Twitter Semantic Search Project

Sarthak Dash, Umang Patel, Ilambharathi Kanniah

May 10, 2014

Abstract

This project aims at semantic search on Twitter data at a large scale. The use of Latent Semantic Indexing (LSI) has been looked into by previous teams/project. Here, we are using Topic Modelling algorithms namely distributed Latent Dirichlet Allocation to model the Twitter dataset.

1 Data Collection

The Twitter dataset used for this project is from NIST TREC 2011 Microblog dataset (TREC - Text REtrieval Conference)
TREC 2011 Microblog Track Website Link

```
python Tweets_Collect/tweets_collect.py -h
usage: tt3.py [-h] [-s S] [-e E] date
Tweets Collection for TREC 2011
positional arguments:
    date

Example Usage :
python Tweets_Collect/tweets_collect.py 20110128 -s 20 -e 88
```

Extracts all tweets in the date 20110128 starting from file 20 to file 88. The above code extracts tweets by scrapping the Twitter website directly and store them in JSON format.

The structure of the TweetsID dataset from TREC 2011 is as follows,

```
20110123
    |_______ 20110123-000.json
    |_______ ...
    |_______ 20110123-091.json
20110124
    |_______ 20110124-000.json
    |_______ ...
    |_______ 20110124-091.json
...
20110208
    |_______ 20110208-000.json
    |_______ ...
    |_______ 20110208-099.json
```

The tweets provided are from the date range 23rd Jan 2011 to 8th Feb 2011 and has close to 11 million multi-lingual tweets.
The collection of tweets were run on **5 m3.large** AWS machines over 4 days using the scripts in **Tweets_Collection** folder.

Type	Content
TweetID	28965341577084928
ReTweet	RT
Author	hbutti
HastTags	#Tron
URL	[]
Tweet Text	Finally saw the movie #Tron ... have to say that I quite enjoyed it!
Processed Tokenized and Stemmed Text	Final saw the movi have to say that I quite enjoy it

Table 1: Collection of Tweets Format in JSON

2 Apache Lucene

The basic raw code present on the Apache Lucene webpage indexes documents, but for the purposes of this project, we needed to index Tweets. Therefore, we modified the existing demo Indexer in order to come up with our own indexer for indexing tweets that are present across multiple files. In addition, we have written a two bash scripts that would make the process of indexing and search simpler. Assuming that we have already build Lucene, using ant, Here's how to invoke the scripts relating to indexing and searching.

```
./indexDocuments.sh <command> <InputDirectory>
```

This bash script is used to index tweets, present in multiple files. The description of arguments are as follows,

- **command** : can be either **compile** or **index**. In case the **command** equals **compile**, it compiles the java Indexer file. In this setting, the second parameter need not be given.

  In case the **command** equals **index**, the second parameter needs to be given. The second parameter is a directory that consists of files, which in turn have tweets within them.

- **InputDirectory** : Described above.

For the purposes of our project, we had nearly 1500 files, each containing 10K tweets. So following the steps above, we can index the tweets. Here's how we can perform search.

```
./searchDocuments.sh <command/queryFile>
```

This bash script is used to search a given query string against the tweets that we indexed in the previous step. The description of parameters for this script is as follows,

- **command/queryFile** : In case this parameter equals **compile**, this compiles the relevant java SearchDocument file. If we want to perform the search operation, then simply put in all the query strings in a file, and set the value of this parameter as the query file name.

3 Distributed LDA for Semantic Search

We wanted to build an Semantic Search Application, using Topic modelling keeping scalability as a primary design paradigm. We looked into multiple hadoop based LDA implementations, and finally decided on an implementation titled Mr.LDA.jar, here's the github link to the same.

```
https://github.com/lintool/Mr.LDA
```

3.1 Positives and Negatives of MrLDA: Our experiences

For performing Topic modelling on tweets, its necessary to do the same on Tweets, and not on the file containing the tweets. Unlike MrLDA, We found many LDA implementations, that performed topic modelling on documents, and not on each individual line of the document.

Performing LDA using MrLDA involved two steps. The first step was to preprocess the given input(remove stopwords, perform stemming, etc.). We faced a couple of problems; the most important being **setting the appropriate size of HADOOP HEAPSIZE variable**. We were using **4 m3.xlarge Amazon EMR machines**, and by trial and error we found that setting the heap size 20GB worked for us.

The second step was to do LDA training, but we found out that this step was extremely slow(**about 1% completion in 30 minutes with 4 m3.xlarge machines**). Despite our efforts, such as going on for bigger machines(with the same number of machines), we couldn't improve the speed greatly.

