Topic Analysis

CS5604 – Spring 2016

Extracting Topics from Tweets and Webpages for the IDEAL Project

May 4, 2016

Submitted by

Sneha Mehta sudo777@vt.edu

Radha Krishnan Vinayagam vradha@vt.edu

Instructor

Prof. Edward Fox

Virginia Polytechnic Institute and State University
Blacksburg, Virginia

Abstract

The IDEAL (Integrated Digital Event Archiving and Library) project aims to ingest tweets and web-based content from social media and the web and index it for retrieval. One of the required milestones for a graduate-level course CS5604 on Information Storage and Retrieval is to implement a state-of-the-art information retrieval and analysis system in support of the IDEAL project. The overall objective of this project is to build a robust Information Retrieval system on top of Solr, a general purpose open-source search engine. To enable the search and retrieval process we use various approaches including Latent Dirichlet Allocation, Named-Entity Recognition, Clustering, Classification, Social Network Analysis and Front end interface for search.

The project has been divided into various segments and our team has been assigned Topic Analysis. A topic in this context is a set of words that can be used to represent a document. The output of our team will be a well defined set of topics that describe each document in the collections we have. The topics will facilitate a facet based search in the frontend search interface.

In this report we introduce the relevant background, design & implementation and the requirements to make our part functional. The developer's manual describes our approach in detail and walk-through tutorials for related software packages have been included in the user's manual. Finally we provide exhaustive results and detailed evaluation methodologies for the topic quality.

Contents

Abstract.....2

List of Figures.....4

List of Tables.....5

1. Introduction.....6
2. Background Knowledge and Literature Review.....6
   2.1 Important Chapters for Topic Analysis.....6
   2.2 Literature Review.....7
3. Work Plan and Deliverables.....10
4. Developer manual.....11
   4.1 Requirements.....11
      4.1.1 Information Retrieval System Overview.....11
   4.1.2 Topic Analysis Component Functional Requirements.....12
      4.1.2.1 Acquiring data and preprocessing.....12
      4.1.2.2 Building the model and evaluation.....12
      4.1.2.3 Storing data in HBase.....13
      4.1.2.4 Collaborating with other teams.....13
   4.1.3 Inter-dependencies between teams.....13
   4.2 Detailed Design.....14
4.3. Implementation.....15
   4.3.1 Topic Model.....15
   Data preprocessing.....15
   4.3.2 Topic labeling.....17
   4.3.2.1 Manual approach.....17
   4.3.2.2 Automated approach.....17
   4.3.3 Output in HBase Schema.....17
4.4. Experiments and Results.....20
      4.4.1 Cleaned Collections.....20
      4.4.2 Experiments on Collections.....21

4.4.3 Results for Tweet Collections 22
4.4.3 Comparison of Tweet and Web page results 23
4.4. LDA Evaluation 24
4.4.1 Choosing number of topics 24
4.4.2 Choosing number of iterations 24
4.4.3 Evaluation of output 24
5. User Manual 26
5.1 Spark Installation 26
5.2 Apache Spark 31
6. Lessons Learnt 36
7. Conclusion and Future work 37
8. Acknowledgements 37
9. References 38
10. Appendix 39
10.1 Sample words after pre-processing 39
10.2 Topic-Word output for different input parameters 39
10.3 Document - Topic distribution for WDBJ Shooting collection 41
10.4 Top words for each topic 43
10.5 User Study - Topic Modeling 43

List of Figures

Figure 1: Topic Analysis for NIPS papers.....9
Figure 2: Project plan.....10
Figure 3: Topic analysis using LDA.....12
Figure 4: Overview of the IR System.....14
Figure 5: Topic model design.....15
Figure 6: Tweet after preprocessing.....16
Figure 7: Document - Topic Output from HBase.....19
Figure 8: Topic-Word Output from HBase.....20
Figure 9: Time size plot for each collection.....22
Figure 10: Evaluation result.....26
Figure 11: Starting Spark Shell.....31

