOT
  -Dpackaging=jar
  -DgeneratePom=true
```

Now you are able to add the appropriate dependency in your own maven project. Specifically you need two dependencies:
1) The project dependency that you have just installed in your local maven repository, i.e.:

```
    <dependency>
        <groupId>gr.forth.ics.isl</groupId>
        <artifactId>QuestionAnswering</artifactId>
        <version>1.2-SNAPSHOT</version>
        <type>jar</type>
    </dependency>
```

2) The json dependency which the output of the project supports, i.e.:

```
    <!-- https://mvnrepository.com/artifact/org.codehaus.jettison/jettison -->
    <dependency>
        <groupId>org.codehaus.jettison</groupId>
        <artifactId>jettison</artifactId>
        <version>1.4.0</version>
    </dependency>
```

Now you can simply initialize the OnFocusRRR model and use it in your code. A Java example follows:

```
package gr.uoc.csd.thesis_demo.onfocusrrr_usage_example;

import gr.forth.ics.isl.demo.main.OnFocusRRR;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Properties;
import org.codehaus.jettison.json.JSONArray;
import org.codehaus.jettison.json.JSONException;
import org.codehaus.jettison.json.JSONObject;

/**
 *
 * @author Sgo
 */
public class Example {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException, JSONException {

        Properties properties = new Properties();

        properties.setProperty("gModelPath", "C:/Users/Sgo/Desktop/Developer/Vector Models/GoogleNews-vectors-
negative300.bin.gz");  //path and file of word2vec model
        properties.setProperty("wnhomePath", "WNHOME");  //path of environment variable Wordnet home
        properties.setProperty("cwList", "problem,issue,report,hotel,complaint,anyone,complain");  //list of context words
        properties.setProperty("word2vec_w", "0.4"); //word2vec weight
        properties.setProperty("wordNet_w", "0.6"); //Wordnet weight
        properties.setProperty("sqe", "true"); //true to use SQE, false for not using it
        properties.setProperty("wordnet_resources", "synonyms,antonyms,hypernyms"); //resources to use to expand the query

        OnFocusRRR model = new OnFocusRRR(properties); //call the constructor of the model

        //input uris
        ArrayList<String> uris = new ArrayList<>();
        uris.add("http://ics.forth.gr/isl/hippalus/#hotel_monte_hermana_kobe_amalie");
        uris.add("http://ics.forth.gr/isl/hippalus/#hotel_monterey_grasmere_osaka");
        uris.add("http://ics.forth.gr/isl/hippalus/#hotel_monterey_hanzomon");
        uris.add("http://ics.forth.gr/isl/hippalus/#hotel_monterey_osaka");

        //input question
        String question = "Is this hotel quiet?";

        //call method to retrieve top 2 relevant reviews
        JSONObject resultListAsJASON = model.getTop2Comments(uris, question);
        //use the bellow method instead, in case you like to choose the size of the result list
        //JSONObject resultListAsJASON = model.getTopKComments(uris, question,10);

        //get specific field of the result, e.g. masxed scored sentence of the two most relevant retrieved reviews.
        JSONArray maxSentences = resultListAsJASON.getJSONArray("maxSentences");
        //print them to console
        for (int maxSentId = 0; maxSentId < maxSentences.length(); maxSentId++) {
            System.out.println(maxSentences.get(maxSentId) + "\n");
        }
    }

}
```

For accessing all fields here is a quick guide:
maxSentences*: a JSONArray that contains the maxed scored sentence of each retrieved review sorted with respect to the result list. (used in the example above)*
commentIds*: a JSONArray that contains a unique identifier of each retrieved review sorted with respect to the result list.*
dates*: a JSONArray that contains the publication date of each retrieved review sorted with respect to the result list.*
fullReview*: a JSONArray that contains the full text of each retrieved review sorted with respect to the result list.*
posParts*: a JSONArray that contains the positive part of each retrieved review sorted with respect to the result list.*
negParts*: a JSONArray that contains the negative part of each retrieved review sorted with respect to the result list.*
scores*: a JSONArray that contains the score of each retrieved review based on its maxed scored sentence and sorted with respect to the result list.*
hotelIds*: a JSONArray that contains the hotel id of each retrieved review sorted with respect to the result list.*
hotelNames*: a JSONArray that contains the hotel name of each retrieved review sorted with respect to the result list.*

How to use the model as a web service:

To use the project as a web service you should send a post request to the service, which is currently hosted in = http://139.91.183.46:8080/QuestionAnswering/service/find/. Input to the post request are the hotel uris with key-uris and value-the uris separated by comma. Second, the query with key-query and value-text of the question. The output would be a JSONObject and its fields can be accessed as shown in the bellow example. Please use the method getValidUrisPost() provided bellow for valid uri representation.

```
function sendQuery() {

    var url = "http://139.91.183.46:8080/QuestionAnswering/service/find/";
    var uris = document.getElementById("uris").value;
    var query = document.getElementById("query").value;

    var urisValid = getValidUrisPost(uris);

    var data = {};
    data['query'] = query;
    data['target_selection'] = urisValid;
    alert(JSON.stringify(data));
    ajax("POST", url, data);

}



function ajax(method, url, data) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {

            document.getElementById("output").innerHTML = "";

            var json = JSON.parse(xhttp.responseText);

            var results = json.results;

            for (var i=0; i<results.length; i++){
                var txt = "<table border='1'>";

                txt += "<tr><td>Hotel name</td><td>" + results[i].hotel_name + "</td></tr>";

                var reviews = results[i].reviews;

                for (var j=0; j<reviews.length; j++){

                    txt += "<tr><td>Review ID</td><td>" + reviews[j].review_comment_id + "</td></tr>";
                    txt += "<tr><td>Review Relevant Mention</td><td>" + reviews[j].max_sentence + "</td></tr>";
                    txt += "<tr><td>Score</td><td>" + reviews[j].score + "</td></tr>";
                    txt += "<tr><td>Review Possitve Part</td><td>" + reviews[j].positive_review_comment + "</td></tr>";
                    txt += "<tr><td>Review Negative Part</td><td>" + reviews[j].negative_review_comment + "</td></tr>";
                    txt += "<tr><td>Review Date</td><td>" + reviews[j].review_date + "</td></tr>";

                }

                txt += "</table>";

                document.getElementById("output").innerHTML += txt + "<br>";
            }


        }
    };
    xhttp.open(method, url, true);
    xhttp.setRequestHeader("Content-type", "application/json");
    xhttp.send(JSON.stringify(data));
}

function getValidUrisPost(uris) {

    var urisArray = uris.split(",");

    return urisArray;
}

function getValidUrisGet(uris) {
    var urisArray = uris.split("#");
    var urisClean = "";
    for (var i = 0; i < urisArray.length; i++) {
        urisClean += urisArray[i];
        if (i < urisArray.length - 1) {
            urisClean += "%23";
        }
    }

    return urisClean;
}

function getValidQueryGet(query) {

    var queryArray = query.split(" ");
    var queryClean = "";
    for (var i = 0; i < queryArray.length; i++) {
        queryClean += queryArray[i];
        if (i < queryArray.length - 1) {
            queryClean += "%20";
        }
    }

    return queryClean;
}
```