Final Report
CS 5604: Information Storage and Retrieval

Solr Team
Abhinav Kumar, Anand Bangad, Jeff Robertson, Mohit Garg,
Shreyas Ramesh, Siyu Mi, Xinyue Wang, Yu Wang
January 16, 2018

Instructed by Professor Edward A. Fox

Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
Abstract

The Digital Library Research Laboratory (DLRL) has collected over 1.5 billion tweets and millions of webpages for the Integrated Digital Event Archiving and Library (IDEAL) and Global Event Trend Archive Research (GETAR) projects [6]. We are using a 21 node Cloudera Hadoop cluster to store and retrieve this information. One goal of this project is to expand the data collection to include more web archives and geospatial data beyond what previously had been collected. Another important part in this project is optimizing the current system to analyze and allow access to the new data. To accomplish these goals, this project is separated into 6 parts with corresponding teams: Classification (CLA), Collection Management Tweets (CMT), Collection Management Webpages (CMW), Clustering and Topic Analysis (CTA), Front-end (FE), and SOLR. This report describes the work completed by the SOLR team which improves the current searching and storage system. We include the general architecture and an overview of the current system. We present the part that Solr plays within the whole system with more detail. We talk about our goals, procedures, and conclusions on the improvements we made to the current Solr system. This report also describes how we coordinate with other teams to accomplish the project at a higher level. Additionally, we provide manuals for future readers who might need to replicate our experiments. The main components within the Cloudera Hadoop cluster that the SOLR team interacts with include: Solr searching engine, HBase database, Lily indexer, Hive database, HDFS file system, Solr recommendation plugin, and Mahout. Our work focuses on HBase design, data quality control, search recommendations, and result ranking.

Overall, through out the semester, we have processed 12,564 web pages and 5.9 million tweets. In order to cooperate with Geo Blacklight, we make major changes on Solr schema. We also play as a data quality control gateway for the Front End team and deliver the finalized data for them. As to search recommendation, we provide search recommendation such as MoreLikeThis plugin within Solr for recommending relative records from search results, and a custom recommendation system based on user behavior to provide user based search recommendations. After the fine tuning over final weeks of semester, we successfully allowed effective connection of results from data relative teams and delivered them to the front end through a Solr core.
# Contents

Abstract .......................................................... 2

List of Tables ...................................................... 5

List of Figures ..................................................... 6

1 Overview ........................................................ 7
    1.1 Management .................................................. 7
    1.2 Challenges Faced ............................................ 7
    1.3 Process ....................................................... 7
    1.4 Solutions Developed ....................................... 8
    1.5 Future Work ................................................ 9

2 Literature Review ............................................. 11

3 Requirements .................................................. 12

4 Design .......................................................... 14
    4.1 Previous System Design ................................... 14
    4.2 Current Design ............................................. 15
        4.2.1 HBase Schema ......................................... 15
        4.2.2 Quality Control ....................................... 15
        4.2.3 Recommender Systems ................................. 16

5 Implementation ............................................... 19
    5.1 Overview .................................................... 19
    5.2 Timeline ..................................................... 19
    5.3 Milestones and Deliverables .............................. 21
    5.4 SNER GeoLocation .......................................... 22
    5.5 Results ........................................................ 22

6 User Manual ................................................... 23
    6.1 Install Solr system ......................................... 23
        6.1.1 Available Solr packages .............................. 23
        6.1.2 Start Solr ............................................... 23
        6.1.3 Check if Solr is running .............................. 23
    6.2 Solr Admin Interface ....................................... 23
    6.3 Solr Query ................................................... 24
        6.3.1 Query Parameters ...................................... 25
        6.3.2 Query Syntax .......................................... 26
        6.3.3 Function Query ........................................ 27
    6.4 Spatial Search .............................................. 28
    6.5 Document Recommendation ................................ 29
        6.5.1 MLT Syntax ............................................. 30
6.5.2 CTACFA Handler Syntax .................................................. 30
6.5.3 Solr Admin UI ............................................................... 30
6.5.4 Relevancy Boosting ....................................................... 32

7 Developer Manual ................................................................. 33
7.1 Solr ................................................................................. 33
7.1.1 Architecture ................................................................. 33
7.1.2 Installation ................................................................. 34
7.1.3 Tutorial ................................................................. 35
7.2 HBase ........................................................................... 43
7.2.1 Architecture ................................................................. 43
7.2.2 Tutorial ................................................................. 44
7.3 Lily HBase Indexer ............................................................ 46
7.3.1 Overview ................................................................. 46
7.3.2 Live Mode Indexing ......................................................... 46
7.3.3 Batch Mode Indexing ...................................................... 48
7.3.4 Incremental Indexing Using NRT Indexer ...................... 49
7.4 Recommender Systems ...................................................... 50
7.4.1 Architecture ................................................................. 50
7.4.2 Configuration for Handler ............................................ 51
7.4.3 Tutorial for Personalized Recommendation .................. 51

8 Cluster Developer Manual ....................................................... 53
8.1 Access Cluster Solr Web UI ............................................... 53
8.1.1 Secure Pipes ................................................................. 53
8.1.2 SSH Tunneling ............................................................... 53
8.2 Command Examples on Cluster ......................................... 54
8.2.1 Solr Commands ............................................................. 54
8.2.2 Lily Indexer Commands .................................................. 54
8.2.3 Morphline.conf Reference ........................................... 54

Bibliography ............................................................................. 56

Appendix A: HBase Schema ..................................................... 57
List of Tables

1. Improvements Compared to Last Year’s Work ........................................... 13
2. Tasks and Timeline ..................................................................................... 19
3. Milestones ................................................................................................. 21
4. Deliverables ............................................................................................... 21
5. Key results of Solr team ........................................................................... 22
6. Core Query Parameters ............................................................................ 25
7. Common Query Parameters .................................................................... 25
8. Common MLT Parameters ....................................................................... 29
9. HBase Schema .......................................................................................... 57
List of Figures

1. Process followed by Solr team ......................................................... 8
2. Position of Solr in the data flow ...................................................... 9
4. An overview of the schema in HBase for our project ......................... 15
5. MovieLens Sample ........................................................................ 16
6. Solr Admin Dashboard .................................................................... 24
7. MoreLikeThis .................................................................................... 31
8. CFACTA Handler ............................................................................. 32
9. The Solr Architecture ..................................................................... 33
10. Get a list of all the documents where the screen name ends with ‘a’ .... 37
11. Get a list of all the documents where the screen name ends with ‘a’ and language is English ................................................................. 38
12. Get a list of all the documents whose language is English. The documents should be sorted by created_time_dt. .................................... 39
13. Get 2 documents where the language is English ............................... 40
14. Fetch documents where the language is English. Response should only contain screen names .............................................................. 41
15. Fetch all the documents where language is English. Use XML for the response type. ................................................................. 42
16. The Architecture of HBase ............................................................... 43
17. Get the small Twitter collection for testing ..................................... 44
18. HDFS ls example ........................................................................... 44
19. Hadoop status check ...................................................................... 45
20. HBase create table example ............................................................ 45
21. Recommendation System Architecture ........................................... 50
1 Overview

1.1 Management

There are 8 members in our team, so team management is crucial to accomplish the project goal. In general, we think working in pairs is a great strategy for making progress. Therefore, we tend to divide our tasks into four subtasks and assign a pair of teammates to each one. Including class sessions, we met 3 times a week to communicate about our progress and plan ahead for future work. Because SOLR acts as the interface between the data processing teams and the front end developers, we also use class sessions to coordinate schema design with other teams.

We have adopted several tools within the group to achieve efficient collaboration. We use Google Team Drive for documentation and file exchange, Slack for team communication, and GitHub for code collaboration and project management.

1.2 Challenges Faced

The first major challenge that we faced was learning all of the technologies involved. Components inside the Cloudera Hadoop ecosystem which are new to us and are relevant to this project include Solr, Lily, HBase, and HDFS. For this reason, we needed an efficient plan to overcome the difficult learning curve; we will discuss this in the next section.

Along with learning new technologies, reviewing the relevant literature has consumed a large amount of our time. Because our main goal is to optimize the current system based on previous development [13], we need to have a thorough understanding of the pros and cons of previous work. Additionally, all relevant technologies are under active development at the quick pace of the tech industry, so we need to keep up with new solutions that are potentially beneficial to our system that might not have existed for previous work on this project.

Furthermore, the SOLR team plays a crucial role coordinating among all the teams. With respect to data storage, we are responsible for communicating with the CLA, CMT, CMW, and CTA teams to design the HBase schema to facilitate data exchange. We also need to work closely with the FE team to optimize the quality and speed of searches to provide a good user experience. We need to consistently talk to each team so that we can agree on the project plan at a high level.

One of the most important and unique challenges we faced this year was regarding the schema development and indexing. These challenges owe to the fact that we had to ingest data from two input sources, i.e., web pages and tweets, and support two clients, i.e., GeoBlacklight and Visualization. To explain this problem let us take an example. GeoBlacklight requires a field called ‘dc_title_s’ which holds the title of the returned document. For a web page this field is mapped to the web page:title column, while for tweets it is mapped to tweet:user_screen_name. To solve this we have to create separate morphline files for web pages and tweets, and index them separately.

1.3 Process

In this section we will give a brief overview of the steps we followed to get our system up and running. We started with creation of an HBase schema. The CMT, CMW, CLA, and CTA teams use this schema to provide the data required by front end systems. We then created a Solr schema
which strictly follows the requirements of both GeoBlacklight and Visualization team. Once the schema file is created we index the data present in HBase into Solr. This is done using a Cloudera CDH map reduce process. We use Morphline as our ETL (Extract Transform Load) pipeline to perform various kinds of cleaning and transform operations. Once all the data is indexed we present the Solr core to the front end systems for review. The front end team evaluates the contents of the core and provides feedback. Feedback can include request of new fields, changing field names, or changing the format of field values. We then go through the entire cycle again to incorporate the feedback.

![Figure 1: Process followed by Solr team](image)

1.4 Solutions Developed

To facilitate learning the new technologies, we divided our team into 4 pairs. Each of the four pairs was responsible for one of the tools involved: HBase, Lily Indexer, Solr functionality, and Solr queries. Each pair also created a user manual for their corresponding part. During our meetings, we put our progress together and go through the whole process so that everyone achieves the same understanding about the whole system. We have one powerful machine set up with Cloudera VM and a team server with partial functionality including Hadoop, HBase, and Solr running. We ran through all processes on our machines to test our work and to be prepared to operate on the real cluster. At the same time, each pair also worked on the literature review for their part, to find potential improvements.

In order to have successful team management, we also created a task table that tracks team tasks. It is shown in detail in the implementation section.
To coordinate with the other teams, we actively talked with the data collection and analysis teams to agree on the HBase schema design. We have also gotten feedback from the FE team about how they would like to perform queries which could be built on in future work.

1.5 Future Work

One aspect of this project we hope to introduce which was not covered in previous work is handling geospatial data. We will need to work closely with other teams on the processing pipeline to be able to fully utilize this unique type of data. We will work on providing fast queries along with corresponding geospatial coordinate data for the FE team.

Additionally, we focused on the recommender system, which is significantly based on previous development. We worked on this in two components: Solr's More-Like-This plugin and Association rules. Solr’s More-Like-This can help us when users are visiting some documents. It can generate some relevant documents based on text similarity because users may be interested in closely related content. But only supporting More-Like-This is not enough. We apply association rules, which can help us when two users have overlapping interests in topics, so we can recommend to users
other topics which they may have not been seen before.