List of Tables

Table 1: Project deliverables....10
Table 2: Sample Document - Topic distribution....16
Table 3: Sample Topic - Word distribution....17
Table 4: Document-Topic HBase schema....18
Table 5: Sample Document-Topic output from HBase....18
Table 6: Topic-Word HBase schema....19
Table 7: Sample Topic-Word output from HBase....20
Table 8: Cleaned collections....21
Table 9: Running time for tweets....21
Table 10: Running time for webpage collection....21
Table 11: Topics for tweet collections....21
Table 12: Topics in Obamacare collection....23
Table 13: Top words in Obamacare collection....23
Table 14: Evaluation results....25
Table 15: Twitter-Topic distribution for shooting collection....41

1. Introduction

The focus of our team is developing the topic analysis component of the information retrieval system. Topic analysis helps in discovering abstract topics from tweets and webpages. This would enable the search and retrieval of documents based on topics. The frontend team uses the topics that are identified by our module to design a faceted search. The topics that are identified can also be used in validating results of the clustering team. For designing a topic model we will use the Latent Dirichlet Allocation (LDA) [1] technique.

The overall report has been subdivided into various sections. First, a literature review and related references are provided for readers to understand the algorithm of LDA and if required, to know details about the LDA model. Second, we give the overview of our work plan. From this part, readers will get an overall idea of our work outline to understand the function of LDA in the whole project and know the relations and inter-dependencies between our team and other teams.

Then we go on to describe in detail the overview of the whole information retrieval system framework and describe the inter-dependencies between different teams. Sections 4.1 through 4.4 describe in detail the implementation and evaluation of our approach. Finally, we include the user's manual that contains a step-by-step walk-through of the important software packages and relevant software installation.

2. Background Knowledge and Literature Review

2.1 Important Chapters for Topic Analysis

The textbook for the course 'Introduction to Information Retrieval' by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze [3] has been an indispensable source of information for gaining background knowledge about topic modeling and putting it in perspective with respect to other teams. The chapters relevant for topic modeling are chapter 6, 7, 8 and 18. Among them, chapters 6 and 7 are the most important chapters. Chapter 6, which is "Scoring, term weighting and the vector space model", talks about TF-IDF which is a fundamental weighting scheme used in general information retrieval systems. Studying this chapter will give us an idea about the importance of document indexing and scoring. Chapters 6 and 7 introduce the basic vector space scoring model and its variations. Chapter 7 also introduces an overview of the complete information retrieval system. Moreover, we need to learn more about LDA. Although the textbook does not provide much detail about LDA, it provides useful references to study the topic in detail. Since LDA is an initial probabilistic extension of the latent semantic indexing (LSI) technique, it is necessary for our team to review Chapter 18, especially the LSI part. "Advanced analytics with Spark" by Sandy Ryza, Uri Laserson, Sean Owen and Josh Wills [8] gives necessary background to implement the LDA model in Spark ML with Scala programming [15].

2.2 Literature Review

A topic model uses Bayesian statistics and machine learning to discover the latent, or hidden, structure in given documents and build the links between documents. Then based on this knowledge, topic models can also output topic compositions of future documents. Latent Dirichlet allocation (LDA) [1] is one such widely used topic model.

LDA is a model that allows sets of observations to be explained by unobserved groups that can represent the similarity of the data. For example, in the case that an observation is a collection of words in a document, LDA extracts a set of words that are likely to be able to describe the document and uses this set of words to represent the document. The paper "Latent Dirichlet Allocation", written by David M. Blei, Andrew Y. Ng and Michael I. Jordan first introduced the technique of LDA.

