Building A Billion Spatio-Temporal Object Search and Visualization Platform

Devika Kakkar, Benjamin Lewis
Goal

Develop a platform to make it easier for researchers to interactively explore large spatio-temporal datasets.
Initial focus on Geo-tweets
(could be any streaming dataset)

- 1-2% of tweets have GPS coordinates from the user’s device, currently about 1 million per day available via the Twitter API
- The CGA has been harvesting geo-tweets since 2012 and has an informal archive of about 8 billion objects
- Northeastern Professor Ryan Qi Wang also harvested during this period and we plan to eventually merge the two datasets to create a more complete version.
Requirements

• Develop back end and client to support interactive visualization of a billion point objects
• Support sub-second queries including heatmaps and temporal histograms
• Expose a general purpose RESTful API so other clients could access the data
• System should run on low cost commodity hardware or VMs
Big data visualization built on 2D faceting

Developed for HHypermap in 2015 (layer search)
Latest billion + long term archive

- Latest ~billion geo-tweets
  - Sharded Solr/Lucene
  - Streaming update
  - Interactive exploration
  - Approx 3 years of data

- Archive for persistent storage all tweets
  - Currently on Kafka
  - May move to Swift
  - Eventual access from Cloud Dataverse
  - Back to 2012
Demo
Logical High-Level Architecture

Data flows via **Apache Kafka**

HTTP

**Docker, Kontena, OpenStack**  **Hosting: Mass OpenCloud**
Apache Kafka

• Kafka: a scalable message/queue platform
• See new Kafka Streams & Kafka Connect APIs
• No back-pressure; can be a challenge
• Non-obvious use:
  – For storage; time partitioning
    • Lots of benefits yet serious limitations
Real-Time Harvesting

Connect to Twitter’s Streaming API → Stream tweets using predefined users and coordinates extent → If the tweet is Geotagged → Kafka Topic
Enrichment

Geo: Query Solr via spatial point query; attach related metadata to tweet
## Sentiment Analysis

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier</td>
<td>Support Vector Machine with Linear Kernel</td>
</tr>
<tr>
<td>Source Code</td>
<td>Python</td>
</tr>
<tr>
<td>Libraries</td>
<td>Scikit-learn, numpy, NLTK, scipy</td>
</tr>
<tr>
<td>Classes of sentiment</td>
<td>Positive (1) and Negative (0)</td>
</tr>
<tr>
<td>Training Corpus</td>
<td>Stanford Sentiment140, Polarity dataset v2.0, University of Michigan</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>Lower case, URLs, @user, #tags, trimming, repeating characters, emoticons</td>
</tr>
<tr>
<td>Stemming</td>
<td>Porter Stemmer</td>
</tr>
<tr>
<td>Precision, recall and f1-score</td>
<td>0.82 (82%)</td>
</tr>
<tr>
<td>Processing speed</td>
<td>20ms/tweet (no emoticon), 5ms/tweet(emoticon)</td>
</tr>
</tbody>
</table>
Sentiment Analysis

Phase 1: Training
- Train the classifier
- Save as pickle

Phase 2: Prediction
- Load the classifier
- Parse
- Preprocess
- Stem
- Predict

For each tweet
Sentiment Analysis

- Classifier: Support Vector Machine (SVM) with Linear Kernel
- Source code in Python
- Uses scikit-learn, numpy, scipy, NLTK
- Two classes of sentiment: Positive (1), Negative (0)
- Training Corpus: Sentiment140, Polarity dataset v2.0, University of Michigan
- Preprocessing: Lower case, URLs, @user, #tags, trimming, repeating characters, emoticons
- Stemming: Porter stemmer
- Precision, Recall, F1 score: 0.82 (82%)
- Processing speed: 20ms/tweet (no emoticon), 5ms/tweet (emoticon)
Solr for Geo Enrichment

“Reverse Geocoding”

- Tweets (docs) can have a geo lat/lon
- Enrich tweet with Country, State/Province, …
  - Gazetteer lookup (point-in-polygon)

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Features</th>
<th>Raw size</th>
<th>Index time</th>
<th>Index size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin2</td>
<td>46,311</td>
<td>824 MB</td>
<td>510 min</td>
<td>892 MB</td>
</tr>
<tr>
<td>US States</td>
<td>74,002</td>
<td>747 MB</td>
<td>4.9 min</td>
<td>840 MB</td>
</tr>
<tr>
<td>Massachusetts Census Blocks</td>
<td>154,621</td>
<td>152 MB</td>
<td>5.9 min</td>
<td>507 MB</td>
</tr>
</tbody>
</table>
Apache Solr

- Search / analytics server, based on Lucene
- Custom add-ons:
  - Time sharded routing (index + query)
  - LatLonPointSpatialField – in Solr 6.5
    - Faster/leaner search & sort for point data
  - HeatmapSpatialField – in Solr 6.6 TBD
    - Faster/leaner heatmaps at scale
**Time “Sharding”**

Solr has no built-in time based sharding.

A Solr custom “URP” was developed to route tweets to the right by-month shard. It auto creates and deletes shards.

A Solr custom “SearchHandler” was developed to decide which subset of shards to search based on custom parameters sent by the web-service.

Generally useful for others. Need more work for contribution to Solr itself.
The BOP Web-Service

• HTTP/REST API
  – Keyowrd search
  – Faceting
    • Heatmaps
  – CSV export

• Why not Solr direct?
  – Define a supported API
  – Ease of use for clients
  – Security

Tech:
  • Swagger
  • Dropwizard
  • Kotlin lang (on JVM)
Heatmaps: Spatial Grid Faceting

• Spatial density summary grid faceting, also useful for point-plotting search results
• Lucene & Solr APIs
• Scalable & fast usually…

• Usually rendered with a gradient radius ->
• See: http://spacemansteve.github.io/leaflet-solr-heatmap/example/index.html
UI Stack

• BOP’s UI uses the following technologies:
  – Angular JS
  – OpenLayers 3
  – npm (dependencies, script minification, development)
Deployment / Operations

• MassOpenCloud “MOC”
  – OpenStack based cloud (mimics Amazon EC2)
• CoreOS
• Kontena & Docker
• Admin/Ops tools:
  – Kafka Manager (Yahoo!)
  – Solr’s admin UI

Stats:
• 12 nodes (machines)
  • 5 to Solr
  • 3 to Kafka
  • 3 to enrichment, …
• 217 GB RAM
• 3500 GB disk
• 17 services (software pieces)
  • 133 containers
Docker

• Easy to find/try/use software
  – No installation
  – Simplified configuration (env variables)
  – Common logging
  – Isolated

• Ideal for:
  – Continuous Int. servers
  – Trying new software
  – Production advantages too
    • but “new”
Docker in Production

• We use “Kontena”
• Common logging, machine/proc stats, security
  – VPN to secure network; access everything as local
• No longer need to care about:
  – Ansible, Chef, Puppet, etc.
  – Security at network or proxy; not service specific
• Challenges: state & big-data
Next steps

• Persistent archive in Swift object storage
• Exploring adding analytic capabilities using GeoMesa
• Support faceting on numeric values (in addition to counts) to support other types of visualizations.
Thank you

Devika Kakkar
kakkar@fas.harvard.edu

Ben Lewis
blewis@cga.harvard.edu