We also considered linking Solr with the Hive data storage system. Hive is a powerful data search engine based on SQL operation within the Cloudera Ecosystem. Hive performs better for relational databases than HBase does due to its support of SQL MapReduce operations. Because of this, Hive could potentially provide significant analytical capabilities that we could make use of. However, no other teams expressed any interest in using Hive, so we did not pursue getting it to work.
2 Literature Review

The textbook “Introduction to Information Retrieval” [14] provides a strong theoretical background on many concepts that are important for us in understanding information retrieval. For example, Chapter 4 explains how to construct indices. There are several ways to pre-process documents so as to speed up their retrieval. All of these fall under the general term ‘indexing’: an index is a structure that facilitates rapid location of items of interest, an electronic analog of a book’s index. We learned about single-pass in-memory indexing, an algorithm that has beneficial scaling properties because it only handles as much text as can fit in main memory at one time, allowing it to be used on arbitrarily large corpora. We also learned about indexing in distributed systems and dynamic indexing which can help us understand Solr and Lily. We hope that using these ideas will allow us to better optimize Solr. In Chapter 5, we learned index compression which can help us reduce disk space. It also has some more subtle benefits, such as increasing the use of caching and facilitating transferring data from disk to memory. From Chapter 8, we learned how to evaluate information retrieval systems which can help us get feedback every time we make some change in Solr and show the experiment results to others.

From Apache Solr’s official website [2], we learned about many of the features Solr provides. For example, Solr has advanced full-text search capabilities, powered by Lucene. Solr has powerful matching capabilities including phrases, wildcards, joins, grouping, and much more across multiple data types. Also, Solr has near real-time indexing. This takes advantage of Lucene’s Near Real-Time Indexing capabilities to make sure you see content immediately after it has been processed. On top of that, Solr provides an extensible plugin architecture which makes it easy to plug in both index and query time plugins.

In the official reference guide [3], there are detailed instructions about how to use Solr, and some basic data operations for Solr. For example, it shows how to use the Solr administration user interface and how to perform index operations such as commit, optimize, and rollback.

From Wide & Deep Learning for Recommender Systems [11], we learned about jointly trained wide linear models and deep neural networks, and how to combine the benefits of memorization and generalization for recommender systems. We were inspired by this idea while implementing the recommender system.

We also benefited greatly from the report from this same project from last year [13]. It covers many of the same things that we present here. See Table 1 for a comparison between our work and the work completed in 2016.
3 Requirements

This is a list of requirements that we hoped to satisfy through this project.

1. Build a Solr schema to index data from the HBase column families populated by the data collection and analysis teams.

2. Configure the Lily indexer to add the support of Near Real Time (NRT) indexing for the data received from the HBase column families. This requires updating Lily configuration files without breaking the current batch indexing support.

3. Create a custom ranking function in the existing Solr query processor using the data from the new HBase column families to improve the Solr search query results.

4. Create a Recommender System which makes use of content as well as user information. We will use classification and clustering results for building this recommender system. Use the MoreLikeThis tool and develop some supplementary recommendation algorithms to provide recommendation system functionality.

5. Implement GeoSpatial Search – Using this we will be able to index points or other shapes and sort or boost scoring by distance between points.

6. Create Solr logs which contain user access data. The info might include user names, their login/logout information, search queries, and documents or tweets clicked on.

7. Optimize the Solr ranking function to give accurate results.

8. Link Solr with the Hive data storage system.

Hive can serve better for relational database operations than HBase because it allows for querying data stored on HDFS for analysis via HQL, an SQL-like language that gets translated to MapReduce jobs. Hive could be used for analytical querying of data collected over a period of time for tasks such as calculating trends or processing web page logs. These might be done using off-line analytics instead of real time querying.

Here we raise one example that could possibly be implemented through Hive. Hive can store user action logs from the FE team and be utilized by offline recommendation systems to do some analysis job. Then, result data could be populated into HBase for Solr to access.

Currently we will list this as lower priority work since no team claimed such need for Hive storage. As to the recommendation system, we made an agreement with the FE team on using the MySQL database instead of Hive. If any team thinks of a good use for Hive and intends to use it, please contact us.

Table 1 represents our requirements compared to last year [13]; fields with star symbols indicate areas yet to be completed:
Table 1: Improvements Compared to Last Year’s Work

<table>
<thead>
<tr>
<th></th>
<th>Fall 2016</th>
<th>Fall 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema.xml</td>
<td>Fine Grained</td>
<td>Improved code convention</td>
</tr>
<tr>
<td></td>
<td>No web page fields</td>
<td>web page and Classification fields</td>
</tr>
<tr>
<td>morphlines.xml</td>
<td>Support Type: String, Text, Date</td>
<td>Support Type: String, Date, Text, Float</td>
</tr>
<tr>
<td></td>
<td>Duplicate index fields</td>
<td>Duplicates removed</td>
</tr>
<tr>
<td>Collection Indexed</td>
<td>1.2 billion tweets collection</td>
<td>5.9 million tweets collection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 thousand Web pages collection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse Tweet collection</td>
</tr>
<tr>
<td>Recommendation</td>
<td>MLT Handler</td>
<td>MLT Handler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTACFA Handler</td>
</tr>
<tr>
<td>Facet Search</td>
<td>Detailed facet search</td>
<td>Detailed facet search</td>
</tr>
</tbody>
</table>
4 Design

4.1 Previous System Design

The previous system design of IDEAL and GETAR [6] is shown in Figure 3. The system enables us to collect tweets and web pages for storage and analytics. The components shown with a solid black arrow represent the data flows that have been already set up, while the dashed blue lines represent the data flows to be completed. The crawled data is stored in HDFS in its original format. The Collection Management (CMW and CMT) teams then normalize this data using linguistic processing and tokenization. They also assign UUIDs based on the hash of the URL and timestamp, then use built in tools of HDFS to insert these documents into HBase. While the rest of the teams run their analysis on this data, the Solr team assists them by providing the means to index the data. Our goal is to use the Lily indexer to further optimize queries and to improve query results by improving the relevancy sorting and recommendation features.

![Figure 3: Architecture of the IDEAL Project [13]](image)

In summary, we studied the previous design [13] and found that we can build on top of this architecture to provide additional functionality as well as performance improvements.
4.2 Current Design

4.2.1 HBase Schema

Conceptually, we organized the schema for HBase into 3 main parts: Metadata, Source Data, and Generated Data.

Under Metadata, we store doc-type and collection information. Under Source Data, we have two sub-parts, one for tweets and one for web pages, because the data we collect for tweets and web pages is sufficiently different that it warrants separate formats. In each sub-part we store raw data and pre-processed or “clean” data. Under Generated Data, we store the data generated by other teams such as topic data and cluster data from the CTA team, and Classification data from the CLA team.

4.2.2 Quality Control

In previous work, many basic functionalities of the morphline process have been done. Beyond these, we plan to take more advantages of morphline process in the pipe line relating to indexed data processing.

As the Solr team delivers the finalized indexed data to the FE team, we play an important part in data quality control process. Thus, we take the advantage of integrated Java support in the morphline process to validate and fix our indexed data in various ways as follows:

- Convert and provide timestamps in the format as requested from the FE team
- Convert necessary data to different types (e.g., string to float)
- Provide hot fix on improper data format in HBase (e.g., remove unnecessary tags)
- Drop records in different scenarios (e.g., bad URL status code)
• Remove unnecessary fields
• Add/Remove values to certain fields
• Sanitize unknown Solr fields that could cause Solr errors
• Log record at debug level to SLF4J

(Code examples could be found in the morphlines.conf section within the Cluster Developer Menu.)

Beyond what we have done, there are more things that could potentially be implemented:

• Produce GeoIP information through indexing
  (In this case CMW/CMT needs to provide IP addresses)

• Use Grok to extract structured data from unstructured data.
  Note: The grok command uses regular expression pattern matching to extract structured fields from unstructured log data. A good practice will be extracting information from system logs, server logs, or database logs.

4.2.3 Recommender Systems

The Solr team will develop a new ranking function for the existing custom Solr query processor to provide better search results using the additional data provided by the other teams. From the FE team, we need to use user profile data. From the CTA and CLA teams we will use tweet and webpage topics, clusters, and classes. This will allow us to build the recommendation system based on those results and user data.

For recommendation based on text similarity, we used the MoreLikeThis handler, so that we can recommend those documents that are more similar, based on text similarity.

For recommendation based on clustering and classification results, we made a handler that, given an ID of one document, recommends other documents that have the same clustering and classification result. This is a more general recommendation method, and provides more diverse choices for users.

For the user-user collaborative filtering algorithm, the main steps we need are:

1. Get the user rating data. Because we did not get any user action data from the FE team into our system due to the cold start problem, we used sample data provide by MovieLens as an example. This is shown in the Figure 5.

<table>
<thead>
<tr>
<th>userid</th>
<th>movield</th>
<th>rating</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>12882</td>
<td>1</td>
<td>4</td>
<td>1147195252</td>
</tr>
<tr>
<td>12882</td>
<td>32</td>
<td>3.5</td>
<td>1147195307</td>
</tr>
<tr>
<td>12882</td>
<td>47</td>
<td>5</td>
<td>1147195343</td>
</tr>
<tr>
<td>12882</td>
<td>50</td>
<td>5</td>
<td>1147185499</td>
</tr>
<tr>
<td>12882</td>
<td>110</td>
<td>4.5</td>
<td>1147195239</td>
</tr>
<tr>
<td>12882</td>
<td>150</td>
<td>3.5</td>
<td>1147195267</td>
</tr>
<tr>
<td>12882</td>
<td>158</td>
<td>2</td>
<td>1147185180</td>
</tr>
<tr>
<td>12882</td>
<td>165</td>
<td>4</td>
<td>1147195325</td>
</tr>
<tr>
<td>12882</td>
<td>260</td>
<td>4</td>
<td>1147195260</td>
</tr>
<tr>
<td>12882</td>
<td>296</td>
<td>5</td>
<td>1147195153</td>
</tr>
</tbody>
</table>

Figure 5: MovieLens Sample
2. Calculate users’ cosine similarity. Cosine similarity is defined as:

\[ \cos(u, v) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|_2 \|\vec{v}\|_2} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2} \sqrt{\sum_i v_i^2}} \]

(1)

Here the numerator represents the sum of products of user u’s and user v’s normalized (mean-centering) ratings for all products, and the denominator represents the product of the normalized values of user u’s ratings and user v’s ratings.

3. Based on similarity score, choose the top 30 nearest neighbors.

4. Use mean-centering to normalize ratings for scoring.

\[ p_{u,i} = \mu_u + \frac{\sum_{v\in N(u|i)} \cos(u, v) (r_{v,i} - \mu_v)}{\sum_{v\in N(u|i)} |\cos(u, v)|} \]

(2)

5. Sort all predicted ratings and recommend the top n to the user.

For recommendation based on associative rules we create a personal recommender system. When users share common search interests they will be recommended documents that they haven’t directly searched for, but were searched for by users with common search interests. Below are two users A and B and their search history along with recommendation.

A: Hurricane, Houston, safety, shooting - Recommendation - vegas, hospitals
B: Houston, safety, vegas, hospitals - Recommendation - Hurricane, shooting

Such data is updated in the database, as provided by the FrontEnd team. The columns are UserId, Keywords, and Recommendation, separated by |

For the last user there are no matchings, hence nothing has been recommended to the user.

The algorithm is such that it goes over all user keywords and stores the unmatched keywords corresponding to matched key patterns. Here 4 recommendations are stored, once this is achieved, the next user is searched. This code changes the data base both dynamically and through a static fashion.