LDA gives two probability distributions. One is the probability distribution over words in the same topic and another is the probability distribution over topics in the document. For example, in the topic of "sports", there will be some words, such as "football", "basketball", "soccer", which occur together frequently, and then LDA gives the probability distribution over these words. Also, by examining the documents, one document can be said as 100% about "sports" or 50% about "sports" and 50% about "politics". In other words, LDA represents documents as mixtures of topics that spit out words with certain probabilities. It assumes that documents are produced in the following way:

- Decide on the number of words N the document will have (say, according to a Poisson distribution).
- Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of K topics).
- Generate each word w_i in the document by:
    - Picking a topic (according to the multinomial distribution that you sampled above).
    - Using the topic to generate the word itself (according to the topic's multinomial distribution).

Before giving the model of Blei's paper [1] , we also need to illustrate the concept of Bayesian statistics. It can be described as follows:

Prior distribution $\pi(\theta)$ + sample information X => Posterior distribution $\pi(\theta|x)$

It tells us the observed sample information will update the knowledge we have for the objects. To be specific, the information will correct the value of parameter θ. It is the process of machine learning for LDA.

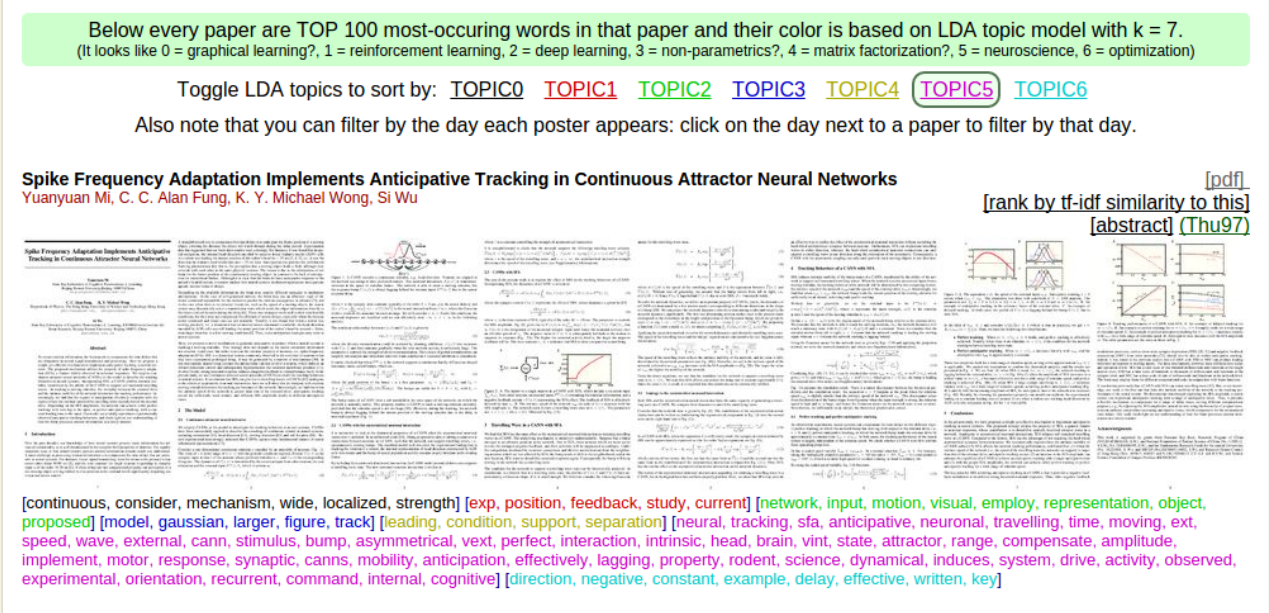
Above we have given the process for forming a document using LDA. However documents and words are obvious but topics are not. So how can we infer the topics from given documents and words?

Given a document how can we know the distribution of topics and the distribution of words in a particular topic? For this, we need to estimate the two arguments in the LDA model. There are different ways to calculate arguments of Φ and Θ. Among them, we want to talk a little about Gibbs Sampling [10], which is based on the Bayesian paradigm. Gibbs Sampling algorithm is a little complicated idea in mathematics.

P(topic|doc) * P(word|topic) is the probability for the way doc—>topic—>word.

