Building an Apache-Solr based Search Engine for DARPA XDATA Employment Data

Neha Ahuja
Poushali Banerjee
Renu Kankamedala
Aldrin Rodrigues

CSCI 572 HW 2
Prof. Chris Mattmann

November 10, 2014
University of Southern California

I. Introduction

The objective of this assignment is to build both an indexing system (inverted index), and a set of ranking algorithms for the XDATA employment dataset. In this project we create two ranking algorithms namely “content based” ranking algorithm, that uses the metadata information generated by OODT that is attached to a JSON file to develop a “score” or “rank” for each JSON document and “link based” ranking algorithm, that uses that relationship between the jobs and provide a method for ranking and citation. We have worked with a subset of the data for the most except where mentioned. The major technologies used in this project are Apache Solr and Apache OODT.

II. Methodology

The data used in this project is taken from the de duplicated data set generated from homework1. The data is a set of JSON files. Each file has one employment record. The parameters include companyName, jobtype, location, postedDate, lastSeenDate, title etc.

Task 1:

a) We installed Solr-4-10.1 and Apache OODT 0.7 using RADIX installation. Installing Solr was an easy task, however there were a number of steps required for installing OODT. Next installation was that of the ETLlib library. We used the OODT file manager and workflow manager to ingest and index our JSON documents into Solr using POSTER command from ETLlib package developed by Prof. Chris Mattmann. Used CAS PGE to create a workflow for ingesting files into Solr via ETLLib. For more information about installing and starting different components please refer ReadMe.txt.

b) Ingesting Metrics
We captured the metrics for Wall clock time and per job ingestion time using the Solr Admin web GUI. For faster ingestion, we distributed our task to four machines just for the sake for getting an averaged set of time. Thus, depending on the results from our for systems (2 with Ubuntu - 3GB RAM, 1 Mac book Pro and 1 Mac book Air). We got an average of **38** seconds as a single file ingestion time. For the full de duplicated data set the average time or the Wall Clock time read as **12616** (this is like 9 days) minutes (Sequential workflow). We divided the data across systems and ingested in parallel so it took around 2 days for all the data chunks.

Task 2:

a) **Content based ranking Algorithm:**

- We considered specific attributes in the JSON file to boost the relevant document score and added meta data about that attributes that helped us find other similar documents to show higher than actual.
- Input for the algorithm is a JSON file.
- Our metadata attributes were job type score, location score, a tag which says whether a job is posted in South America or not, number of days in which the job closed out and a title tag which captures specific tokens from title of the job and classifies what type of job it is.
- This meta data helped us in performing specific challenge queries given in task 3 and reshuffle the Solr's original ordering of results to that with our content based algorithm.
- Date boosting feature of Solr is also integrated into our algorithm along with other metad which helps order the results based on Posted Date.
- Boosting of results can be done with &bf function which is part of functionquery. A parameter can be supplied to this field based on which the resuts will be improved. Typically a score should be should be calculated and fed into the argument.

Justification: After trying different examples with and without using ranking we identified that the out metadata helped us in ranking the documents properly. Example: if we are looking for location with more part time jobs we should show the location which has more locScore and more jobTypeScore - part time. LocScore and JobType Score descriptions are mentioned in linkbased.

b) **Link based Ranking Algorithm:**

Our link based algorithm takes into consideration the virtual links or relationships that exist between different JSON documents. Our approach was to weigh location, company and job type and generate a total link score that would boost jobs that have links (relationship) and thus show them up higher in the results.

Algorithm:

1. The fields for which realtionship score should be caluclated are jobtype, location, company.
2. For each fle in the subset we will calculate a location score based on the location name in the JSON. For example, if the location is Los Angeles we will check how many other Json documents have job posting in Los Angeles. From this we can infer that the Json document is linked to all the other documents if they same location
3. The location score is considered based on the lat, long pair

4. Similarly we will calculate the score for company and jobType too
5. Generate the link score for every document using the formula below
   link score = a*company score + b*job type score + c*location score
   Incorporated this into solr for ranking

Integeration with Solr:
We can incorporate out algorithm in to Solr by creating a jar of our program and add it to the lib folder of solr. Also we can create a custom scoring by extending the Custom Score Generator class in lucene. We followed the first step and added the required fields to our metadata and used in ranking the documents.
Justification: After trying different examples with and without using ranking we identified that the out metadata helped us in ranking the documents properly. Example: if we are looking for location with more part time jobs we should show the job which has more link score on the top because if the link score is high then obviously it has more connections and users may want to see that.