A Solr query is created in which a user’s ID is entered which returns the recommended documents as response. The query is /recommendation?id=d98745 The query handler is unable to work on the cluster due to request permissions, but works on a local machine.

We update the database provided by the Front End Team by adding values to a recommendation column. We faced issues on running the handler request for the cluster.
public class RequestHandler extends RequestHandlerBase {

    @Override
    public void handleRequestBody(SolrQueryRequest solrQueryRequest, SolrQueryResponse solrQueryResponse) throws Exception {
        Recommendation recommendation = new Recommendation();
        String id = solrQueryRequest.getParams().get("id");
        String recommendedKeywords = recommendation.ABC(id);
        solrQueryResponse.addResponse(recommendedKeywords);
    }

    @Override
    public String getDescription() {
        return "The request handler returns recommendations based on similarity between user views";
    }
}
5 Implementation

5.1 Overview

We have had discussion with the GTA and other teams to get a better understanding of our tasks. Based on the current setup of the Hadoop Cluster and previous work [13], our team hoped to improve the infrastructure and add more functionality for a better search job. Our approach was as follows:

1. Become familiar with the technologies by following the tutorials and the instructions written by the people who worked on this previously [13].

2. Understand the existing data infrastructure.

3. Coordinate with all teams to design the HBase schema. We are working with the other teams to improve the current schema file and make it clear how the column families and columns are stored in HBase.

4. Index all of the tweets and webpages provided by the other teams.

5. Ensure fast query handling for all content by way of distributed search using multiple Solr nodes.

6. Develop good similarity computation methods that consider all available additions to the raw content.

7. Improve recommendation capabilities (done by a sub-group of the Solr team).

8. Collaborate with the FE team to understand their requirements and design a system to provide them an easy way for the user to access data indexed by SOLR.


5.2 Timeline

Table 2 shows our schedule. It contains the task description, timeline week, member responsible for accomplishing the task, and the current status. This schedule has been added to and changed over time.

Table 2: Tasks and Timeline

<table>
<thead>
<tr>
<th>Task</th>
<th>Timeline (week)</th>
<th>Assignee</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up Solr team</td>
<td>1</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Set up Solr on local machine and do tutorial</td>
<td>1</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Set up shared folder using Google Drive for sharing, Slack for instant messaging, and Github for project management</td>
<td>1</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Divide our team into four subgroups Solr, Lily indexer, HBase, and interaction with FE</td>
<td>2</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Review HBase and design schema</td>
<td>2</td>
<td>Jeff, Yu</td>
<td>DONE</td>
</tr>
<tr>
<td>Task</td>
<td>Week</td>
<td>Student(s)</td>
<td>Status</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
<td>------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Review Lily Indexer</td>
<td>2</td>
<td>Xinyue, Anand</td>
<td>DONE</td>
</tr>
<tr>
<td>Review Solr main mechanism</td>
<td>2</td>
<td>Abhinav, Shreyas</td>
<td>DONE</td>
</tr>
<tr>
<td>Review Solr queries and usage and interaction with FE</td>
<td>2-3</td>
<td>Siyu</td>
<td>DONE</td>
</tr>
<tr>
<td>Go through the basic process of importing data into HBase and perform basic searching using Solr inside a VM:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Import the original small collection into HDFS and HBase</td>
<td>3</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>2. Use Lily Indexer to map columns in HBase with fields in Solr and perform a basic query test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Use Lily Indexer to map columns in HBase with fields in Solr (Live Mode)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Use Lily Indexer to accomplish the offline indexing job (Batch Mode)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Create a Solr instance for the original small collection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send out initial schema to all data processing teams and exchange opinions</td>
<td>3</td>
<td>Jeff</td>
<td>DONE</td>
</tr>
<tr>
<td>Set up server given by Dr. Fox with HBase, Lily, Solr working</td>
<td>3</td>
<td>Jeff</td>
<td>DONE</td>
</tr>
<tr>
<td>Prepare for team presentation 1</td>
<td>3-4</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Draft interim report 1</td>
<td>4</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Communicate with other teams and make agreement on how to use geospatial data in the pipeline</td>
<td>4-5</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Learn about Solr’s Morelikethis component for further improving the recommendation system</td>
<td>5-8</td>
<td>Yu, Xinyue</td>
<td>DONE</td>
</tr>
<tr>
<td>Learn about Mahout for further improving the recommendation system</td>
<td>5-8</td>
<td>Yu</td>
<td>DONE</td>
</tr>
<tr>
<td>Learn how to use Hive and how to connect it to Solr</td>
<td>5-8</td>
<td>Xinyue</td>
<td>HOLD</td>
</tr>
<tr>
<td>Learn the knowledge of faceted search</td>
<td>5-8</td>
<td>Siyu</td>
<td>DONE</td>
</tr>
<tr>
<td>Further design for HBase schema and communicate to other teams</td>
<td>7</td>
<td>Jeff</td>
<td>DONE</td>
</tr>
<tr>
<td>Create small test collection on cluster</td>
<td>7</td>
<td>Abhinav, Shreyas, Xinyue</td>
<td>DONE</td>
</tr>
<tr>
<td>Create Eclipse test collection on cluster</td>
<td>8</td>
<td>Abhinav, Shreyas</td>
<td>DONE</td>
</tr>
<tr>
<td>Prepare for team presentation 2</td>
<td>8</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Draft interim report 2</td>
<td>8</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Relevancy boosting</td>
<td>8</td>
<td>Yu, Xinyue</td>
<td>DONE</td>
</tr>
<tr>
<td>User-user collaborative filtering algorithm implementation</td>
<td>8-11</td>
<td>Yu</td>
<td>DONE</td>
</tr>
<tr>
<td>Help FE team to solve query problem in Solr</td>
<td>8-11</td>
<td>Siyu, Abhinav</td>
<td>DONE</td>
</tr>
<tr>
<td>Prepare for team presentation 3</td>
<td>10</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Draft interim report 3</td>
<td>11</td>
<td>ALL</td>
<td>DONE</td>
</tr>
<tr>
<td>Morpholine creation</td>
<td>10-11</td>
<td>Abhinav</td>
<td>DONE</td>
</tr>
<tr>
<td>Index solar eclipse dataset</td>
<td>10-14</td>
<td>Abhinav</td>
<td>DONE</td>
</tr>
<tr>
<td>Index Las Vegas shooting dataset</td>
<td>10-14</td>
<td>Abhinav</td>
<td>DONE</td>
</tr>
<tr>
<td>Schema changes for GeoBlacklight team</td>
<td>10-14</td>
<td>Abhinav</td>
<td>DONE</td>
</tr>
<tr>
<td>Schema changes for Visualization team</td>
<td>10-14</td>
<td>Abhinav</td>
<td>DONE</td>
</tr>
</tbody>
</table>
5.3 Milestones and Deliverables

Our milestones over time are shown below.

<table>
<thead>
<tr>
<th>Task #</th>
<th>Completion Date</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>09/05</td>
<td>Set up Solr team</td>
</tr>
<tr>
<td>2</td>
<td>09/06</td>
<td>Set up Solr on local machine</td>
</tr>
<tr>
<td>3</td>
<td>09/09</td>
<td>Set up a Cloudera VirtualBox VM</td>
</tr>
<tr>
<td>4</td>
<td>09/18</td>
<td>Go through all basic process from data importing to HBase to basic searching in Solr through VM</td>
</tr>
<tr>
<td>5</td>
<td>09/21</td>
<td>Finish tentative HBase schema design</td>
</tr>
<tr>
<td>6</td>
<td>10/17</td>
<td>Finalize working Schema</td>
</tr>
<tr>
<td>7</td>
<td>11/07</td>
<td>Working geospatial search functionality</td>
</tr>
<tr>
<td>8</td>
<td>12/6</td>
<td>Provide recommendation handler</td>
</tr>
</tbody>
</table>

As the project progresses, we will provide deliverables for the other teams.

<table>
<thead>
<tr>
<th>Task #</th>
<th>Completion Date</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>09/18</td>
<td>Send out initial HBase schema</td>
</tr>
<tr>
<td>2</td>
<td>Determined</td>
<td>Share raw data indexes with FE team</td>
</tr>
<tr>
<td></td>
<td>Unnecessary</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10/17</td>
<td>Share tweet schema file with relevant teams</td>
</tr>
<tr>
<td>4</td>
<td>10/20</td>
<td>Enable MoreLikeThis query handler for FE</td>
</tr>
<tr>
<td>5</td>
<td>11/5</td>
<td>Parse indexed data for FE</td>
</tr>
<tr>
<td>6</td>
<td>11/5</td>
<td>Provide functioning query endpoint for FE team</td>
</tr>
<tr>
<td>7</td>
<td>11/20</td>
<td>Provide indexed data quality control</td>
</tr>
<tr>
<td>8</td>
<td>12/6</td>
<td>Add inferred geolocation data to all documents with SNER locations</td>
</tr>
<tr>
<td>9</td>
<td>12/6</td>
<td>Provide &quot;ctacfa&quot; handler which uses data from the CTA and CLA teams to recommend documents</td>
</tr>
</tbody>
</table>
5.4 SNER GeoLocation

Many documents do not have location data, but do mention locations in their text. The collection management teams ran named entity recognition tools on their documents and populated several fields in HBase with people, organizations, and locations that were found in the documents. The FE team wanted to be able to use this data to supplement the location data that documents are tagged with for visualization and searching. The task of constructing bounding boxes from SNER locations fell to the SOLR team. A standard solution to the problem of finding geolocations from location names is to use an online service API which already has large amounts of geolocation data such as the Google Maps API. Unfortunately, the Google Maps API has a rate limit which makes it not conducive to processing large amounts of data like this. Instead, we downloaded geolocation data from Mapzen [15], OpenStreetMap [17], and OpenAddresses.io [16] and processed it ourselves. We first parsed all the raw data and generated a bounding box for each place name. We then created a suffix array of all these place names to facilitate fuzzy matching with the SNER locations from the original documents. Because of the large number of documents in HBase we tried to do all of the processing in Map Reduce. Unfortunately, the large suffix array of all locations we were working with required about 18 GB of memory, which exceeded the 16 GB available within each node for Map Reduce. Instead, we used Map Reduce to export the SNER locations from HBase and matched them to geolocations using parallel processing in a stand alone process which was able to use a a larger amount of memory. Although it was written in Python, the fuzzy matching using a suffix array only ended up taking at most 3 minutes to run, so we did not consider that it was worth investing any more time into re-implementing it to run on the cluster. Lastly, we used Map Reduce to import the geolocations (bounding boxes) back into HBase. Using this process of exporting locations, matching them, and re-importing we populated the solr_geom field for 324714 tweets and 4442 webpages. This was out of 324714 tweets and 4451 webpages which had SNER location data which was out of 5.9 million total tweets and 12,564 total web pages.

5.5 Results

Table 5 shows the key results of our project.

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Document Count</th>
<th>Time</th>
<th>Memory (Heap Usage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Webpage</td>
<td>12,564</td>
<td>Map: 88 sec</td>
<td>789 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reduce: 531 sec</td>
<td></td>
</tr>
<tr>
<td>Tweets</td>
<td>5,969,120</td>
<td>Map: 780 sec</td>
<td>3 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reduce: 5300 sec</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Key results of Solr team
6 User Manual

In this section, we introduce how to set up the Solr system, the Solr admin interface, and how to use Solr to query and search.

6.1 Install Solr system

In this section, we introduce how to set up the Solr system quickly.