Despite the fact that it was a wonderful application, we had to discard it because of speed. And we decided to try out **Apache Mahout** suite of Machine-learning algorithms for LDA, especially the Collapsed Variational Bayes algorithm.

3.2 Apache Mahout : Our experiences

Performing Topic Modelling using Apache Mahout comprises of Five steps. We have listed down these, along with our modifications(if-any) and the innumerous versioning/compatibility issues that we faced and successfully/unsuccessfully overcame.

We have used Apache-mahout-0.8 for our project. From within the mahout-distribution folder, run the following commands, within Hadoop 2.x version. These wouldn't run properly within Hadoop 1.x version(throws lots of versioning errors).

```
hadoop jar mahout-examples-0.8-job.jar org.apache.mahout.text.SequenceFilesFromDirectory
-i /se/dataset -o /se/outputseq -xm sequential

./mahout seq2sparse -i /se/outputseq -o /se/outputsparsedvec --namedVector -wt tf
```

For versioning issues, the above two commands only run on Hadoop 2.x version, while the below two commands run only on Hadoop 1.x version. We tried hard to make all four of these commands run on a single Hadoop version, but couldn't succeed. Therefore, we used two EMR machines, one having Hadoop 2.2.0(for the top two commands) and the second having Hadoop 1.0.3 (for the bottom two commands)

```
./mahout rowid -i /mahoutlda/outputsparsedvec/tf-vectors -o /mahoutlda/matrix

./mahout cvb -i /mahoutlda/matrix/matrix -o /mahoutlda/lda-output -mt
/mahoutlda/ldaoutput/models -dt /mahoutlda/ldaoutput/docTopics -dict
/mahoutlda/outputsparsedvec/dictionary.file-0 -k 400 -x 40 -ow
```

The previous command completes the LDA training for 400 topics for 40 iterations. At the end of this step, the output is still in SequenceFile format, and we need it to be in human-readable format so as to do postprocessing. Hence, the final command, (Note that this has to run again on Hadoop 2.x, since it throws errors with the `--sortVectors` option with Hadoop 1.x).

```
./mahout vectordump -i /mahoutlda/ldaoutput/docTopics \
-o /mahoutlda/ldaoutput/output-docTopics \
-p true \
-d /mahoutlda/outputsparsedvec/dictionary.file-0 \
-dt sequencefile \
-sort /mahoutlda/ldaoutput/docTopics \

./mahout vectordump -i /mahoutlda/ldaoutput/model/model-1 \
-o /mahoutlda/ldaoutput/output-model-1 \
-p true \
```

```
-d /mahoutlda/outputsparsedvec/dictionary.file-0 \
-dt sequencefile \
-sort /mahoutlda/ldaoutput/models/model-1 \
```

The above two vector dump commands are run for getting the topic-document matrix in a sequence file format into human-readable format and topic-term matrix in a sequence file format into human-readable format. The topic-term matrix is a large matrix hence stored across many part model files named model-1 through model-10 (in this case).

3.3 Post-processing

- vectorize.py
  Command : python vectorize.py path_to_inputfolder path_to_queryfile
  Output : This script takes the input queryfile having one query per line and outputs the corresponding query vector for each query. The query vector is made from term topic matrix.
  Here the query vector is built from the term-topic matrix (could also do query vector build from document-topic matrix, in the ideal case each document is one tweet, but in our each document has close $\tilde{6}725$ tweets each). So, each query vector is the average of all the topics vectors by taking the topic vector corresponding to each term in the query.

- output.py
  Command : python output.py path_to_folder1 path_to_folder2
  Arguments : folder1 : folder containing files having the output from CDS.py , folder2 : folder containing files having tweets
  Output : Since, the tweet to Topic distributions spans over multiple files, and CDS.py gives us results for each individual tweet to Topic distribution file, these results(spanning across multiple files) needs to be combined in order to extract top 1000 tweets.

  In this context, this script takes the output format from files in folder1 and displays the corresponding tweets . This code reads each file in folder1 having $\{Linenumber\_filename : CDSscore\}$ and for each query tweet, sorts the obtained tweets(in previous step) via their CDS scores. Once the sorting is done, it extracts tweets from files in folder2 using filename and linenumber . The tweets are displayed in the following format {Tweet_id : Tweet}.

- CDS.py
  Command : python CDS.py arg1 arg2
  Arguments : arg1 : file containing tweet to topic distribution, arg2 : file containing query vectors
  Output : This script computes the CDS score (Cosine Similarity) between each tweet in the training dataset and the given input query, and for each query tweet, it outputs top 10 similar tweets.