Task 3:

Query 1: Predict which geospatial areas will have which job types in the future.

http://localhost:8983/solr/select?wt=json&indent=true&q=*:*&&fl=location2,jobtype&facet=true&facet.pivot=location2,jobtype

In this query, we have used 'facet' function query of the Apache Solr. In the facet.pivot = location2 , jobtype , we are basically grouping the results based on the geographical location and the job type like full time and half time. The pivot returns the result based on highest to lowest value.

So, the results returned by this query, groups the data by location and for each location, we have different job type and we have a count that is returned for each job type. This count is displayed from highest to lowest. The higher number value for the job type is the result to our query. As, according to us, the job type with the high value for a specific location is would be repeated in the future.

Our results demonstrate that for location Buenos Aires, Argentina, there are 3240 jobs of type Tiempo Completo and 145 of type medio tiempo. Results are sorted by descending order and therefore for this particular location, it can be predicted that in future as well this will be the highest job type in this area. The margin can also be taken into account. If two results come up with a narrow margin then, other factors would have to be taken into account before predicting. In this particular example, the margin is huge between 3240 and 145. Also, just like we have done the grouping based on job type, the grouping can also be done based on title to figure out the categories of jobs in a particular area. The results in this query can be improved by including this factor in our query and or boosting the query with the jobscore that we have calculated.

→ Refer to Query1.txt file for sample output run on one of our 4 machines.

Query 2: Compare jobs in terms of quickly they're filled specifically in regards to region.

i) http://localhost:8983/solr/select?q=*:*&stats=true&facet=true&facet.pivot=location2,jobtype,noOfDays&facet.field=jobtype&wt=json&indent=true

In this query we have displayed what is the number of days it takes for a jobtype in a particular location to be filled up. So for, example, in Buenos Aires, each jobtype shows what is the number of days it took to fill up.

Note, it would be best to include statistics as part of the pivot facets which would make this query robust and this has been implemented via patch 3583 on solr, so going forward it will be possible to make the stats fields like min, max and avg directly as part of the pivots which would give the average number of days it took to fill a job at a particular location.

ii) http://localhost:8983/solr/select?q=*:*&facet=true&stats=true&stats.field=noOfDays&stats.facet=location2&facet.field=location2&facet.pivot=location2&wt=json&indent=true

In this query, we used the metadata "noOfDays" as a parameter to query on the results to get the duration when it was posted and lastSeenDate. The stats.field give the mean (average) number of days taken to fill a job in a particular location.

Our results demonstrate that for location San Antonio Oeste, Río Negro, Argentina", on an average it takes 35 days to fill up jobs and shows how many days it will take on average to fill up jobs at any location.

→ Refer to Query2.txt file for sample output run on one of our 4 machines.

iii) http://localhost:8983/solr/select?q=*:*&stats=true&stats.field={!tag=piv2}filled&facet=true&facet.pivot={!stats=piv2}location2,jobtype&wt=json&indent=true

This query is the valid and correct way of displaying the answer to this question. We define tags for the pivot so that it can be referenced from the facet field. This query works on solr 5.0, but this kind of tagging is not supported in solr 4.

Query 3: Can you classify and zone cities based on the jobs data (E.G. commercial shopping region, industrial, residential, business offices, medical, etc)?

http://localhost:8983/solr/select?wt=json&indent=true&q=*:*&fl=location2,title&facet=true&facet.pivot=title,location2&sort=jobTypeScore%20desc&sort=locScore%20desc

In this query, we are classifying the jobs based on the job titles and we have used the jobTypeScore and locScore metadata to return the results with a higher score. If we don't use the jobTypeScore or the locScore paramates, it would work as a normal Content based algorithm.

We tried this query, however, it does not consider specific job titles. Based on our research on this query, it might be possible to collaborate the query3 with clustering feature of the solr to get an exact output for the same.

→ Refer to Query3.txt file for sample output run on one of our 4 machines.

