

## HW4: Inverted-index creation, using Lunr and Solr

### Summary

In this 4th HW, you're going to use Lunr (a JS library) in three ways, and Solr (a Java library) in one way, to inverted-index documents/data (each piece of data, eg a student's info, is called a 'document' and is expressed as JSON).

---

### Description

We are doing to describe each of the 4 pieces separately below, since they are all independent of each other...

#### 1. Lunr, using xem

Start with <https://bytes.usc.edu/~saty/tools/xem/run.html?x=lunr-cs572-hw4>. As you can see, we fetch [https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp\\_academic\\_dataset\\_business.json/yelp\\_academic\\_dataset\\_business-50.json](https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp_academic_dataset_business.json/yelp_academic_dataset_business-50.json) [a portion of the famous Yelp businesses dataset, with only 50 rows], index the 'state' field, then display the 'name' field as the output (in Lunr terminology, this (name) is the 'ref' field).

Your turn: also fetch [https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp\\_academic\\_dataset\\_business.json/yelp\\_academic\\_dataset\\_business-1000.json](https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp_academic_dataset_business.json/yelp_academic_dataset_business-1000.json), and after that, [https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp\\_academic\\_dataset\\_business.json/yelp\\_academic\\_dataset\\_business.json](https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp_academic_dataset_business.json/yelp_academic_dataset_business.json), do the same state searches. **For "PA", how many results do we get, for the three different files?**

Next, pick a different column (field) to index [https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp\\_academic\\_dataset\\_business.json/yelp\\_academic\\_dataset\\_business-50.json](https://bytes.usc.edu/cs572/s23-sear-chhh/hw/HW4/data/yelp_academic_dataset_business.json/yelp_academic_dataset_business-50.json), then **search for a value in that field, grab a screenshot to submit.**

---

#### 2. Lunr, using repl.it

Bring up <https://replit.com/@satychary/TrivialGrouchyScreenscraper>, fork and run it. You'll see a file called fun.jsonl, containing two simple docs that get indexed and searched. **REPLACE the data with your own, make it have 10 rows/docs** - that also has two fields, eg. (movieName, rating), (course, grade), (foodItemUnitPortion, calories)... Alternately you can also add more schools and rankings if you like (eg from <https://www.usnews.com/best-colleges/rankings/national-universities>), rather than use your own alternate fields. **Programmatically search for a range of the second field (eg rank, grade, calories, ratings...), and display the result as a simple JS array - get a screenshot.** In other words, suppose you have food names and calories - you'd search for a range of calories, eg. 500 to 1000, via a for() or forEach() loop for each of 500,501,502...1000 [search 501 times]; if the search comes back non-null, you'd add the result an empty array you start with. When you're done, that array (eg. myResults) will contain the range results; print it, get a screenshot.

---

#### 3. Lunr, via TypeScript in StackBlitz

Fork <https://stackblitz.com/edit/lunr-getting-started-z5rtjm?file=index.ts>, and take a look at index.ts, to find a 'documents' array (these are what get indexed and searched). Put in 10 paragraphs from HW3's data2.zip (the 'smaller' files), to create 10 indexable documents. **Search the documents for two different terms, which you know, occur in more than one document. Grab a screenshot for each search.**

#### 4. Solr, using Docker

Solr is a powerful search engine with a rich search syntax, easy setup, fast indexing - we specify a search 'schema' (fieldname, fieldtype), for the fields we want to index and search, then we add documents (data) which kicks off the indexing, then we search (via a URL query, or via one of MANY APIs in multiple languages).

Start by installing Docker: <https://docs.docker.com/engine/install/> Docker makes it so easy to run Solr inside a lightweight virtual machine 'container' (runtime) - to do this, we'd first download an 'image' (template), then run it to launch a container.