- stopwords.py
  Command : python stopwords.py arg1 arg2
  Arguments : arg1 : folder containing files to be cleaned , folder2 : file containing stop words
  Output : This script removes all the stop words from files in folder of arg1 . Arg2 is a file having all the stop words with each stopword in new line .

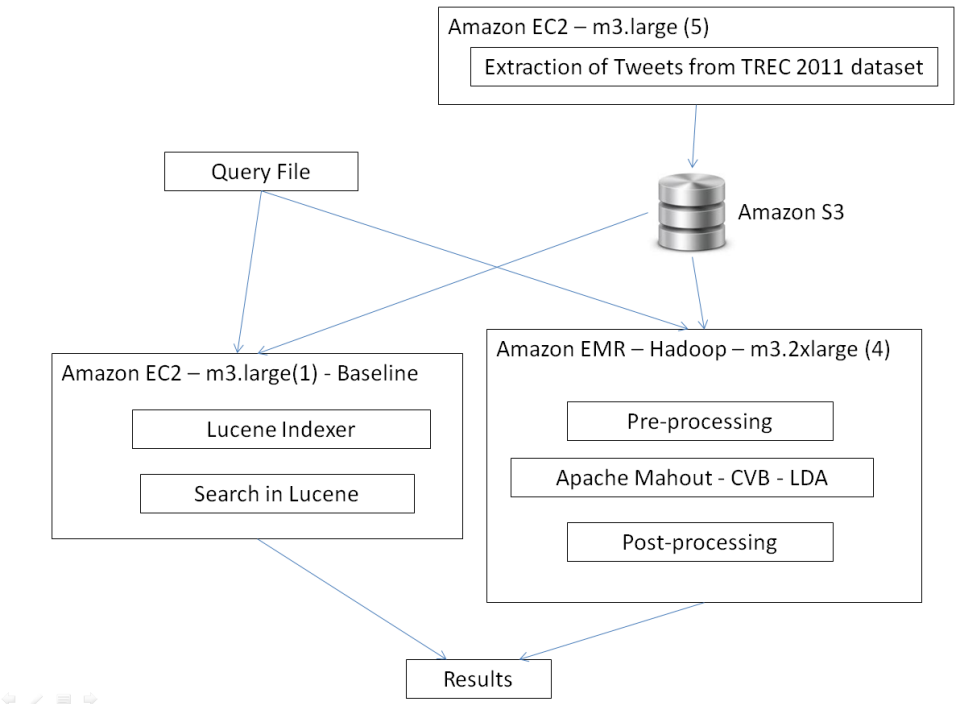

Figure 1: Process Flow Diagram

4 Issues and Stuff that doesn't work

In addition to issues already mentioned earlier, these are other issues that we faced along the way/stuff that doesn't work, and we learned it the hard way. We hope that it helps some one, who's trying to build his first hadoop based ML application.

- Tweets collection was the first major challenge we faced . Twitter API has a download limit of 180 tweets/15 minutes . Calculating that way the average download time came out to be 73 days. So we thought of applying for whitelisting to Twitter which takes 1 week. In the end we decided to write our own HTML scrapping code which helped us to gets tweets faster. The new downloading time was reduced to 5 days.

- Apache Mahout allows the user to perform Topic modelling on documents(containing 10,000 tweets each), but it was required of us to perform Topic modelling at a tweet level and not at a document level.

  A naive approach that we tried initially, was to write each tweet in a file, and then use these files as input to Mahout's CVB algorithm. However, this idea didn't work since, loading these files from a a local file system to hdfs itself takes a lot of time(in the order of hours).

- Continuing on the previous issue, we figured out that the CVB algorithm(in Apache Mahout) converts each document into a SequenceFile, before passing it off to the actual training algorithm. We did work on this idea, wherein we built our own SequenceFile generator at the tweet level granularity(Its called as the SequenceFileWrite.jar).

  We converted the entire dataset onto Sequence files, and attempted to start from the second stage of the CVB algorithm. Unfortunately, the execution of the second stage failed, and on googling the stack trace, the most common reply on threads was to go for an updated Mahout implementation. We even tried the same with Apache-Mahout 0.9(the most up-to-date Mahout implementation), but still the same error kept crawling up. Eventually, after beating our heads for a day or two, we decided to give up on this idea.

- During testing, the most common error that we faced was exceeding **JAVA HEAP space**. Now, there are lots of configuration files on hadoop, and it took us some time to figure out which

configuration file to change, for increasing **JAVA HEAP space**. Besides, related errors such as **GC Memory Limit exceeded** kept crawling up in the meantime and ate up a lot of our development time. But it was an interesting learning curve.

- Running 4 clusters on Amazon Web Services , the total bill came out to be 606$.