6.1.1 Available Solr packages

The Solr system is available from the Solr website: http://www.apache.org/dyn/closer.lua/lucene/solr/7.1.0

There are three separate packages:

- solr-7.0.0.tgz for Linux/Unix/OSX systems
- solr-7.0.0.zip for Microsoft Windows systems
- solr-7.0.0-src.tgz the package Solr source code. Once extracted, the computer is ready to run Solr.

6.1.2 Start Solr

Solr includes a command line interface tool called bin/solr (Linux/MacOS) or bin\solr.cmd (Windows). This tool allows you to start and stop Solr, create cores and collections, configure authentication, and check the status of your system.

To use it to start Solr you can simply enter:

bin/solr start

If you are running Windows, you can start Solr by running bin solr.cmd instead.

bin\solr.cmd start

This will start Solr in the background, listening on port 8983.
When you start Solr in the background, the script will wait to make sure Solr starts correctly before returning to the command line prompt[2].

6.1.3 Check if Solr is running

If you’re not sure if Solr is running locally, you can use the status command[2]:

bin/solr status

This will search for running Solr instances on your computer and then gather basic information about them, such as the version and memory usage.
That’s it. Solr is running. If you need convincing, use a web browser to see the Admin Console.
http://localhost:8983/solr/
If Solr is running you will see a dashboard like in Figure 6.
If Solr is not running, your browser will complain that it cannot connect to the server.

6.2 Solr Admin Interface

Solr has a usable admin interface to make it easy for Solr administrators and programmers to view Solr configuration details, run queries, and analyze document fields in order to fine-tune a Solr
configuration, and access online documentation and other help. Accessing URL http://localhost:8983/solr/ will show the main dashboard, which is like Figure 6.

![Figure 6: Solr Admin Dashboard](image)

The left-side is a menu under the Solr logo providing the navigation through the screens of the UI. The first set of links is for system-level information and configuration, and provides access to Logging, Collection/Core Administration, and Java Properties, among other things. At the end is a pulldown, listing Solr cores configured for this instance. Click on a core selector to perform queries on indexed data.

### 6.3 Solr Query

Using Solr for indexing and querying is an important function provided by the Solr admin UI. To efficiently utilize the Solr admin interface, we need to understand the meaning of each query parameter, query syntax, and function query.
### 6.3.1 Query Parameters

#### Table 6: Core Query Parameters

<table>
<thead>
<tr>
<th>Query Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>The query event.</td>
</tr>
<tr>
<td>q.op</td>
<td>Overriding the default operator specified in SchemaXml. Possible values are &quot;AND&quot; or &quot;OR&quot;.</td>
</tr>
<tr>
<td>df</td>
<td>This parameter overrides the default field defined in SchemaXml, if provided.</td>
</tr>
<tr>
<td>qt</td>
<td>Query type. Determine which Query Handler should be used to process the request. The default is &quot;standard&quot;.</td>
</tr>
<tr>
<td>wt</td>
<td>Write type. Determine which QueryResponseWriter should be used to process the request. The default value is &quot;standard&quot; (XML).</td>
</tr>
<tr>
<td>echoHandler</td>
<td>If the echoHandler parameter is true, Solr places the name of the handle used in the response to the client for debugging purposes.</td>
</tr>
<tr>
<td>echoParams</td>
<td>The response header can include the parameters sent in the query request. This parameter controls what is contained in that section of the response header. Valid values are none, all, and explicit. The default value is explicit.</td>
</tr>
</tbody>
</table>

#### Table 7: Common Query Parameters

<table>
<thead>
<tr>
<th>Query Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>Sort. Format: sort=&lt;field name&gt;+[desc</td>
</tr>
<tr>
<td>start</td>
<td>Used to paginate results from a query. Default value is “0”.</td>
</tr>
<tr>
<td>rows</td>
<td>Used to paginate results from a query. It specifies the maximum number of documents from the complete result set to return to the client for every request. The default value is &quot;10&quot;.</td>
</tr>
<tr>
<td>fq</td>
<td>Filter query. It can be very useful for speeding up complex queries since the queries specified with fq are cached independently from the main query. For example: q=mm&amp;fq=date_time:[20161001 TO 20170914], searches for the keyword “mm” and filters the date_time to be between 20161001 and 20170914.</td>
</tr>
<tr>
<td>fl</td>
<td>Field list. The set of fields to be returned is specified as a space (or comma) separated list of field names.</td>
</tr>
<tr>
<td>debugQuery</td>
<td>If this parameter is present (regardless of its value) then additional debugging information will be included in the response.</td>
</tr>
<tr>
<td>explainOther</td>
<td>Allows clients to specify a Lucene query to identify a set of documents.</td>
</tr>
<tr>
<td>defType</td>
<td>Selects the query parser to be used to process the query. defType=lucene, defType=dismax, defType=edismax</td>
</tr>
</tbody>
</table>
3.3.2 Query Syntax

Solr supports multiple query syntaxes through its query parser plugin framework.[2] We will use the standard parser for our examples.

**Keyword matching**
Search for word "foo" in the title field.

\texttt{title:foo}

Search for phrase "foo bar" in the title field.

\texttt{title:"foo bar"}

Search for phrase "foo bar" in the title field AND the phrase "quick fox" in the body field.

\texttt{title:"foo bar" AND body:"quick fox"}

Search for either the phrase "foo bar" in the title field AND the phrase "quick fox" in the body field, or the word "fox" in the title field.

\texttt{(title:"foo bar" AND body:"quick fox") OR title:fox}

Search for word "foo" and not "bar" in the title field.

\texttt{title:foo -title:bar}

**Wildcard matching**
Search for any word that starts with "foo" in the title field.

\texttt{title:foo*}

Search for any word that starts with "foo" and ends with bar in the title field.

\texttt{title:foo*bar}

(Note that Lucene doesn’t support using a * symbol as the first character of a search.)

**Proximity matching**
Lucene supports finding words that are within a specific distance of each other.
Search for the words "foo" and "bar" within 4 words of each other.
"foo bar" 4
Note that for proximity searches, exact matches are proximity zero, and word transpositions (bar foo) are proximity 1.
A query such as "foo bar" 10000000 is an interesting alternative to foo AND bar.
While the queries are effectively equivalent with respect to the documents that are returned, the proximity query assigns a higher score to documents for which the terms foo and bar are closer together.
The trade-off is that the proximity query is slower to perform and requires more CPU.
The Solr DisMax and eDisMax query parsers can support queries for phrase proximity matches.

Range searches
Range Queries allow one to match documents whose field(s) values are between the lower and upper bound specified by the Range Query. Range Queries can be inclusive or exclusive of the upper and lower bounds. Sorting is done lexicographically. mod_date:[20020101 TO 20030101]
Solr’s built-in field types are very convenient for performing range queries on numbers without requiring padding.

Solr-specific query syntax
There are several differences between the Solr Query Parser and the standard Lucene query syntax (from the Solr wiki):
A * may be used for either or both endpoints to specify an open-ended range query.
   field:[* TO 100] finds all field values less than or equal to 100
   field:[100 TO *} finds all field values greater than or equal to 100
   field:[* TO *] matches all documents with the field
Pure negative queries (all clauses prohibited) are allowed.
   -inStock:false finds all field values where inStock is not false
   -field:[* TO *] finds all documents without a value for field
A hook into the FunctionQuery syntax. Quotes will be necessary to encapsulate the function when it includes parentheses.
   Example: _val_:myfield
   Example: _val_:"recip(rord(myfield),1,2,3)"
Nested query support for any type of query parser (via QParserPlugin). Quotes will often be necessary to encapsulate the nested query if it contains reserved characters.
   Example: _query_:{!dismax qf=myfield}how now brown cow"

6.3.3 Function Query
There are a few ways to use a FunctionQuery from Solr’s HTTP interface[19]:

1. Invoke the FunctionQParserPlugin via LocalParams syntax, e.g., q=!funclog(foo)
   Alternatively: Set func as the default query type, e.g., defType=func&q=log(foo)

2. Invoke the FunctionRangeQParserPlugin via LocalParams syntax in a filter query
   e.g., fq={!frange l=0}sub(field1,field2)
3. Use a parameter that has an explicit type of FunctionQuery, such as DisMaxQParserPlugin’s bf (boost function) parameter, or extended dismax boost parameter (multiplicative boost). NOTE: The bf parameter actually takes a list of function queries separated by whitespace and each with an optional boost. Make sure to eliminate any internal whitespace in single function queries when using bf.
Example: q=foo&bf="ord(popularity)^0.5 recip(ord(price),1,1000,1000)^0.3"

4. Embed a FunctionQuery in a regular query expressed in SolrQuerySyntax via the _val_ hook

6.4 Spatial Search

Solr supports location data which can be used for geospatial search. Using this feature we can perform the following operations:

- Index point data
- Filter search results using a bounding box, a circle or any rectangular region

Once the data is indexed as LatLonType (Solr’s field type to support spatial search) you can use the following types of filter queries (fq):

- Using Solr query parsers for geospatial search. There are 2 such query parsers:
  1. geofilt- This query parser takes the following parameters:
     - pt: location of the center in lat,lon format
     - sfield: name of the field which is indexed as LatLonType
     - d: radius of the circle in terms of kilometers
     Here is an example of a filter query using the geofilt query parser: fq={!geofilt sfield=coordinates pt=45.15,-93.85 d=5}
  2. bbox- This query parser takes the same parameters as a geofilt query parser. Here is an example of a filter query using the bbox query parser: fq={!bbox sfield=coordinates pt=45.15,-93.85 d=5}

Both query parsers take the same parameters as input but the results are different. The following two figures will illustrate the difference between them:
• Range query on location fields. Using a range query Solr lets you define a rectangular region to filter the search results. It takes coordinates of lower left and top right corner as the input. Here is an example of a range filter query on LatLonType data: fq=[54.15,-93.85 TO 56.6].

6.5 Document Recommendation

Sometimes users themselves may not know what they really want. At that time, if recommendation can be provided, user experience may be better. Our first step in Recommendation is based on document textual similarity, by using MoreLikeThis (MLT). MLT enables users to query for documents similar to a document in their result list. It does this by using terms from an original document to find similar documents in the index. To make full use of this, here are some common MoreLikeThis parameters supported by Lucene/Solr.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlt.fl</td>
<td>Specifies the Minimum Term Frequency, the frequency below which terms will be ignored in the source document.</td>
</tr>
<tr>
<td>mlt.mintf</td>
<td>Used to paginate results from a query. Default value is “0”.</td>
</tr>
<tr>
<td>mlt.mindf</td>
<td>Specifies the Minimum Document Frequency, the frequency at which words will be ignored which do not occur in at least this many documents.</td>
</tr>
<tr>
<td>mlt.maxdf</td>
<td>Specifies the Maximum Document Frequency, the frequency at which words will be ignored which occur in more than this many documents.</td>
</tr>
<tr>
<td>mlt.minwl</td>
<td>Sets the minimum word length below which words will be ignored.</td>
</tr>
<tr>
<td>mlt.maxwl</td>
<td>Sets the maximum word length above which words will be ignored.</td>
</tr>
<tr>
<td>mlt.maxqt</td>
<td>Sets the maximum number of query terms that will be included in any generated query.</td>
</tr>
<tr>
<td>mlt.maxntp</td>
<td>Sets the maximum number of tokens to parse in each example document field that is not stored with TermVector support.</td>
</tr>
<tr>
<td>mlt.boost</td>
<td>Specifies if the query will be boosted by the interesting term relevance. It can be either &quot;true&quot; or &quot;false&quot;.</td>
</tr>
<tr>
<td>mlt.qf</td>
<td>Query fields and their boosts using the same format as that used by the DisMax Query Parser. These fields must also be specified in mlt.fl.</td>
</tr>
</tbody>
</table>
Our second step is based on clustering and classification results, we made a handler for users which can recommend documents that belongs to same clustering and classification result. We named that handler "ctacfa".