Bring up a conda shell on a PC (<https://docs.conda.io/projects/conda/en/latest/user-guide/install/windows.html>) or a terminal on Mac/Linux, and type the following, to download the Solr image.

```
docker pull solr
```

Now we can run Solr via [see [https://hub.docker.com/\\_/solr](https://hub.docker.com/_/solr) for more]

```
docker run -d -p 8983:8983 --name my_solr solr solr-precreate my_core
```

In the above, 'my\_core' is our name for a Solr 'core', which is an area where Solr keeps the docs to index, the index itself, and other files (eg config).

FYI, you can run 'docker ps' to verify that the container is indeed, up and running:



```
Anaconda Prompt

(base) C:\Users\satyc>docker run -d -p 8983:8983 --name my_solr solr solr-precreate my_core
432c612ab803d7eccbf2966e619bb6216fb6fac5063bb599c7eb07d52c8a1bfa

(base) C:\Users\satyc>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
432c612ab803   solr      "docker-entrypoint.s..." 42 seconds ago Up 38 seconds 0.0.0.0:8983->8983/tcp   my_solr

(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>
```

We are almost ready to start using Solr, via the browser (by visiting <http://localhost:8983/>) -but- we need one more thing first - a webserver. Here is one way: in the conda shell, run

```
python serveit.py
```

Here is serveit.py (you'd create it by copying and pasting my code below):

```
import http.server

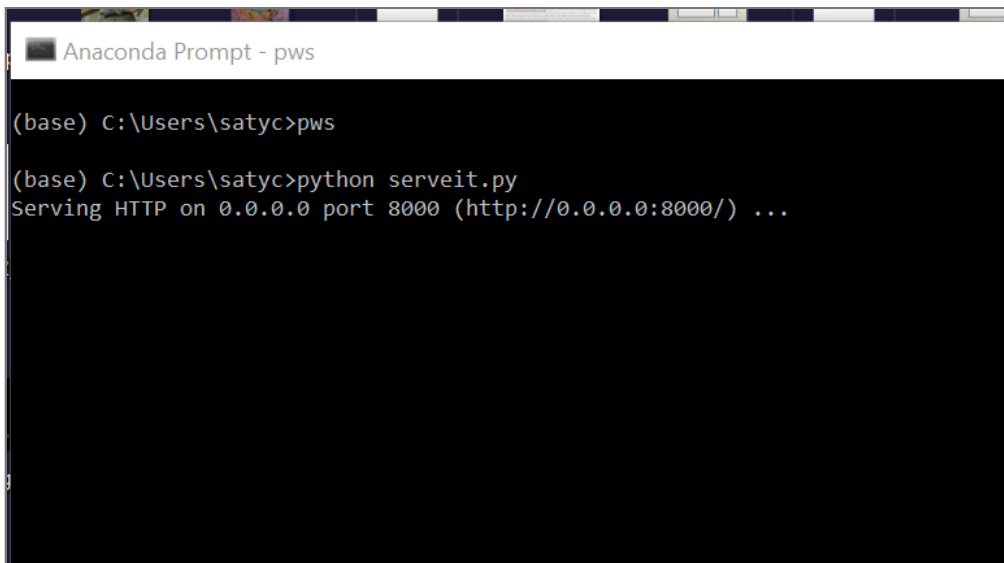
class MyHTTPRequestHandler(http.server.SimpleHTTPRequestHandler):
    def end_headers(self):
        self.send_my_headers()
        http.server.SimpleHTTPRequestHandler.end_headers(self)

    def send_my_headers(self):
        self.send_header("Cache-Control", "no-cache, no-store, must-revalidate")
        self.send_header("Pragma", "no-cache")
        self.send_header("Expires", "0")

if __name__ == '__main__':
    http.server.test(HandlerClass=MyHTTPRequestHandler)

# from https://stackoverflow.com/questions/12193803/invoke-python-simplehttpserver-from-command-line-with-no-cache-option
# usage: python serveit.py :) GREAT, because it does NOT cache files!! Serves out of port 8000 by default.
```

