TEXT TAGGING WITH FINITE STATE TRANSDUCERS

David Smiley
Software Systems Engineer, Lead
Text Tagging with Finite State Transducers

David Smiley

Lucene/Solr Revolution 2013

© 2012 The MITRE Corporation. All rights reserved.
About David Smiley

- Working at MITRE, for 13 years
  - web development, Java, search

- Published 1st book on Solr; then 2nd edition (2009, 2011)

- Apache Lucene / Solr committer/PMC member (2012)


- Solr search consultant within MITRE and its sponsors, and privately
What is “Text Tagging” and “FSTs”?  

- *First*, I need to establish the context:  
  - JIEDDO’s OpenSextant project  
  - Though this presentation is *not* about OpenSextant or geotagging  

- Ultimately, I want to convey *how cool* Lucene’s FSTs are  
  - And you may have a need for a text tagger  
    - Or a geotagger (like OpenSextant)
OpenSextant

A DoD Funded Project: JIEDDO/COIC & NGA
Open Source approval recently obtained
A geotagging solution for unstructured text

Finds place name references in natural language
- “... I live near Boston ... ”
- Finds “Boston” with input character offset #s
- Often resolves to multiple gazetteer entries: “Boston” has 73

What’s a Gazetteer?
- A dictionary of place names with metadata like latitude & longitude
Select the output format:  KML  Shapefile  CSV  JSON  WKT

You can upload a file for geotagging:

File:  Choose File  no file selected

Upload file

Or you can type or paste text for geotagging:

Output file name:  OutputFromText

Submit text
How does it work?

Figure 2. The OpenSextant document processing pipeline.
The “Naïve” Tagger

- AKA “Text Tagger”
- *Simply* consults a dictionary/gazetteer; no fancy NLP
  - There’s nothing geospatial about it
  - Subsequent NLP processing eliminates low-confidence tags
- Actually, not so simple
  - Names vary in word length
  - Must find overlapping names
    - but not names within names
13 million place name records
- 8.1M distinct place names
- Why not 13M?
  - Ambiguous names (e.g. San Diego)
  - Text analysis normalization (e.g. diacritic removal, etc.)
- 2.8M are single-word names (1/3rd)
- 2.3 avg. words / name
- 14 avg. chars / name
3 Naïve Tagger Implementations

- GATE’s Tagger
  - In-memory Aho-Corasick string-matching algorithm
  - Requires an estimated 80 GB RAM !! (for our data)
  - FAST

- A JIEDDO developed MySQL based Tagger
  - “Reasonable” RAM requirements ~4GB
  - SLOW (~15x, 20x? not certain). ~1 doc/second

- A JIEDDO developed Solr/FST based Tagger …
Finite State Transducers

Applied to text tagging
Finite State Automata (FSA)

- **SortedSet<char[]>**: mop, moth, pop, slop, sloth, stop, top

Note: a “Trie” data structure is similar but only shares prefixes

- Shared prefixes and suffixes = reduced space
Finite State Transducer (FST)

- Adds optional output to each arc
- `SortedMap<char[], int>`
  - mop: 0, moth: 1, pop: 2, slop: 3, sloth: 4, stop: 5, top: 6
Lucene’s FST Implementation

- FST encoded as a byte[]
  - Memory efficient! And fast to load from disk.
- Write-once API (immutable)
  - Build minimal, acyclic FST from pre-sorted inputs
  - Fast (linear time with input size), low memory
  - Optional two-pass packing can shrink by ~25%
- SortedMap<int[],T>: arcs are sorted by label
  - getByOutput also possible if outputs are sorted
- [http://s.apache.org/LuceneFSTs](http://s.apache.org/LuceneFSTs)

Based on a Mihov & Maurel paper, 2001
My approach involves two layers of FSTs:

- **A word dictionary FST** to hold each unique word
  - Enables using integers as substitutes for char[]
    - Via `getByOutput(12345) -> “New”`

- **A word phrase FST** comprised of word id string keys
  - Ex: “New York City” -> [12345, 5522111, 345]
  - Value are arrays of gazetteer primary keys
Memory Use

- Word Dict FST:
  - 3.3M words with ordinal ids in **26MB** of RAM

- Name Phrase FST:
  - 8.1M word id phrases in **90 MB** of RAM

- Plus **82MB** of arrays of gazetteer primary key ids

- **Total: 198 MB** (compare to 80GB GATE Aho-Corasick)