6.5.1 MLT Syntax
MLT provides similar syntax to Query; it uses the mlt request handler, with several parameters. Here are some examples:
1. Search similar text related to "shoot" based on "full_text" field

```plaintext
http://localhost:8983/solr/getar-cs5604f17_shard1_replica1/mlt?q=full_text%3A%22shoot%22&wt=json&indent=true
```

2. Request interesting terms to be returned along with documents

```plaintext
http://localhost:8983/solr/getar-cs5604f17_shard1_replica1/mlt?q=full_text%3A%22shoot%22&wt=json&indent=true&mlt.interestingTerms=True
```

3. Search similar text related to document where the ID is 889161247551352832

```plaintext
http://localhost:8983/solr/getar-cs5604f17_shard1_replica1/mlt?q=id%3A%22889161247551352832%22&wt=json&indent=true
```

6.5.2 CTACFA Handler Syntax
The CTACFA handler provides similar syntax to Query. Here are some examples:
1. Search similar documents related to document which id is 889161247551352832 based on cluster and category results

```plaintext
http://localhost:8983/solr/getar-cs5604f17_shard1_replica1/ctacfa?q=id%3A%22889161247551352832%22&wt=json&indent=true
```

6.5.3 Solr Admin UI
Figures 7 and 8 give a simple example of using the document recommendation systems from the Admin UI.
First, set Request-Handler to "/mlt". In the "q" field, fill in a collection ID and execute the query. The response will contain documents that have high similarity based on text.
First, set Request-Handler to "/ctacfa". In the "q" field, fill in a collection ID and execute the query. The response will contain documents that have the same cluster and category as the current collection. More functionality can be implemented by changing parameters in the "solrconfig.xml" file.

6.5.4 Relevancy Boosting

Solr provides many ways to modify the relevancy calculation, allowing users to modify the weight applied to fields and terms:

Query Time Field Boosting
Boost field at query time. Here is an example, when title is more important than description

Query Time Per-term Boosting
Boost terms at query time. Here is an example, when users are querying solar eclipse, and eclipse is more important than solar
7 Developer Manual

7.1 Solr

7.1.1 Architecture

Apache Solr is a scalable, ready-to-deploy enterprise search engine that’s optimized to search large volumes of text-centric data and return results sorted by relevance. The search engine is built over the Apache Lucene Java library, which is optimized for searching.

Figure 9 shows the major building blocks of Apache Solr:

1. Request Handlers - The request handlers receive and process requests from the unified request dispatcher. The requests can be query requests or index update requests.

2. Search Components - A search component can be registered as a Search Handler within Solr. Solr makes use of these components to provide intelligent suggestions and responses to its clients.

3. Distributed Search - Solr provides distributed search through the process of sharding. Sharding partitions the index and associates each partition with a shard. Query requests made are distributed to all Solr cores holding shards, effectively parallelizing the search over the index. Results from the shards are then aggregated together and returned to the client.

4. The formula for determining theoretical query speed after adding an additional partition, given the same number of documents, is:

\[
(\text{Query Speed on } N+1 \text{ indexes}) = \text{Aggregation Overhead} + \frac{(\text{Query Speed on } N \text{ indexes})}{(N+1)}
\]
5. Query Parser - The Apache Solr query parser parses the queries that are passed to Solr and verifies the queries for syntactic errors. The parser then translates queries to a format which Lucene understands.

6. Response Writers - The response writer is a component within Apache Solr which generates formatted output. The formats generated include, but are not limited to, JSON, XML, and CSV. Every format has its associated response writer defined within Solr.

7. Analysis/Tokenizer - Apache Solr analyzes the query text content and generates a token stream. The tokenizer then breaks the token stream generated by the analyzer into tokens for Lucene to process.

8. Update Request Processor - When an update request is sent to Apache Solr, the request is run through a set of plugins (signature, logging, indexing), collectively known as the update request processor. This processor is responsible for modifications such as dropping or adding fields.

Solr enables interaction with client applications and client management systems through the following steps:

1. Define the schema for each self-contained document unit. The schema file (schema.xml) formally defines the structure of all fields within the documents which will be indexed.

2. Define the solrconfig.xml and solr.xml files. These two files determine the Solr runtime configuration for every collection. The solrconfig.xml file is additionally used by the Lily Indexer for distributed search and replication.

3. Deploy Solr based on the three XML files defined in Steps 1 and 2.

4. Add documents to be searched.

5. Expose search functionality to a user application.

7.1.2 Installation

For this project we are using CDH [4] as our distribution platform including Hadoop. According to Cloudera’s website [4], CDH is Cloudera’s 100% open source platform distribution, including Apache Hadoop and built specifically to meet enterprise demands. CDH delivers everything you need for enterprise use right out of the box. One such service is Cloudera Search. Cloudera Search is powered by Apache Solr. To enable Cloudera Search, it should be added as a service in automated installation by the Cloudera Manager as shown in the screenshot.
In the Cloudera VM, Solr features only one node where each collection typically consists of one shard (one core) as shown in the screenshot.

7.1.3 Tutorial

To manage Solr we use ‘solrctl’. It is a utility to manage Solr Cloud deployments. It helps in managing SolrCloud collections, collection directories, and individual cores.

- Starting the Solr service

  `service solr-server start`

  On running the above command a user should see the line as shown in the screenshot below, indicating that the Solr server has started successfully as a daemon.

- Stopping the Solr service

  `service solr-server stop`

  On running the above command, a user should see a line as shown in the screenshot below, indicating that the Solr server has stopped successfully.

- Creating Solr collection

  Creating a Solr collection from scratch is a three step process as described below:

    `solrctl instancedir --generate /HOME/getar-cs5604f17-eclipse-collection`
This command tells Solr to generate a template of the instance directory. It is stored in the designated path in the local file system and contains a configuration inside the ./conf folder. One important file present in the ./conf folder is the schema.xml which defines the schema of the Solr document.

```
$ solrctl instance-dir --create datatest\_collection $HOME/getar--cs5604f17--eclipse--collection
```

This command pushes a copy of the instance directory from the local file system to SolrCloud.

```
$ solrctl collection --create getar--cs5604f17--eclipse--collection
```

This command creates a new collection.

- **Indexing data for Solr**

  Indexing data for Solr is done using the Lily indexer. It supports both batch and near real time indexing. For more information on Lily please refer to Section 7.3.1.

- **Querying Solr**

  In this section we learn about queries in Solr and discuss various ways in which a user can execute their queries. Solr provides an excellent API which can be used to perform CRUD (create, read, update, delete) on documents. There are various tools which can use these APIs to provide an interface for search. In this section we will discuss two of them. The first one is the web interface built into the ‘Search Service’ of Cloudera CDH [4]. This is a great interface which hides all the complex parts from the user like building the query and parsing the results. It provides all the options available in the Solr query in the form of text boxes, dropdowns, and checkboxes, which makes it more usable. The second tool which we will be discussing is called ‘cURL’. cURL [18] is a command line tool which is used for getting and sending files using the URL format. Unlike the web interface provided by Cloudera’s CDH, cURL is for people who prefer shell commands over GUI. cURL is a good tool for working with documents with a small number of fields. When the number of fields increases, a command line tool to send XML can be quite cumbersome. Now we will jump into the practical aspects of querying Solr and run some queries using both of the tools. For the purpose of this tutorial we will be referring to the Solr setup described in the example found at https://github.com/cs5604solr/dataset_for_test. Our Solr document schema contains the following fields:

```
<field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />
<field name="text" type="text_general" indexed="true" stored="true" multiValued="true"/>
<field name="source_s" type="string" indexed="true" stored="true"/>
<field name="user_screen_name_s" type="string" indexed="true" stored="true"/>
<field name="created_time_dt" type="date" indexed="true" stored="true"/>
<field name="lang" type="string" indexed="true" stored="true"/>
<field name="_version_" type="long" indexed="true" stored="true"/>
```

We will cover the following types of queries:

1. Raw query, i.e., querying using the ‘q’ parameter only
2. Using the filter query (fq) parameter
3. Using the ‘sort’ parameter
4. Fetching a range of documents. This is useful for pagination.
5. Fetching only required fields from the document.
6. Specifying the type (JSON, XML, etc.) of response
We have indexed over 6 million tweets in our cluster under the collection name `getar-cs5604f17-solar-eclipse_shard1_replica1`. Now we will provide examples for types of queries described above using both the tools along with the snapshot.