I've aliased the Python command above, to 'pws', since I run it a lot:



```
Anaconda Prompt - pws

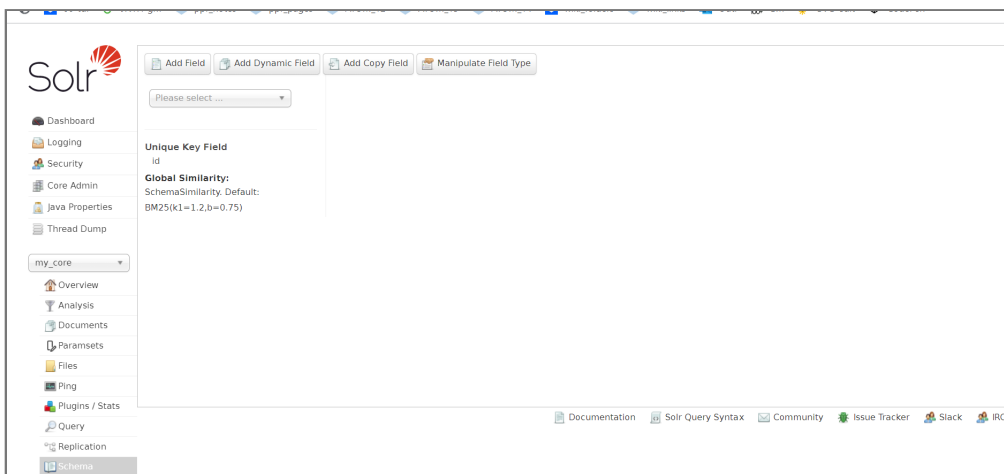
(base) C:\Users\satyc>pwd

(base) C:\Users\satyc>python serveit.py
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

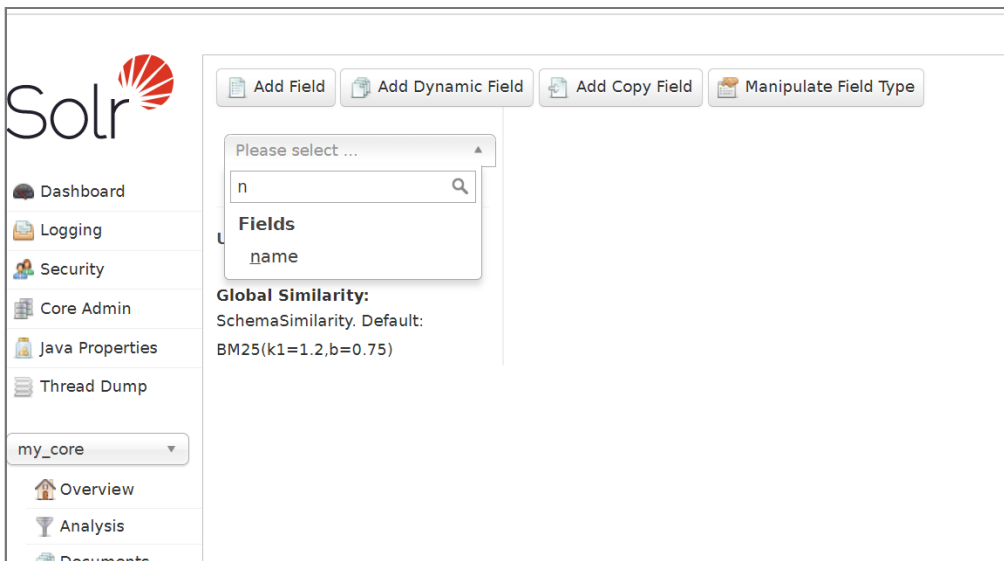
Ready to roll! To see the 'Solr panel' [lol!], go to <http://localhost:8983> - ta da!!

Poke around in the panel, by clicking on the various buttons, to get a feel for the interface - you can see that there is a LOT you can specify, customize, run - please do explore these after the course. But for this exercise, we only need to do three things:

1. Add field names and types, for the docs (data) we will index - in other words, we need to specify a schema (click on 'Add Field', to add fields one by one):



After creating our schema, we can verify that it looks right by searching for the fieldnames we put in:



Solr

- Dashboard
- Logging
- Security
- Core Admin
- Java Properties
- Thread Dump
- my\_core
- Overview
- Analysis
- Documents

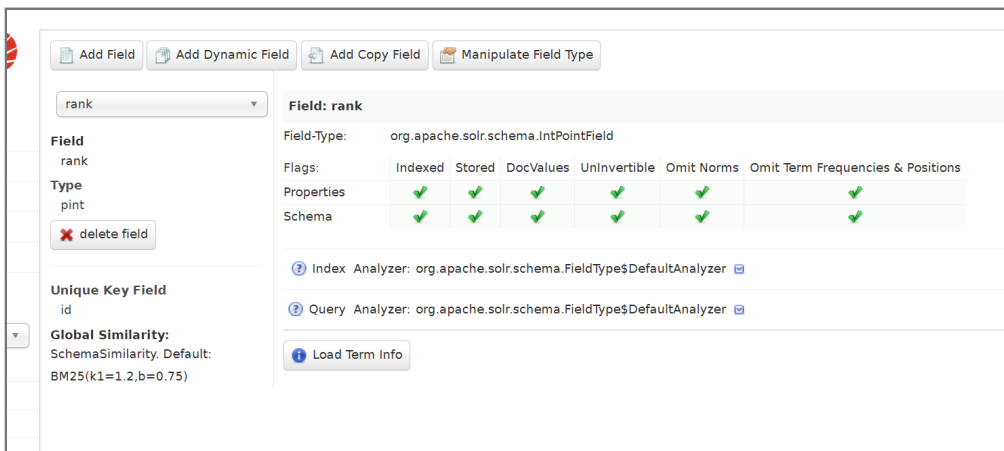
Please select ...

n

**Fields**

name

