Acknowledgments

● Global Event and Trend Archive Research (GETAR) project, supported by NSF IIS-1619028

● Integrated Digital Event Archiving and Library (IDEAL) project supported by NSF IIS-1319578

● CS5604 GTA Liuqing Li, Instructor Edward A. Fox
Team Members

Topic Analysis
Ashish Baghudana
Shruti Shetty
Aman Ahuja
Ashish Malpani

Clustering
Rammohan Chintha
Mo Yang
Pavan Bellam
Prathyush Sambaturu
Overview

Use topic analysis and clustering algorithms to find sub-themes and similar patterns in collections of webpages and tweets about real world events

- Pull and clean documents
- Topic Analysis
- Clustering
- Store results in HBase
Overview

Classify Documents into Real World Events

Preprocessing

Topic Analysis: LDA
- Topic Names
- Topic Probability

Clustering: K-means
- Cluster Name
- Cluster Probability

HBase
Topic Analysis
A (really) brief overview of topic models

Topic models discover “abstract” topics in a collection of documents where a topic is a group of “semantically similar” words.

- Mainly latent Dirichlet allocation (LDA) or its variants
  - Probabilistic algorithm
  - Optimized via Gibbs Sampling or Expectation Maximization

- LDA assumes documents a mixture over topics and topics a mixture over words
Tech Stack - Python++

Database Access
Data retrieval and upload pipelines written using the Python library happybase.

Topic Modeling
Preprocess tokenized data using NLTK for stopwords and punctuation removal. Allow custom pipelines. Use multi-core support from gensim for LDA.

Data Extraction & Visualization
Extract topics for each document and visualize topic clusters for each collection.

- happybase
- NLTK
- gensim
- PySpark
- pyLDAvis
Alternate Tech Stacks

- **Scala + Spark + MLLib** is an alternate stack
  - Might scale better for larger datasets
  - However, lacks the visualization and evaluation mechanisms that are built into gensim and pyLDAvis

- **Gotchas**
  - **PySpark + Spark + MLLib**
    - does not implement the class DistributedLDAModel
    - LDAModel does not implement several important methods such as: `topTopicsPerDocument()`, `logPerplexity()` and `topicCoherence()`

- **Python + Gensim** is best suited (multi-core support really helps!)
Preprocessing Pipeline (where the magic happens)

Built a fast and customizable preprocessing pipeline.

**Tokenizer**
- Use NLTK's `word_tokenize` if using `<document>:clean-text`
- Use `CommaTokenizer` or `SemicolonTokenizer` if using `<document>:clean-tokens`

**Mappers**
- Map tokens to a different form.
- Examples include:
  - `LowercaseMapper`
  - `PorterStemmer`
  - `WordnetLemmatizer`

**Filters**
- Filter out specific tokens.
- Examples include:
  - `StopwordFilter`
  - `PunctuationFilter`
  - `LengthFilter`
  - `ASCIIFilter`
  - `CollectionFilter`
Collection Filter - Customizing for each collection

Certain words occur repeatedly in each document and can be regarded as stop words for that collection

<table>
<thead>
<tr>
<th>Solar Eclipse 2017</th>
<th>Hurricane Irma</th>
</tr>
</thead>
<tbody>
<tr>
<td>solar</td>
<td>hurricane</td>
</tr>
<tr>
<td>eclipse</td>
<td>irma</td>
</tr>
<tr>
<td>totality</td>
<td>sept</td>
</tr>
<tr>
<td>tse</td>
<td>business</td>
</tr>
<tr>
<td>eclipse2017</td>
<td>sept</td>
</tr>
<tr>
<td>account</td>
<td>csbn</td>
</tr>
<tr>
<td>password</td>
<td>guardian</td>
</tr>
<tr>
<td>facebook</td>
<td>reuters</td>
</tr>
<tr>
<td>subnav</td>
<td>subscribe</td>
</tr>
<tr>
<td>twitter</td>
<td>us</td>
</tr>
<tr>
<td>aug</td>
<td>florida</td>
</tr>
<tr>
<td>published</td>
<td>bloomberg</td>
</tr>
<tr>
<td>username</td>
<td></td>
</tr>
<tr>
<td>aug</td>
<td></td>
</tr>
</tbody>
</table>
Developer Manual

Topic Modeling is end-to-end integrated as a single script with several built-in options