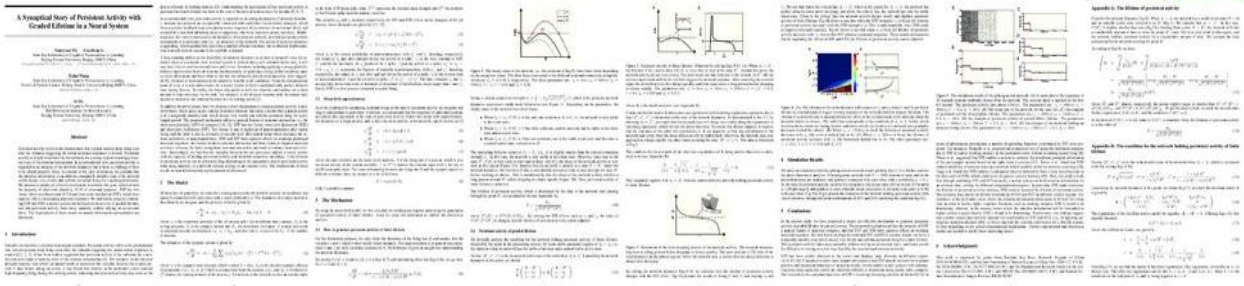
By calculating the Posterior distribution of topics and words, we successfully know the distributions of topics and words, which are related to unknown arguments. The LDA model is a Bayesian version of pLSA model, which makes it perform well on small datasets. This advantage comes from the good quality of a Bayesian method that it will not overfit data. And for big datasets, the performance of both pLSA and LDA is almost the same. So LDA is widely used nowadays.

Now we provide one example [14] of topic modeling usage. In this example, topics are extracted for a dataset consisting of all technical papers from the conference NIPS (Neural Information Processing Systems), an artificial intelligence conference. As you can see in Figure 1, seven topics are extracted from the contents of all the papers accepted in the conference in the year 2014. The topics have been labeled: Topic0 - graphical learning, Topic1 - reinforcement learning, Topic2 - deep learning, Topic3 - non-parametric, Topic4 - matrix factorization, Topic5 - neuroscience, Topic6 - optimization. When a topic, say Topic5, is selected, papers are ranked in ascending order of the quantity in which they contain words from that topic. As can be seen, the first document has a large number of words colored pink with words from Topic5. The pink words keep on decreasing, as you go down the results.



A Synaptical Story of Persistent Activity with Graded Lifetime in a Neural System
Yuanyuan Mi, Luozheng Li, Dahui Wang, Si Wu

[pdf]
[rank by tf-idf similarity to this]
[abstract] (Tue71)



[mechanism, consider, strength, tsp] [appendix, neutral, order, probability, depending] [network, input, china, long, view, proposed, conventional, wang, natural] [figure, amount, simulation, process] [condition, matrix, point, smaller, close] [persistent, activity, neural, state, lifetime, neuronal, time, stf, stimulus, attractor, system, external, stable, std, synaptic, stp, neuron, memory, silent, prefrontal, three, holding, decay, cortex, membrane, slowly, property, cortical, spike, neuroscience, science, beijing, brain, jij, retaining, arrival, connection, graded, cognitive, unstable, silence, prolonged, depression, poisson, marginally, dynamical] [working, rate, active, hold, critical, computational, theoretical, example, key, plateau, strong, solving]

Fast Sampling-Based Inference in Balanced Neuronal Networks
Guillaume Hennequin, Laurence Aitchison, Mate Lengyel

[pdf]
[rank by tf-idf similarity to this]
[abstract] (Mon45)