- Building it consumes ~1.5GB Java heap, for 2 minutes
Experimental measurements

- Single FST Experiment
  - 1 FST of analyzed character word phrase -> int id
  - “new york city” -> 6344207
  - Theory: more than 2x the memory
  - Result: 69 MB! (compare to 26+90) **41% reduction**

- Retrospective: What I would have done differently
  - Index a field of concatenated terms (custom TokenFilter).
    - More disk needed but reduces build time & memory requirements. Unclear effect on tagging performance.
  - Potential to use MemoryPostingsFormat, a Lucene Codec that uses an FST internally + vInt doc ids, instead of custom FST code.
It's complicated! Single-pass (streaming) algorithm

- For each input word, lookup its ordinal id, then:
  1. Create an FST arc iterator for name phrase
  2. Append the iterator onto a queue of active ones
  3. Try to advance all iterators
     - Remove those that don’t advance

Iterator linked-list queue:
Head: New, York, City ✔
Head+1: York, City
Head+2: City ...
## Speed Benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Docs/Sec</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSextant: GATE Tagger</td>
<td>?</td>
<td>80</td>
</tr>
<tr>
<td>OpenSextant: MySQL based Tagger</td>
<td>1.1</td>
<td>4</td>
</tr>
<tr>
<td>OpenSextant: <strong>Solr/FST Tagger</strong></td>
<td>15.9</td>
<td>2*</td>
</tr>
</tbody>
</table>

Measures single-threaded performance of geotagging 428 documents in the “ACE” collection. OpenSextant tests all had the same gazetteer.
Integrated with Solr

- As a custom Solr Request Handler
  - Builds the FSTs from the index (the gazetteer)
  - Configurable
    - Text analysis (e.g. phonetic)
    - Exclude gazetteer docs by configured query
    - Optional partial word phrase matching
    - Optional sub-tags tagging

- Solr integration benefits
  - Solr as a taxonomy manager! Web-service, searchable, scalable, easy to update, ...
~$ curl -XPOST 'http://localhost:8983/solr/tag'
?fl=*&wt=json&indent=2' -H 'Content-Type:text/plain' -d "I live near Boston"
{
  "responseHeader":{
    "status":0,
    "QTime":1898},
  "tagsCount":1,
  "tags":[
    "startOffset",12,
    "endOffset",18,
    "ids",[1190927, 1099063, 2562742, 2667203, 2684629, 2695904, 2653982, 2657690, 11890986, 11891415]],
  "matchingDocs":{
    "numFound":73,"start":0,"docs":[
      {
        "id":12719030,
        "place_id":"USGS1893700",
        "name":"Boston",
        "lat":65.01667,
        "lon":-163.28333,
        "feat_class":"L",
        "feat_code":"AREA",
        "FIPS_cc":"US",
        "ISO_cc":["US"],
        "cc":"US",
        "ISO3_cc":"USA",
        "adm1":"US02",
        "adm2":"US02.0180",
        "name_bias":0.0,
        "id_bias":0.04,
        "geo":"65.01667,-163.28333"},
      ...
    ]
  }
}
Where can you get this?

- [https://github.com/openSextant/SolrTextTagger](https://github.com/openSextant/SolrTextTagger)
- An independent module of OpenSextant
  - Might seek incubator status at [http://www.osgeo.org](http://www.osgeo.org)
- Includes documentation, tests
Concluding Remarks

- **Lucene FSTs are awesome!**
  - Great for storing large amounts of strings in-memory
  - Or other string-like data: e.g. IP addresses, geohashes
  - The API is hard to use, however

- The text tagger should be useful independent of OpenSextant
  - Tag people/org names or special keywords
  - Might be ported to Lucene as an alternative to its synonym token filter

- I’ve got an idea on applying these concepts to Lucene “Shingling” as a codec to make it more scalable
CONFERENCE PARTY
The Tipsy Crow: 770 5th Ave
Starts after Stump The Chump
Your conference badge gets you in the door

TOMORROW
Breakfast starts at 7:30
Keynotes start at 8:30

CONTACT
David Smiley
dsmiley@mitre.org