Figure 10: Get a list of all the documents where the screen name ends with ‘a’
Figure 11: Get a list of all the documents where the screen name ends with ‘a’ and language is English

```
Request-Handler (qt)
/select

    common

    q
    *

    fq
    user_screen_name:+a AND lang:en

    sort

    start, rows
    0, 10

    wt
    json

Raw Query Parameters
key1=val1&key2=val2

    wt
    json

    indent

    debugQuery

    dismax
    edismax
    hl
    facet
    spatial
    spellcheck

http://localhost:9983/solr/getall-cs5604f17-eclipse-collection_shard1_replica1/select?q-

{
    "responseHeader": {
        "status": 0,
        "QTime": 81,
        "params": {
            "indent": "true",
            "q": "*:*",
            "fq": "1588469926445",
            "wt": "json",
            "fq": "user_screen_name:+a AND lang:en"
        }
    },
    "response": {
        "numFound": 27413,
        "start": 0,
        "docs": [
            {
                "user_statuses_count": 1154,
                "retweeted": "false",
                "user_screen_name": "adldeva",
                "place_name": "",
                "user_location": "Mekah dan Old Trafford",
                "user_verified": "false",
                "possibly_sensitive": "",
                "place_country_code": "",
                "in_reply_to_status_id_str": "",
                "user_description": "",
                "in_reply_to_user_id_str": "",
                "user_created_at": "Thu Dec 01 00:30:44 +0000 2011",
                "user_protected": "false",
                "in_reply_to_status_id": "",
                "user_friends_count": 2265,
                "place_place_type": "",
                "user_time_zone": "",
                "place_id_str": "",
                "user_name": "Adil"
            }
        ]
    }
}
```
Figure 12: Get a list of all the documents whose language is English. The documents should be sorted by `created_time_dt`.

```
#39 http://localhost:9883/solr/getter-cs5404f17-eclipse-collection_shard1 Replica1/select?q=%3A%26q%3Dlang:en&sort=created_at+asc

{
  "responseHeader": {
    "status": 0,
    "QTime": 414,
    "params": {
      "sort": "created_at asc",
      "indent": "true",
      "q": "*:*",
      "wt": "JSON",
      "fq": "lang:en"
    }
  },
  "response": {
    "numFound": 551382,
    "start": 0,
    "docs": [
      {
        "user_statuses_count": 14,
        "retweeted": "False",
        "user_screen_name": "BOEY docs",
        "place_name": "",
        "user_location": "Mission Viejo, CA",
        "user_verified": "False",
        "possibly_sensitive": "False",
        "place_country_code": "",
        "in_reply_to_status_id_str": "",
        "user_description": "",
        "in_reply_to_user_id": "",
        "user_created_at": "Mon Apr 18 18:41:48 +0000 2017",
        "user_protected": "False",
        "in_reply_to_status_id": "",
        "user_friends_count": 89,
        "place_place_type": "",
        "user_time_zone": "",
        "place_id_str": ""
      }
    ]
  }
}
```
Figure 13: Get 2 documents where the language is English

```json
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "indent": "true",
      "q": "*:*",
      "_": "1588479327746",
      "wt": "json",
      "fq": "lang:en",
      "rows": 2
    }
  },
  "response": {
    "numFound": 551382,
    "start": 0,
    "docs": [
      {
        "user_status_count": 47255,
        "retweeted": "false",
        "user_screen_name": "shereenstrachan",
        "place_name": "",
        "user_location": "Edinburgh, Scotland",
        "user_verified": "false",
        "possibly_sensitive": "",
        "place_country_code": "",
        "in_reply_to_status_id_str": "",
        "user_description": "21, chemistry student. I enjoy reading, gaming and science.",
        "in_reply_to_user_id_str": "",
        "user_created_at": "Mon Mar 30 18:22:02 +0000 2009",
        "user_protected": "false",
        "in_reply_to_status_id": "",
        "user_friends_count": 1086,
        "place_place_type": "",
        "user_time_zone": "Europe/London",
        "place_id_str": "",
        "user_name": "Shereen Strachan",
        "user_mentions_id_str": "1924364738",
        "user_lang": "en"
      }
    ]
  }
}
```
Figure 14: Fetch documents where the language is English. Response should only contain screen names

```json
{
  "responseHeader": {
    "status": 0,
    "QTime": 2,
    "params": {
      "fl": "user_screen_name",
      "indent": "true",
      "q": "**:",
      "_": "1500470675151",
      "wt": "json",
      "fq": "lang:en",
      "rows": "10"
    }
  },
  "response": {
    "numFound": 551382,
    "start": 0,
    "docs": [
      {
        "user_screen_name": "shereenstrachan"
      },
      {
        "user_screen_name": "SCEMD"
      },
      {
        "user_screen_name": "emilycoininabox"
      },
      {
        "user_screen_name": "walkerbait1982"
      },
      {
        "user_screen_name": "yvonne41277"
      },
      {
        "user_screen_name": "chba_edinmont"
      }
    ]
  }
}
```
Figure 15: Fetch all the documents where language is English. Use XML for the response type.

```
Request-Handler (q)
<table>
<thead>
<tr>
<th>select</th>
</tr>
</thead>
<tbody>
<tr>
<td>- common</td>
</tr>
<tr>
<td>q</td>
</tr>
<tr>
<td>&quot;*&quot;</td>
</tr>
</tbody>
</table>

fq
lang:en

sort

start: rows
0 | 10

wt

xml

`
7.2 HBase

7.2.1 Architecture

HBase [1] is a non-relational database maintained by the Apache Software Foundation. It is modeled after Google’s BigTable [10] database and is intended for use cases where the data being stored is at such a scale that it would require or benefit from being stored in a distributed manner. HBase differs from BigTable in that it uses the Hadoop File System to store its files. Because it is a NoSQL database, Apache HBase uses different schema design strategies than traditional relational databases. Like Google’s BigTable database, a table in HBase can be thought of as a multidimensional map.

The first key within a table is the row identifier. Each row has multiple columns, each with a unique column identifier for a key. These column identifiers can be defined on the fly as data is being put into the table and rows do not have to have the same columns. One restriction that is placed on column identifiers is that each column has to be part of a column family that was defined at table creation. This column family is expressed as a prefix to the column identifier.

For example, the columns tweets:author and tweets:body are both part of the column family tweets. HBase stores a timestamp/version with each cell, allowing applications to retrieve any version of the data value for a cell. Physically, HBase guarantees that each row is stored in a single HDFS file, allowing for atomic operations within a row. The files themselves can be distributed among nodes in a cluster which are called region servers. These region servers are managed by Apache ZooKeeper, a tool for managing distributed applications.

HBase can run in standalone mode or distributed mode. In standalone mode, all region servers daemons are running in the Java Virtual Machine running on a single server. In distributed mode, the region servers daemons are running on separate physical servers.

![Figure 16: The Architecture of HBase](image)
7.2.2 Tutorial

Figure 17: Get the small Twitter collection for testing

Upload the CSV file into HDFS
Use command “hadoop fs -put”. On success, you should be able to find the file you uploaded to HDFS using "hadoop fs -ls". We use the same command for web page data.

```
$hadoop fs -put /home/cloudera/cs_5604/Dataset_for_Test-master/dataset_test.csv #(put the 'data_test.csv' to HDFS)
$hadoop fs -ls #(list all we have in HDFS)
```

Figure 18: HDFS ls example

Check service status

```
$service -status-all
```
Make sure all necessary components are running, otherwise, use the command:

```
$sudo service SERVICE-NAME start to start any “not running” service
```
Create HBase table for testing Twitter file.

```bash
$ hbase shell
$ create 'test','raw'
```

Figure 20: HBase create table example

Put data into HBase

```bash
$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=','
$ -Dimporttsv.columns="HBASE_ROW_KEY,raw:c1,raw:c2,raw:c3,raw:c4,raw:c5,raw:c6,
raw:c7,raw:c8,raw:c9,raw:c10,raw:c11,raw:c12" test dataset_test.csv
```
7.3 Lily HBase Indexer

7.3.1 Overview

What it is?
The HBase Lily Indexer [5] provides indexing (via Solr) for content stored in HBase. It provides
a flexible and extensible way of defining indexing rules, and is designed to scale for large datasets.
Indexing is performed asynchronously, so it does not impact write throughput on HBase. SolrCloud
is used for storing the actual index to ensure scalability of the indexing.

How it Works
The HBase Indexer works by acting as an HBase replication sink. As updates are written to HBase
region servers, they are “replicated” asynchronously to the HBase Indexer processes. The indexer
analyzes incoming HBase mutation events, and where applicable it creates Solr documents and
pushes them to SolrCloud servers.
The indexed documents in Solr contain enough information to uniquely identify the HBase row that
they are based on, allowing you to use Solr to search for content that is stored in HBase. HBase
replication is based on reading the HBase log files, which are the precise source of truth for what
is stored in HBase: there are no missing or extra events. In various cases, the log also contains all
the information needed to index, so that no expensive random-read on HBase is necessary (see the
read-row attribute in the Indexer Configuration https://github.com/NGDATA/hbase-indexer/
wiki/Indexer-configuration).

HBase replication delivers (small) batches of events. HBase-indexer exploits this by avoiding
double-indexing of the same row if it would have been updated twice in a short time frame, and
will batch/buffer the updates towards Solr, which gives important performance gains. The updates
are applied to Solr before confirming the processing of the events to HBase, so that no event loss
is possible.

Horizontal scalability
All information about indexers is stored in ZooKeeper. New indexer hosts can always be added
to a cluster, in the same way that HBase region servers can be added to to an HBase cluster. All
indexing work for a single configured indexer is shared over all machines in the cluster. In this
way, adding additional indexer nodes allows horizontal scaling.

Automatic failure handling
The HBase replication system upon which the HBase Indexer is based is designed to handle
hardware failures. Because the HBase Indexer is based on this system, it also benefits from the
same ability to handle failures.
In general, indexing nodes going down or Solr nodes going down will not result in any lost data in
the HBase Indexer.

7.3.2 Live Mode Indexing

Create Lily Indexer Configuration File
The configuration file for the Lily Indexer is “Morphline-hbase-mapper.xml”. The configuration
file controls the mapping between HBase columns and Solr indexes. Typically, there is one Lily
HBase Indexer configuration for each HBase table, but there can be as many Lily HBase Indexer
configurations as there are tables, column families, and corresponding collections in Search. Each Lily HBase Indexer configuration is defined in an XML file, such as morphline-hbase-mapper.xml.

Here is a sample Morphline-hbase-mapper.xml [12]:

```xml
<?xml version="1.0"?>
<indexer table="record"
  mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">
  <!-- The relative or absolute path on the local file system to the
   morphline configuration file. -->
  <!-- Use relative path "morphlines.conf" for morphlines managed by
   Cloudera Manager -->
  <param name="morphlineFile" value="/etc/hbase-solr/conf/morphlines.conf"/>
  <!-- The optional morphlineId identifies a morphline if there are multiple
   morphlines in morphlines.conf -->
  <param name="morphlineId" value="morphline1"/>
</indexer>
```

In our VM, we create the configuration file under our home directory:

```
/home/cloudera/morphline-hbase-mapper.xml
```

We keep the content of morphline-hbase-mapper.xml as it is in the sample.

Then, we need to create morphlines.conf file as mentioned in the morphline-hbase-mapper.xml. By default, the morphlines.conf should be created in the path:

```
/etc/hbase-solr/conf/morphlines.conf
```

Here is a sample morphlines.conf:

```json
morphlines : [
  { id : morphline1
    importCommands : ["org.kitesdk.morphline.***", "com.ngdata.***"]
    commands : [
      { extractHBaseCells {
        mappings : |
          { inputColumn : "data:*" outputField : "data" type : string source : value
        }
        { logTrace { format : "output record: {}", args : [@{[]}] } }
      }
    ]
  }
]
```

For our future test, we need to modify morphlines.conf to fit our input data format. Here is the version we used:

```json
morphlines : [
  { id : morphline1
    importCommands : ["org.kitesdk.morphline.***", "com.ngdata.***"]
    commands : [
      { extractHBaseCells {
        mappings : |
          { inputColumn : "raw:c1" outputField : "text" type : string source : value
        }
          { inputColumn : "raw:c3" outputField : "screen_name_s" type : string source : value
        }
      }
      { logTrace { format : "output record: {}", args : [@{[]}] } }
    ]
  }
]
```

47
Running HBaseMapReduceIndexer Tool

Here we are going to index our HBase table using MapReduce in live mode. First, we create the log4j.properties file to set the Java logging properties:

/home/cloudera/cs_5604/lily_index/logs/log4j.properties

Here is a sample log4j.properties file:

# Root logger option
log4j.rootLogger=INFO, stdout

# Direct log messages to stdout
log4j.appenders.stdout=org.apache.log4j.ConsoleAppender
log4j.appenders.stdout.Target=System.out
log4j.appenders.stdout.layout=org.apache.log4j.PatternLayout
log4j.appenders.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%p] %c{1}:%L - %m%n

Then we use the following command to generate indexes for our table:

hadoop -config /etc/hadoop/conf j ar/usr/lib/hbase-solr/tools/hbase-indexer-mr--job.jar
--conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts= -Xmx500m'
--hbase-indexer-file /home/cloudera/morphline-hbase-mapper.xml --zk-host 127.0.0.1:solr --collection hbase-collection --go-live
--log4j /home/cloudera/cs_5604/lily/_index/logs/log4j.properties

When the job finishes, the generated index should be automatically imported to Solr. (In Batch mode we need to import index files manually.)

Note: If a Java heap exception is thrown in the process, you need to adjust mapred-site.xml to increase the Java heap for doing MapReduce. The file is located at

/etc/hadoop/conf/mapred-site.xml.

In this file, change values for the following two properties mapreduce.map.java.opts and mapreduce.reduce.java.opts. We also increased our heap size to Xmx1024M.

7.3.3 Batch Mode Indexing

Create Lily Indexer Configuration File

Configuration files for batch mode are the same as Live Mode. Refer to Section 7.3.2 Live Mode Indexing to create relative files: morphline-hbase-mapper.xml and morphlines.conf [12]

Run HBaseMapReduceIndexer Tool

Here, we are going to index our HBase table using MapReduce in batch mode. We can use the same log4j.properties file as the one in the Live Mode Indexing part. Refer to the previous section to create one if you do not have it created.

We use the following command to generate indexes and output to Solr in batch mode:

hadoop -config /etc/hadoop/conf j ar
/usr/lib/hbase-solr/tools/hbase-indexer-mr--job.jar
--conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts= -Xmx500m'
--hbase-indexer-file /home/cloudera/morphline-hbase-mapper.xml --zk-host 127.0.0.1:solr --collection hbase-collection --verbose
--output-dir hdfs://quickstart.cloudera/user/cloudera/cs5604f17-test-index
--overwrite-output-dir --shard 1

When the job finishes, index files should be generated in HDFS. You can use the following command to check it: hadoop fs -ls /solr/hbase-collection.core_node1/data/index

Note: If a Java heap exception is thrown in the process, you need to adjust mapred-site.xml to increase the Java heap for doing MapReduce. The file is located at

/etc/hadoop/conf/mapred-site.xml. In this file, change values for the following two properties
mapreduce.map.java.opts and mapreduce.reduce.java.opts. We increased our heap size to Xmx1024M.

**Move Index Files to OS from HDFS**
Move the index files from HDFS to the local filesystem to transfer index files to any other machine for reuse.
Here, we use the following command to get index files from HDFS:
```
hadoop fs -get /solr/hbase-collection.core_node1/data/index index-export
```

**Remove previous index files from Solr**
We use the following commands to remove all our collection index files under Solr:
```
sudo -u hdfs hadoop fs -rm -r -skipTrash /solr/hbase-collection.core_node1/data/index
sudo -u hdfs hadoop fs -rm -r -skipTrash /solr/hbase-collection.core_node1/data/tlog
```

**Import offline index files to Solr**
We use the following commands to import our offline index files to Solr:
```
sudo -u solr hadoop fs -put index /solr/hbase-collection/core_node1/data/
sudo service solr-server restart
```

**Note:** Remember to restart the Solr service to make the changes effective.
**Note:** If a Java heap size error is thrown in the process, you need to modify the `solrconfig.xml` file and set the `solr.hdfs.blockcache.enabled` property to false. Then restart the Solr service.

### 7.3.4 Incremental Indexing Using NRT Indexer

**Enable HBase Column Family Replication**
To enable HBase column family replication, you need to define the replication property in all tables and set it for all column families that need to be indexed. This operation can be done in the HBase shell.

To set the property for our current test table, we use the following commands in the HBase shell:
```
disable 'test'
Alter 'test',{NAME=> 'raw', REPLICATION_SCOPE => 1}
```
```
enable 'test'
```

**Register Lily HBase Indexer**
First, make sure the `morphline-hbase-mapper.xml` is ready (refer to the Live Mode Index section if it is not).

Then, we use following command to register the Lily HBase indexer:
```
hbase-indexer add-indexer
   --name Indexer\_NRTIndexer
   --indexer-conf /home/cloudera/morphline-hbase-mapper.xml
   --connection=param solr zk=localhost:2181/solr
   --connection=param solr collection=hbase\_collection
   --zookeeper localhost:2181
```

To verify that the indexer is successfully created:
```
hbase-indexer list-indexers
```
First, we need to configure the Lily HBase NRT Indexer Service with the ZooKeeper ensemble for the HBase cluster. Add the following property to `/etc/hbase-solr/conf/hbase-indexer-site.xml`. Here is the sample `hbase-indexer-site.xml` file:

```xml
<?xml version="1.0"?>
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>localhost</value>
  </property>
  <property>
    <name>hbaseindexer.zookeeper.connectstring</name>
    <value>localhost:2181</value>
  </property>
</configuration>
```

Start Lily HBase NRT Indexer Service
First, we restart `hbase-solr-indexer` to make previous changes effective:

```bash
sudo service hbase-solr-indexer restart
```

**Note:** If a Java heap error is thrown, go to the configuration page for the Lily HBase indexer (Key-value-store indexer). Change the Java heap size to at least 512MB. Then stop the `hbase-solr-indexer` service and restart the Key-Value store indexer in the Cloudera Manager page. See [8] and [9] for more information.

### 7.4 Recommender Systems

#### 7.4.1 Architecture

An overview of a recommender system is shown in figure 21. A query, which can include various user and contextual features, is generated when a user visits the website.

The recommender system returns a list of items (also referred to as impressions) on which users can perform certain actions such as clicks or rates.

These user actions, along with the queries and impressions, are recorded in the logs as the training data for the learner.

![Figure 21: Recommendation System Architecture](image)
7.4.2 Configuration for Handler

Based on text similarity, we configured the MoreLikeThis handler

Based on clustering and classification results, we configured the CTACFA handler

After changing solrconfig.xml, we need to update the instanceDir and reload the collection.

7.4.3 Tutorial for Personalized Recommendation

Based on those steps from the design part, we implemented the program for a user-user collaborative filtering algorithm; here is the core part of this program:
Here is an example: when we run this code, and want to know which top 10 items we should recommend to the user whose ID is 320, based on score from high to low, here are the top 10 result:

```
recommndations for user 320:
858 (Godfather, The (1972)): 4.562
2360 (Celebration, The (The Festen) (1998)): 4.556
318 (Shawshank Redemption, The (The (1994)): 4.556
8638 (Before Sunset (2004)): 4.512
7371 (Dogville (2003)): 4.511
922 (Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)): 4.503
1217 (Ran (1985)): 4.497
44555 (Lives of Others, The (Das leben der Anderen) (2006)): 4.491
2859 (Stop Making Sense (1984)): 4.486
108 (Reservoir Dogs (1992)): 4.479
```

For this a Recommendation.jar file is included. Simply run
	$-java Recommendation.jar userlog.Recommandation. This command will generate recommendations in the sqlite server provided by the front end. The Solr queries for these have been implemented, but are not working due to permission issues.
8 Cluster Developer Manual

8.1 Access Cluster Solr Web UI

In order to access the Solr web interface on the cluster, we need to set up port forwarding to forward Solr packets to our local machine. There are a variety of ways to do this. Here we recommend the following options: Secure Pipes for Mac user or Terminal access through SSH tunneling.

8.1.1 Secure Pipes

For Mac users, first download the Secure Pipes installation package from the link below:

https://www.opoet.com/pyro/

After installation, in preference add and select New Local Forward

In the configuration panel, set all properties as shown in the screen shot except your own SSH user name and password (same as the one to SSH into the cluster):

8.1.2 SSH Tunneling

All platform users can use SSH tunneling to port forward packets to your own local machine through the `-L` option:

```
ssh -L 9983:solr2.dlr1:8983 user_name@hadoop.dlib.vt.edu -N
```
8.2 Command Examples on Cluster

Notice that many commands share the same format as ones in the VM section, but there are differences due to the different architecture of the cluster. Here we provide example commands for easy access and to clear up any ambiguities.

8.2.1 Solr Commands

Create Solr instance directory

```
Solrctl instanceDir --generate /your/path/your-collection-name
```

Create a Solr collection

```
Solrctl collection --create your-collection-name
```

Upload/Update Solr configuration files to a collection

```
Solrctl instanceDir --create your-collection-name /your/path/to/collection-conf
Solrctl instanceDir --update your-collection-name /your/path/to/collection-conf
```

Delete a Solr collection

```
Solrctl collection --delete your-collection-name
```

Reload a Solr collection

```
Solrctl collection --reload your-collection-name
```

Remove all Solr collection files on HDFS

```
hdfs dfs -rm -r /solr/your-collection-name/core_node*
```

8.2.2 Lily Indexer Commands

Batch-mode Indexing

```
Hadoop --config /etc/hadoop/conf jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-1.5-cdh5.12.0-job.jar --conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx3000m' -D 'hbase-indexer-file=/your/path/to/morphline-hbase-mapper.xml -zk-host node00.dlr1:2181, node01.dlr1:2181, node02.dlr1:2181, node03.dlr1:2181, node04.dlr1:2181/solr' -D 'log4j.properties=collection=your-solr-collection --verbose --go-live --output-dir hdfs://your/collection/path --overwrite-output-dir --shards 1
```

8.2.3 Morphline.conf Reference

Link for Morphlines Reference Guide:

http://kitesdk.org/docs/1.1.0/morphlines/morphlines-reference-guide.html

A sample of basic morphlines.conf:

```json
SOLR_LOCATOR : {
  collection : getar-cs5604f17-solar-eclipse
  zkHost : "solr2.dlr1:2181, node2.dlr2:2181, node3.dlr1:2181, node1.dlr1:2181, node4.dlr1:2181/solr"
}
morphlines: [{
  id: getar-cs5607f17-eclipse
  importCommands : ["org.kitesdk.morphline.***", "com.ngdata.***"]
  commands : [{
    extractHBaseCells {
      mappings : [
        # all input/output mapping goes here
        { inputColumn : "tweet.in_reply_to_status_id" outputField : "in_reply_to_status_id" type : string source : value }
      ]
    }
  }]
}
```
Sanitize unknown Solr fields

```java
sanitizeUnknownSolrFields {
    # Location from which to fetch Solr schema
    solrLocator : ${SOLR_LOCATOR}
}

```

Timestamp conversion

```java
convertTimestamp {
    field : created_at
    inputFormats : ["unixTimeInSeconds"]
    inputTimezone : UTC
    outputFormat : "yyyy-MM-dd'T'HH:mm:ss'Z'"
    outputTimezone : EST
}

```

custom java codes

```java
java {
    imports : "import java.util.*;"
    code : "Your custom java codes go here"
}
```

if else condition

```java
if {
    conditions : []
    then : []
} {some other commands}
```

Custom Java sample for fixing data:

```
# Remove tags around improper data in HBase
# Sample input:
#"&lt;a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>"
# Sample output:
#"TweetDeck"

java {
    imports : "import java.util.*;"
    code : "
    // records could be parsed into Java List<String>
    List<String> dc_source = record.get("dc_source_sm");
    // Use list iterator to get content
    ListIterator<String> iterator = dc_source.listIterator();
    while(iterator.hasNext()){
        String next = iterator.next();
        String new_str = new String();
        if(next.matches("<a href=".*")
            // modify string by regex
            new_str = next.replaceAll("<a href=".*");
        // set new value to record
        iterator.set(new_str);
    }
    // return the custom process
    return child.process(record);
    
    "
    }"}
```

If Condition sample for dropping bad web page records:

```
# drop web-page records the contains bad status codes
if {
    conditions : [
} then : [
    logTrace { format : "Ignoring record because it bad web-page status code", args : ["{}"] }
    { dropRecord () }
]}
```

55
References


### Table 9: HBase Schema

<table>
<thead>
<tr>
<th>Column Family</th>
<th>Column Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Family</td>
<td>Column-name</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>metadata</td>
<td>doc-type</td>
<td>type of the document</td>
<td>tweet/webpage</td>
</tr>
<tr>
<td>metadata</td>
<td>collection-id</td>
<td>number of the collection</td>
<td>651</td>
</tr>
<tr>
<td>metadata</td>
<td>collection-name</td>
<td>name of the collection</td>
<td>electricity</td>
</tr>
<tr>
<td>metadata</td>
<td>dummy-data</td>
<td>designates that this is dummy data</td>
<td>true/false</td>
</tr>
<tr>
<td>tweet</td>
<td>tweet-id</td>
<td>tweet’s unique identifier</td>
<td>299755872668758016</td>
</tr>
<tr>
<td>tweet</td>
<td>archive-source</td>
<td>twitter API’s type</td>
<td>twitter-search, twitter-stream</td>
</tr>
<tr>
<td>tweet</td>
<td>source</td>
<td>platform’s type</td>
<td>Android, iPhone</td>
</tr>
<tr>
<td>tweet</td>
<td>text</td>
<td>tweet’s original text</td>
<td>&quot;I can’t believe it was a Virginia Tech student that posted that yik yak today. Just so disappointing!&quot;</td>
</tr>
<tr>
<td>tweet</td>
<td>screen-name</td>
<td>tweeter’s username</td>
<td>FiremanDave32</td>
</tr>
<tr>
<td>tweet</td>
<td>user-id</td>
<td>user’s unique identifier</td>
<td>385665827</td>
</tr>
<tr>
<td>tweet</td>
<td>tweet-deleted</td>
<td>flag indicating that this tweet has been deleted</td>
<td>true/false</td>
</tr>
<tr>
<td>tweet</td>
<td>user-deleted</td>
<td>flag indicating that this user has been deleted</td>
<td>true/false</td>
</tr>
<tr>
<td>tweet</td>
<td>contributor-enabled</td>
<td>flag indicating that this user has contributor enabled</td>
<td>true/false</td>
</tr>
<tr>
<td>tweet</td>
<td>created-timestamp</td>
<td>created-time (UNIX time)</td>
<td>1428951621</td>
</tr>
<tr>
<td>tweet</td>
<td>created-time</td>
<td>created-time (readable)</td>
<td>Mon Apr 13 19:00:21 +0000 2015</td>
</tr>
<tr>
<td>tweet</td>
<td>language</td>
<td>tweet’s main language</td>
<td>en</td>
</tr>
<tr>
<td>tweet</td>
<td>geo-type</td>
<td>point / polygon</td>
<td>point</td>
</tr>
<tr>
<td>tweet</td>
<td>geo-0</td>
<td>latitude</td>
<td>43.02099179</td>
</tr>
<tr>
<td>tweet</td>
<td>geo-1</td>
<td>longitude</td>
<td>-80.44612986</td>
</tr>
<tr>
<td>tweet</td>
<td>url</td>
<td>original URL in tweet</td>
<td><code>&lt;a href=&quot;http://twittercounter.com&quot;&gt;The Visitor Widget&lt;/a&gt;</code></td>
</tr>
<tr>
<td>tweet</td>
<td>to-user-id</td>
<td>unique identifier of the reply-to user</td>
<td>0</td>
</tr>
<tr>
<td>tweet</td>
<td>profile-img-url</td>
<td>image URL from the user profile</td>
<td><a href="http://a0.twimg.com/profile_images/3149217853/0026816c03013356b569a87/5af351fb_normal.jpeg">http://a0.twimg.com/profile_images/3149217853/0026816c03013356b569a87/5af351fb_normal.jpeg</a></td>
</tr>
<tr>
<td>clean-tweet</td>
<td>clean-text-solr</td>
<td>[clean text for Solr and FE]</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. no pornographic URLs, hashtags.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. inappropriate plaintext, e.g. fuck, redacted as f***</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clean-tweet</th>
<th>clean-text-cla</th>
<th>[clean text for CLA]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. no porngraphic hashtags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. regular hashtags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. inappropriate plaintext, e.g. fuck, redacted as f***</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. all URLs removed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. stop words removed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. text lemmatized</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. remove # or @ symbol from mentions or hashtags&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clean-tweet</th>
<th>clean-text-cta</th>
<th>[clean text for CTA]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. no porngraphic hashtags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. regular hashtags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. inappropriate plaintext, e.g. fuck, redacted as f***</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. all URLs removed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. stop words removed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. text lemmatized</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. remove @ symbol only from mentions, but keep #&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clean-tweet</th>
<th>NER</th>
<th>Name entity recognition. Tag format is desired.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I miss &lt;em class=&quot;LOCATION&quot;&gt;California&lt;/em&gt; at &lt;em class=&quot;TIME&quot;&gt;mid-night&lt;/em&gt;!</td>
<td></td>
</tr>
<tr>
<td>clean-tweet</td>
<td>POS</td>
<td>Part of speech. Tag format is desired. No need to tag stop words.</td>
</tr>
<tr>
<td>-------------</td>
<td>-----</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I &lt;em class=&quot;V&quot;&gt; found &lt;/em&gt; a &lt;em class=&quot;NN&quot;&gt; gun &lt;/em&gt; in &lt;em class=&quot;NN&quot;&gt; California &lt;/em&gt; at &lt;em class=&quot;NN&quot;&gt; midnight &lt;/em&gt;!</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>rt</td>
<td>tag for the retweets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0/1</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>geo-location</td>
<td>readable location from Google API</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Blacksburg, Virginia</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>spatial-coord</td>
<td>Point coordinates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(34.3, -118.27)</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>spatial-bounding</td>
<td>Spatial bounding box for S,W,N,E (if not a point)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(34.0, -118.2, 36.5, -117.4)</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>solr_geom</td>
<td>Derived from spatial-bounding. This field is indexed as a Solr spatial (RPT) field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENVELOPE(34.0, -118.2, 36.5, -117.4)</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>geom-type</td>
<td>Point /polygon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>point</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>hashtags</td>
<td>tweet’s hashtags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#hurricane</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>mentions</td>
<td>tweet’s mentions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@VT</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>long-url</td>
<td>extended URL</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>dates</td>
<td>timestamps of dates in text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Apr 13 19:00:21 +0000 2015,Mon Apr 14 14:00:53 +0000 2015</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>sner-people</td>
<td>extract names from each tweet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Obama; Jimmy</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>sner-organizations</td>
<td>extract organizations from each tweet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Virginia Tech</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>sner-locations</td>
<td>extract locations from each tweet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New York; London</td>
</tr>
<tr>
<td>clean-tweet</td>
<td>tweet-importance</td>
<td>The importance value for each tweet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-1</td>
</tr>
<tr>
<td>webpage</td>
<td>html</td>
<td>raw HTML of webpage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[raw HTML text]</td>
</tr>
<tr>
<td>webpage</td>
<td>language</td>
<td>webpage’s main language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[language:confidence,language:confidence]</td>
</tr>
<tr>
<td>webpage</td>
<td>url</td>
<td>full url of the webpage</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://www.roanoke.com/news">http://www.roanoke.com/news</a></td>
</tr>
<tr>
<td>webpage</td>
<td>mime-type</td>
<td>mime type of document fetched</td>
</tr>
<tr>
<td></td>
<td></td>
<td>text/plain</td>
</tr>
<tr>
<td>webpage</td>
<td>status-code</td>
<td>status of the document (success, failure, ...)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>examples and further documentation forthcoming from CMW team</td>
</tr>
<tr>
<td>webpage</td>
<td>title</td>
<td>extract title from the webpage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Student arrested after threatening Virginia Tech Yik Yak post</td>
</tr>
<tr>
<td>webpage</td>
<td>author/publisher</td>
<td>extract author from the webpage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tom LoBianco and Pamela Brown, CNN</td>
</tr>
<tr>
<td>webpage</td>
<td>created-time</td>
<td>extract created-time from the webpage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Apr 13 19:00:21 +0000 2015</td>
</tr>
</tbody>
</table>

59
<table>
<thead>
<tr>
<th>webpage</th>
<th>sub-urls</th>
<th>sub urls in the webpage</th>
<th><a href="http://www.fs.fed.us/">http://www.fs.fed.us/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>webpage</td>
<td>domain-name</td>
<td>extract the domain name from the webpage</td>
<td></td>
</tr>
<tr>
<td>webpage</td>
<td>domain-location</td>
<td>extract the country name from the webpage</td>
<td>us</td>
</tr>
<tr>
<td>webpage</td>
<td>organization-name</td>
<td>extract the organization name from the webpage</td>
<td>Cable News Network</td>
</tr>
<tr>
<td>webpage</td>
<td>fetched-timestamp</td>
<td>fetched time (readable)</td>
<td>Mon Apr 13 19:00:21 +0000 2015</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>clean-text</td>
<td>all HTML tags removed</td>
<td>[clean HTML text]</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>clean-text-profanity</td>
<td>clean text with no profanity</td>
<td>[clean HTML text]</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>clean-text-tokenized</td>
<td>tokenized clean text</td>
<td>[clean HTML text]</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>clean-text-stemmed</td>
<td>stemmed clean text</td>
<td>[clean HTML text]</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>clean-text-stemmed</td>
<td>lematized clean text</td>
<td>[clean HTML text]</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>NER</td>
<td>Name entity recognition. Tag format is desired.</td>
<td>I miss &lt;em class=&quot;LOCATION&quot;&gt;California&lt;/em&gt; at &lt;em class=&quot;TIME&quot;&gt;mid-night&lt;/em&gt;!</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>POS</td>
<td>Part of speech. Tag format is desired. No need to tag stop words.</td>
<td>I &lt;em class=&quot;V&quot;&gt;found&lt;/em&gt; a &lt;em class=&quot;NN&quot;&gt;gun&lt;/em&gt; in &lt;em class=&quot;NN&quot;&gt;California&lt;/em&gt; at &lt;em class=&quot;NN&quot;&gt;mid-night&lt;/em&gt;!</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>Year</td>
<td>Webpage year</td>
<td>2017</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>solr_geom</td>
<td>Derived from spatial-bounding. This field is indexed as a Solr spatial (RPT) field.</td>
<td>ENVELOPE(34.0, -118.2, 36.5, -117.4)</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>keywords</td>
<td>keywords from tags in the webpage</td>
<td>Shooting, california, elementary, gun</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>real-world-events</td>
<td>list of the real world events from collection team</td>
<td>Hurricane Sandy; Hurricane Arthur</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>sner-people</td>
<td>extract names from each webpage</td>
<td>Obama; Jimmy</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>sner-organizations</td>
<td>extract organizations from each webpage</td>
<td>Virginia Tech</td>
</tr>
<tr>
<td>clean-webpage</td>
<td>sner-locations</td>
<td>extract locations from each webpage</td>
<td>New York; London</td>
</tr>
<tr>
<td></td>
<td>topic-list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>topic</td>
<td>1. labels generated by LDA model</td>
<td>Signed, students; event, excited; today, register; april, thanks; community, little</td>
<td></td>
</tr>
<tr>
<td>topic</td>
<td>2. extract the top two labels from each topic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>topic</td>
<td>probability-list</td>
<td>each value presents the probability of the tweet belongs to a certain topic</td>
<td>0.29112; 0.01820; 0.12435; 0.02572; 0.54058</td>
</tr>
<tr>
<td>topic</td>
<td>topic-displaynames</td>
<td>user friendly topic names</td>
<td></td>
</tr>
<tr>
<td>cluster</td>
<td>cluster-list</td>
<td>label of the tweet’s cluster</td>
<td>NAACP stories</td>
</tr>
<tr>
<td>cluster</td>
<td>probability-list</td>
<td>the probability of the doc in the cluster</td>
<td>0.55167194</td>
</tr>
<tr>
<td>cluster</td>
<td>cluster-displaynames</td>
<td>user friendly clusters</td>
<td></td>
</tr>
<tr>
<td>classification</td>
<td>classification-list</td>
<td>list of labels of each document</td>
<td>hurricane</td>
</tr>
<tr>
<td>classification</td>
<td>probability-list</td>
<td>the probability of each label</td>
<td>0.29112; 0.01820; 0.12435; 0.02572; 0.54058</td>
</tr>
<tr>
<td>classification</td>
<td>classification-displaynames</td>
<td>user friendly classification labels</td>
<td></td>
</tr>
</tbody>
</table>