[random, solution, fact, balanced, variable, provide, law] [optimal, order, appropriate, max, main] [network, learning, input, scale] [sampling, distribution, posterior, covariance, gaussian, model, mixing, inference, latent, gibbs, sampler, monte, markov, carlo, stationary, chain, sample, figure, university] [matrix, fast, faster, symmetric, diagonal, pairwise, small] [langevin, time, activity, neural, speed, recurrent, optimized, slow, noise, slowing, inverse, choice, synaptic, magnitude, weight, brain, circuit, fastest, neuronal, excitatory, state, behavior, connectivity, inhibitory, drawn, external, neuron, slowest, nonlinear, control] [stochastic, linear, large, note, machine, constant, size, statistical, cost, depends, optimization, theory, increasing]

A Multiplicative Model for Learning Distributed Text-Based Attribute Representations
Ryan Kiros, Richard Zemel, Ruslan R. Salakhutdinov

[pdf]
[rank by tf-idf similarity to this]
[abstract] (Tue102)

[consider, notion, correspond, set] [context, factored, interesting, illustrates] [word, language, attribute, sentence, learning, representation, english, blog, table, sentiment, network, learned, embedding, recursive, embeddings, author, text, atd, training, german, proposed, trained, learn, richard, computed, test, represented, work, performance, industry, performed, christopher, geoffrey, task, authorship, semantic, gender, metadata, french, train, evaluation, subphrases, jointly, andrew, character] [model, conditional, generate, parallel, allows, figure] [vector, tensor, matrix, denotes, compute, corresponding, denote, small] [neural, multiplicative, conditioned, corpus, three, capture, represent, inferred, document, correlation, recurrent, experimental] [distributed, method, linear, size, dimensional, framework, corresponds]

Figure 1: Topic analysis for NIPS papers (Courtesy: https://cs.stanford.edu/people/karpathy/nips2014/, Andrej Karpathy)

3. Work Plan and Deliverables

The project work plan is represented through the Gantt chart in Figure 2. We have scheduled our work on weekly basis. The project was started on January 18, 2016 and it runs till May 4, 2016. The project deliverables are detailed in Table 1.

Deliverable Name	Date
Interim report 1	February 12, 2016
Interim report 2	March 11, 2016
Interim report3	March 31, 2016
Interim report 4	April 20, 2016
Final report and presentation	May 4, 2016
Code base	May 4, 2016