Run LDA on a given collection

required arguments:
  -c COLLECTION_NAME, --collection_name COLLECTION_NAME Collection name
  -t TOPICS [TOPICS ...], --topics TOPICS [TOPICS ...] Number of topics to run the model on
  -p, --preprocess Preprocess data
  --table_name TABLE_NAME Table name for HBase
  -hb, --hbase Get collection from HBase
  -f FILE, --file FILE File name for tokens

preprocessing arguments to be added when using -p flag:
  --tokenizer TOKENIZER Tokenizer to use
  --mappers MAPPERS [MAPPERS ...] Mappers to use
  --filters FILTERS [FILTERS ...] Filters to use
  --filter_words FILTER_WORDS Filename with words to filter out

optional arguments:
  -h, --help show this help message and exit
  -l LOGS, --logs LOGS Log directory
  -a ALPHA, --alpha ALPHA Alpha hyperparameter
  -b BETA, --beta BETA Beta hyperparameter
  -i ITER, --iter ITER Number of iterations
  --save_dir SAVE_DIR Save directory for topic models
Choosing the Right Number of Topics

Solar Eclipse Webpages

Hurricane Irma Webpages
Visualizing Topics
Naming Topics

Once we finalize on the number of topics, each “topic” has to be named. We follow two strategies:

● **Automatic Naming**
  ○ Choose the first non-repeating word in the topic-word distribution (sorted by probability)
  ○ Suitable for large number of topics

● **Manual Naming**
  ○ Look at top words manually to decide topic name
  ○ Suitable for small number of topics
## Solar Eclipse: Tweets

<table>
<thead>
<tr>
<th>eclipse</th>
<th>safety</th>
<th>pictures</th>
<th>experience</th>
<th>midnight</th>
<th>forecast</th>
<th>exo</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>view</td>
<td>photos</td>
<td>truly</td>
<td>catch</td>
<td>cloud</td>
<td>exo</td>
</tr>
<tr>
<td>moon</td>
<td>look</td>
<td>pictures</td>
<td>remarkable</td>
<td>breathtaking</td>
<td>shadow</td>
<td>totaleclipse</td>
</tr>
<tr>
<td>block</td>
<td>glass</td>
<td>photobomb</td>
<td>beautiful</td>
<td>mid</td>
<td>weather</td>
<td>thepowerofmusic</td>
</tr>
<tr>
<td>watch</td>
<td>don’t</td>
<td>timelapse</td>
<td>breathtaking</td>
<td>flight</td>
<td>path</td>
<td>planet</td>
</tr>
<tr>
<td>cover</td>
<td>eye</td>
<td>space</td>
<td>great</td>
<td>international</td>
<td>rain</td>
<td>message</td>
</tr>
<tr>
<td>totality</td>
<td>watch</td>
<td>lifetime</td>
<td>pretty</td>
<td>space</td>
<td>outside</td>
<td>verexo</td>
</tr>
<tr>
<td>circle</td>
<td>safely</td>
<td>live</td>
<td>happy</td>
<td>wow</td>
<td>forecast</td>
<td>que</td>
</tr>
</tbody>
</table>

These results aren’t perfect however. Despite strict filtering, we see words like *trumpresign* and *president* in the list.
## Computational Complexity

<table>
<thead>
<tr>
<th></th>
<th>Collection Size</th>
<th>Preprocessing</th>
<th>Model Creation</th>
<th>Log Perplexity</th>
<th>Topic Coherence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hurricane Irma</td>
<td>2714</td>
<td>56s</td>
<td>1m:07s</td>
<td>1m:01s</td>
<td>3s</td>
</tr>
<tr>
<td>(Webpages)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solar Eclipse</td>
<td>722</td>
<td>16s</td>
<td>41s</td>
<td>19s</td>
<td>2s</td>
</tr>
<tr>
<td>(Webpages)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solar Eclipse</td>
<td>2667726</td>
<td>11m28s</td>
<td>16m14s</td>
<td>24m13s</td>
<td>55s</td>
</tr>
<tr>
<td>(Tweets)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Models trained with 500 iterations and 10 topics each
+ Preprocessing is faster on local machines because of SSDs
HBase Fields

We populate:

- `topic:topic-list`
- `topic:probability-list`
- `topic:display-topicnames`

How these are used:

- The front-end team uses `display-topicnames` as a facet in their interface
- It is also possible to use probability scores for recommending similar documents
Shortcomings and Improvements

- Automatic elimination of collection-specific stop-words
  - Currently requires manual curation

- Crowd experiments to decide topic names and coherence
  - Topic names decided either naively or based on the experimenter’s judgement
  - Better to use the class strength to crowdsource annotations

- Topic modeling visualizations can be part of front end
  - pyLDAvis provides visualizations of the documents in a cluster via a MDS algorithm
  - Could be part of the faceted search

- Joint models for Tweets and Webpages
Clustering
A brief overview of Clustering

Clustering categorize data into clusters such that objects grouped in same cluster are similar to each other according to specific metrics

- **K-means Algorithm**
  - Elbow method to find number of K
  - Clustering based on cosine similarity

\[
sim(d_1, d_2) = \frac{\tilde{V}(d_1) \cdot \tilde{V}(d_2)}{||\tilde{V}(d_1)|| ||\tilde{V}(d_2)||}
\]
Tech Stack

- **DataBase Access:**
  Data retrieval and upload pipelines written using the Python library happybase.