**Global Similarity:**  
SchemaSimilarity. Default:  
BM25(k1=1.2,b=0.75)



rank

**Field: rank**

Field-Type: org.apache.solr.schema.IntPointField

Flags: Indexed Stored DocValues UnInvertible Omit Norms Omit Term Frequencies & Positions

Properties

Schema

Index Analyzer: org.apache.solr.schema.FieldType\$DefaultAnalyzer

Query Analyzer: org.apache.solr.schema.FieldType\$DefaultAnalyzer

Load Term Info

**Field**  
rank

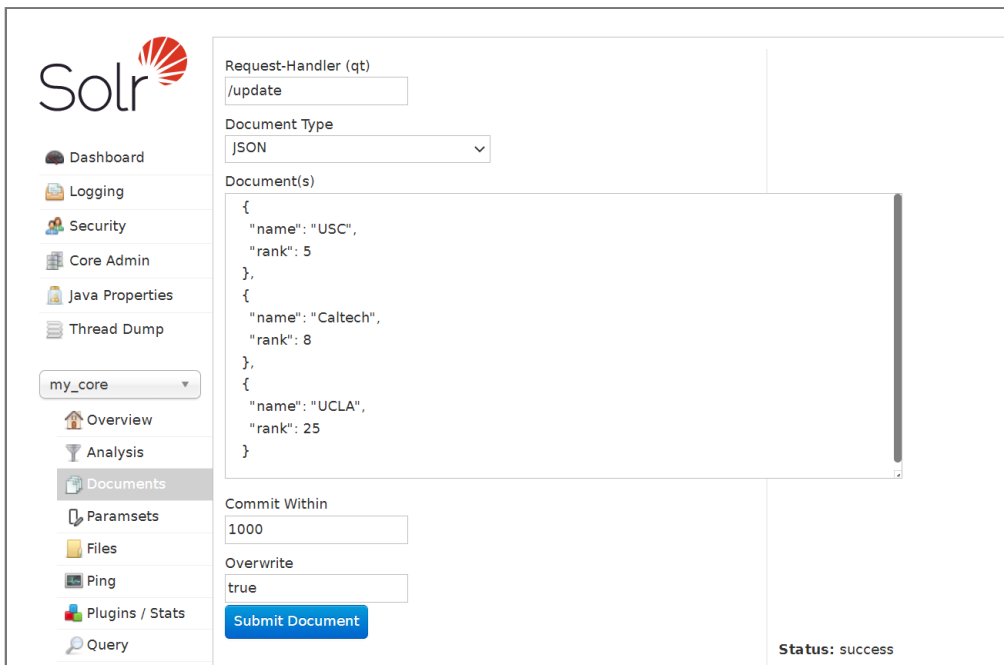
**Type**  
pint

delete field

**Unique Key Field**  
id

**Global Similarity:**  
SchemaSimilarity. Default:  
BM25(k1=1.2,b=0.75)

2. Add docs you'd like Solr to index, that correspond to the schema (note the non-standard syntax - we are NOT specifying valid JSON, instead we're adding a list of comma-separated JSONs (one for each doc), a lot like for Lunr (which called for a jsonl, ie. a list of JSONs without commas)):



Solr

- Dashboard
- Logging
- Security
- Core Admin
- Java Properties
- Thread Dump
- my\_core
- Overview
- Analysis
- Documents
- Paramsets
- Files
- Ping
- Plugins / Stats
- Query

Request-Handler (qt)  
/update

Document Type  
JSON

Document(s)