Table 1: Project deliverables

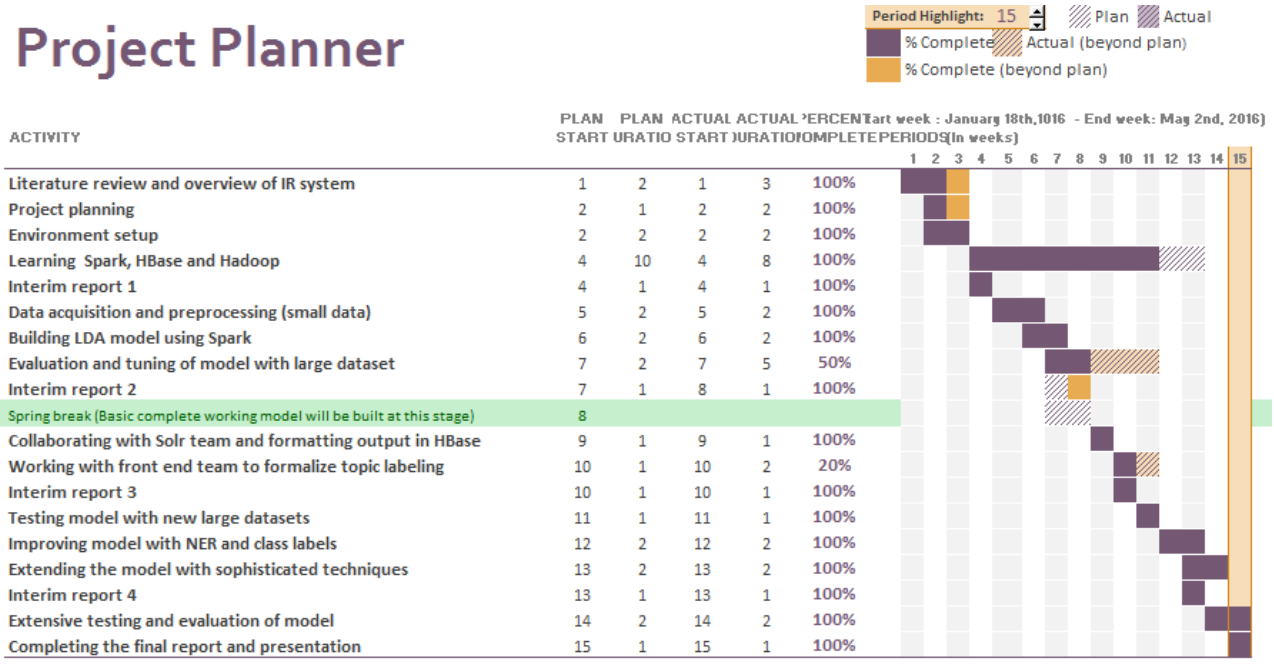


Figure 2: Project plan

4. Developer manual

This section details our development methodology including data flow, architecture and module overviews. It includes the complete instructions for project setup, project installation and maintenance.

4.1 Requirements

This section provides the detailed requirements. The first subsection describes the details of the overall information retrieval system and shows where the Topic analysis component fits into the big picture. The second subsection provides the functional requirements of the topic analysis component and the third subsection analyzes the interdependencies between our team and other teams.

4.1.1 Information Retrieval System Overview

The system is designed with the aim of providing a seamless search experience to the end-users. To allow the users to efficiently search for the required documents based on the events, various techniques including classification, clustering, LDA, NER, Noise Reduction and Social Network Analysis, will be integrated in the system. Figure 3 gives an overview of the whole project.

Our team focuses on the topic analysis component in this system. In topic analysis we build a model for discovering abstract "topics" that occur in the collection of documents. At a high level, topic modeling aims to find structure within an unstructured collection of documents. After learning this "structure," a topic model can answer questions such as: What is document X discussing? How similar are documents X and Y? If I am interested in topic Z, which documents should I read first? LDA is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics. To build the topic model we use LDA and determine the topic distribution for each document.

The flow of the whole system (Figure 4) can be divided into 2 major parts: preprocessing and query time processing. The preprocessing phase comes before query time processing. The preprocessing phase can be considered as offline processing, while the query time processing is online processing.
In the preprocessing phase, the system takes the documents (tweets and webpages) as the input and creates features vectors using LDA and other components (original Solr scoring, NER, Classifying Types, Extraction and Feature Selection, Clustering, and Social Network Analysis). These feature vectors will be stored in the Solr database and will be used further in the query time processing. The query time processing occurs when a user performs searching. The query string from the user will be used to create the feature vectors as in the preprocessing phase. Then the vectors from this step will be compared to the preprocessed vectors of each document in order to evaluate the importance of the document. The documents will be ranked based on the custom ranking algorithm developed by the Solr team. The documents with high importance will be shown to the user.

4.1.2 Topic Analysis Component Functional Requirements

To make our component functional we have divided the task into four areas:

1) Acquiring data and preprocessing
2) Building model and evaluation
3) Storing data
4) Collaborating with other teams

4.1.2.1 Acquiring data and preprocessing

Our first step is to load and preprocess the documents for our own use. The class has been provided with data containing a collection of tweets divided into six categories including 'Accident', 'Bombing', 'Storm', 'Shooting', 'Disease' and 'Community' and a collection of web pages. For each of the tweet collections there are two collections; one is smaller in size than the other. We will use the data provided by the collection management (CM) team. The CM team has cleaned the tweets of the 'Shooting' collection. The cleaned tweets have URLs, profanities and non-characters removed from the raw tweets.

4.1.2.2 Building the model and evaluation

We will use Apache Spark's ML library to build the LDA model from our preprocessed data. After building the model we can get a topic distribution for each of the documents and words contained in each topic. Some analysis can be performed to fine-tune the number of topics to get good quality topics representative of the data. Figure 3 shows the input and output of LDA.