Query 4: What are the trends as it relates to full time vs part time employment in South America?

http://localhost:8983/solr/select?q=isSA:true&sort=noOfDays%20asc&facet=true&facet.pivot=jobtype,title&group=true&group.field=jobtype&fl=title,jobtype,id&wt=json&indent=true

In this query, we are using the metadata "isSA" (is South America") to get the results for the South America region. We have sorted those results by the job type (Part Time and Full Time) and displayed the details with the job title.

Our query looks whether the location is South America or not. It display the total number of jobs for each Part time, Full Time, Part and Full Time. Then, for each job type, it lists which job title has how many jobs.

In this query, we have considered the trend to be what kind of jobtype are present for what job title as our trend, for South America location.

→ Refer to Query4.txt file for sample output run on one of our 4 machines.

NOTE: If we add the linkScore to any of the queries and sort them by the asc or desc order, we will get the results for the link based algorithm.

Task 4:

This is an interactive program (QueryProgram.java) that displays the results for the four challenge questions

III. Results

1. Query1.txt
2. Query2.txt
3. Query3.txt
4. Query4.txt

Notes: Generated result for subset of files

REPORT ANALYSIS:

.

1. For example,how effective was the link based algorithm, compared to the content based ranking algorithm?
   Our content based algorithm focused on the challenge questions given in task 3. We first identified the attributes that were useful and more relevant for the challenge questions and then used these attributes for boosting the score in content based ranking. However, for the link based algorithm, we started off directly by constructing relationships between documents by scoring similar location, company and job type jobs and then boosting relevant documents. The link based approach is more general whereas the content based is specific t the four challenge questions.

2. What challenge questions were more appropriate for the link based algorithm compared to the content one?

   It is easier to answer question thats have queries about geolocations with functionquery, content based, because there are many inbuilt and well defined functions that make geospatial searching easy to execute. Especially like defining a bounding box, etc is very easy to do with functionquery map, etc.

3. What do you think were the strengths of OODT? What did you think were the weaknesses of OODT?

Strengths of OODT:

a) It allows to generate the metadata for the input files The server side metadata is generated by the OODT itself and the client side metadata can be generated by us.
b) The generated metadata helps to retrieve the results faster. Example, adding the metadata like noOfDays, jobScore, isSA (is South America),title tag, titleScore.

c) It has separate components like file manager, workflow manager, and, resource manager which allows to work to these components individually.
d) The resource stub is a part which helped us to find our errors while we were trying / workin on different parts of the assignment.
e) The pipeline feature of the OODT is very beneficial for the scenario where we have multiple tasks that are required to be run in a sequence, although it could not be completely used in this assignment.
f) For CAS-PGE, the commands were properly explained.

Weaknesses of OODT:

a) It is not consistent on all machines. For example, on downloading the OODT 0.7 after uninstalling OODT 0.6, we found that on different machines, the installation folder was different. One machine did not 0.7 cas-pge jars, others did not generate both 0.6 and 0.7, and on a one machine all the 0.3, 0.6, 0.7 folders were generated.
b) The 'stop' command does not work properly every time, Many a times, we had to manually browse to the 'run' folder of the 'file manager', 'workflow manager', and, 'resource manager' to delete the cas-pid files to restart (stop and start).

4. Describe in detail and formally both of your ranking algorithms. You should describe the input, what your algorithms do to compute a rank, how to test them (and prove that they are working as expected).

   Please refer to the ranking algorithms section for the answer to this question.

5. Describe the indexing process what does your OODT workflow do? How does it interact with ETLlib?

For ingesting the files on the Apache Solr, the OODT uses the file manager, workflow manager, resource manager and resource stub. For using the poster command of the ETLlib, we have used the command in the PGEConfig.xml file. Also, we changed the Schema.xml file of the solr / collection folder according to our deduplicated json files which we had to upload on Apache Solr.

6. Also include your thoughts about ETLlib what was easy about using it? What wasn't'?

a) The documentation on the use of the ETLLib commands is very useful as it gave a proper example of how to run the commands.
b) In our opinion, using ETLLib was a lot easier than the OODT and Apache Solr.

c) However, we found the ETLLib uses certain specific versions of the jcc, jdk, pthyon. Like it is not yet compatible with the jdk 1.8 version. So, we had to uninstall and install certain versions of it base requiremnets.

Submitting patches and contributing to open source:

Created an issue an in Jira for OODT and a pull request for the same. Described the solution in the ticket itself.

Reference : Jira Ticket OODT-784