```
{
  "name": "USC",
  "rank": 5
},
{
  "name": "Caltech",
  "rank": 8
},
{
  "name": "UCLA",
  "rank": 25
}
```

Commit Within  
1000

Overwrite  
true

Status: success

Doing the above (pressing 'Submit Document') leads to our data getting indexed!

FYI if you ever want to clean out all the docs and start over [more here: <https://stackoverflow.com/questions/23228727/deleting-solr-documents-from-solr-admin>] - click on 'Submit Document' after typing the 'delete' command:

Request-Handler (qt)

/update

Document Type

Solr Command (raw XML or JSON) ▾

Document(s)

```
{'delete': {'query': '*:*'}}
```

Commit Within

1000

Overwrite

true

Submit Document

Status: success

Response:

```
{
  "responseHeader": {
```

3. Now we can search - via URL query syntax, among many other ways [more here: <https://solr.apache.org/guide/solr/latest/query-guide/standard-query-parser.html>]

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "q": "name:USC",
      "indent": "true"
    },
    "response": {
      "numFound": 1, "start": 0, "numFoundExact": true, "docs": [
        {
          "name": "USC",
          "rank": 5,
          "id": "a96db773-d51b-4d42-b9f0-b9b8aa019713",
          "_version_": 1762787125927870464
        }
      ]
    }
  }
}
```

Cool! And that's all there is to it, for Solr basics. If you exit your shell, that will stop Docker and also the webserver.

Do a simple search (like 'name:USC'). Next, do a range search (eg. schools with ranks between 5 and 15). For the range search you won't do it via a loop, like you did in Lunr under replit; instead, you'd use Solr's own syntax for that (look that up).

## Rubrics

Your (max 10) points will come from your submitting these:

- 2 points for: using the code in <https://bytes.usc.edu/~saty/tools/xem/run.html?x=lunr-cs572-hw4> to run the 'state' search on docs of 3 sizes and report result counts, index another field rather than the 'state' one, and do a sample search for it, submit a screenshot of the search result
- 3 points for: going off of <https://replit.com/@satychary/TrivialGrouchyScreenscraper> and modifying the .jsonl file to put in a different kind of data, doing a 'range' search and printing out the aggregated result as an array
- 2 points for: using StackBlitz [<https://stackblitz.com/edit/lunr-getting-started-z5rtim?file=index.ts>] to add paragraph text data from HW3, and doing two different searches of terms known to occur in multiple docs, submitting a screenshot for each
- 3 points for: running Solr under Docker, doing a simple query (field:value), and a 'range' one as well

**Getting help**

There is a hw4 'forum' on Piazza, for you to post questions/answers. You can also meet w/ the TAs, CPs, or me.

**Have fun! This is a quick+easy+useful+fun one!**

---