- **Clustering:**
  Preprocess tokenized data using NLTK for *stopword* and *punctuation* removal. Scala and Spark for used for Clustering

- **Result Evaluation & Visualization:**
  Python and matplotlib are used for calculating inter-/intra-cluster similarity and plotting the results
Create a Scala Package using

```
sbt-package
```

And submit the generated jar to Spark

```
Spark-submit <jar file> <input file>
```

For large datasets, performance metrics like driver memory can be enhanced
Clustering Results

“Solar Eclipse” Tweets

- Cluster 0: 48.9%
- Cluster 1: 26.0%
- Cluster 2: 11.3%
- Cluster 3: 7.3%
- Cluster 4: 4.6%

“Solar Eclipse” Webpages

- Cluster 0: 31.7%
- Cluster 1: 20.1%
- Cluster 2: 15.9%
- Cluster 3: 6.8%
- Cluster 4: 22.3%
- Cluster 5: 3.2%
## Similarity Analysis

### “Solar Eclipse” Webpages

<table>
<thead>
<tr>
<th></th>
<th>Cluster 0</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
<th>Cluster 3</th>
<th>Cluster 4</th>
<th>Cluster 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 0</td>
<td>0.235892</td>
<td>0.057602</td>
<td>0.095294</td>
<td>0.077461</td>
<td>0.067748</td>
<td>0.087039</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>0.057602</td>
<td>0.579325</td>
<td>0.092020</td>
<td>0.083038</td>
<td>0.081589</td>
<td>0.076596</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>0.095294</td>
<td>0.092020</td>
<td>0.146127</td>
<td>0.105437</td>
<td>0.095530</td>
<td>0.099726</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>0.077461</td>
<td>0.083038</td>
<td>0.105437</td>
<td>0.176871</td>
<td>0.077173</td>
<td>0.086996</td>
</tr>
<tr>
<td>Cluster 4</td>
<td>0.067748</td>
<td>0.081589</td>
<td>0.095530</td>
<td>0.077173</td>
<td>0.111608</td>
<td>0.084908</td>
</tr>
<tr>
<td>Cluster 5</td>
<td>0.087039</td>
<td>0.076596</td>
<td>0.099726</td>
<td>0.086996</td>
<td>0.084908</td>
<td>0.714758</td>
</tr>
</tbody>
</table>

Average intra-cluster similarity: 0.33
Average inter-cluster similarity: 0.08
Naming Clusters by Frequent words

- K-means just returns clusters of documents, but does not name the clusters
- We can name each cluster by looking at a handful of documents in the cluster manually, which can be tedious in Big Data scenario
- Therefore,
  - we determine the most frequent words in all documents within a cluster
  - name the cluster based on a few very frequent words in the cluster
  (Assumption: The frequent words in the cluster describe the information in the documents within the cluster)
## Clusters in “Solar Eclipse” Tweets and Webpages

<table>
<thead>
<tr>
<th>Cluster ID</th>
<th>Cluster Name for Tweets</th>
<th>Cluster Name for Webpages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DiamondRing</td>
<td>EclipseChasers</td>
</tr>
<tr>
<td>1</td>
<td>WatchEclipse</td>
<td>AjcEclipseNews</td>
</tr>
<tr>
<td>2</td>
<td>SafeEclipse</td>
<td>EclipseScience</td>
</tr>
<tr>
<td>3</td>
<td>ExoPlanetMusic</td>
<td>BusinessInsiderEclipseArticles</td>
</tr>
<tr>
<td>4</td>
<td>MidFlightEclipse</td>
<td>Eclipseville</td>
</tr>
<tr>
<td>5</td>
<td>NonEnglish</td>
<td>MuseumEclipse</td>
</tr>
</tbody>
</table>
HBase Fields

We populate:

- `cluster:cluster-list`
- `cluster:cluster-probability`
- `cluster:display-clusternames`

Script and Command:

- `Python hbase_write_cluster.py [Cluster_Result].csv`
- `Cluster_Result: Document_ID, Cluster_Name, Cluster_Probability`
Shortcomings and Improvements

- **Cluster Probability**
  - We set it as “1” because we only did hard clustering

- **Comparison with other clustering algorithm**
  - Hierarchical Clustering
  - Density-based Clustering
Questions?

Code Repository: https://github.com/ashishbaghudana/cs5604
https://github.com/yuan9/CS5604_CTA