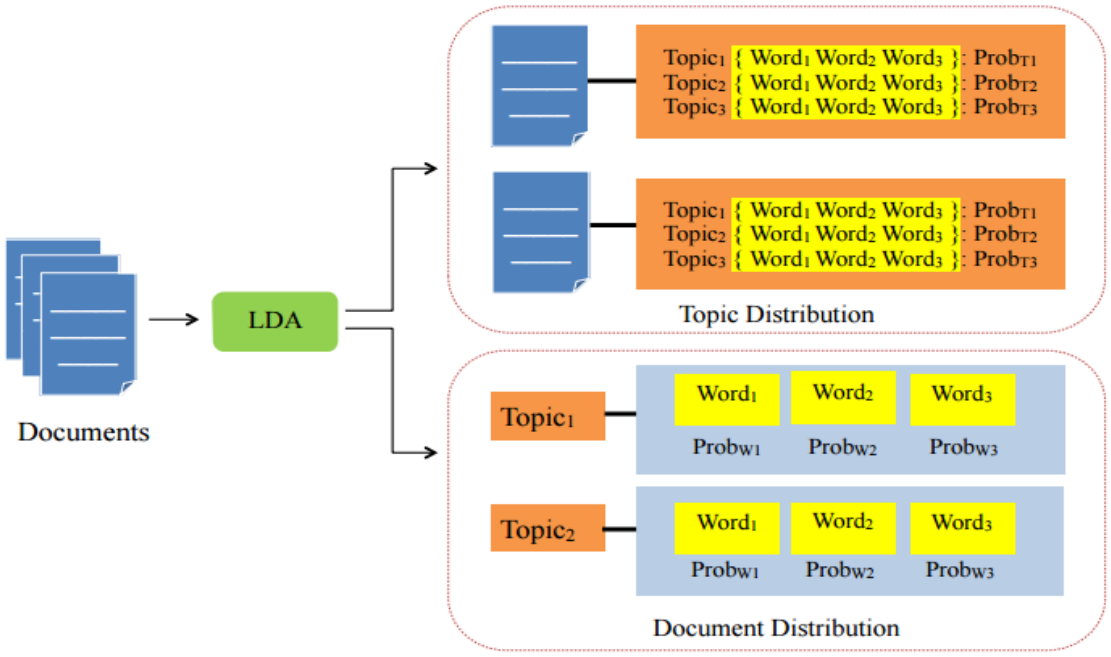


Figure 3: Topic analysis using LDA (Courtesy: LDA team report Spring 2015)

4.1.2.3 Storing data in HBase

The topic data will be stored in HBase, a database supported by Solr and Spark. We collaborate with the Solr team to decide upon a schema for an HBase table that will store the information about the extracted topics.

4.1.2.4 Collaborating with other teams

The Clustering team will cluster the documents using algorithms like k-means to obtain clusters and a cluster membership for each document. We will compare the topics we obtained from our analysis with the cluster centers generated by the clustering team. In this way, we can collaboratively evaluate the quality of our topics and clusters and make improvements. Finally, we will also collaborate with the Frontend team. The topics from our model will be fed to the Frontend web interface for faceted search. Topics will be shown as facets for search in the UI. Once the 'topic' facet is selected documents can be searched based on the topics we generated.

4.1.3 Inter-dependencies between teams

1. URLs, hashtags and mentions are extracted from the tweets using Apache Pig by the collection management team. They have also cleaned the tweets by removing from the raw tweets the URLs, profanities and non-characters.
2. The cleaned tweets will be fetched from HDFS and indexed by the classification team, the clustering team, NER team and our team.
3. Our team is responsible for extracting topics from the tweets and webpages which will be used for faceted search.
4. For our team, after getting the extracted topics, we store the topics and the document-topic distribution in HBase. We will provide the Solr team a schema for storing the topic related data. The Solr team is responsible for storing our data in HBase and indexing the documents based on topics.

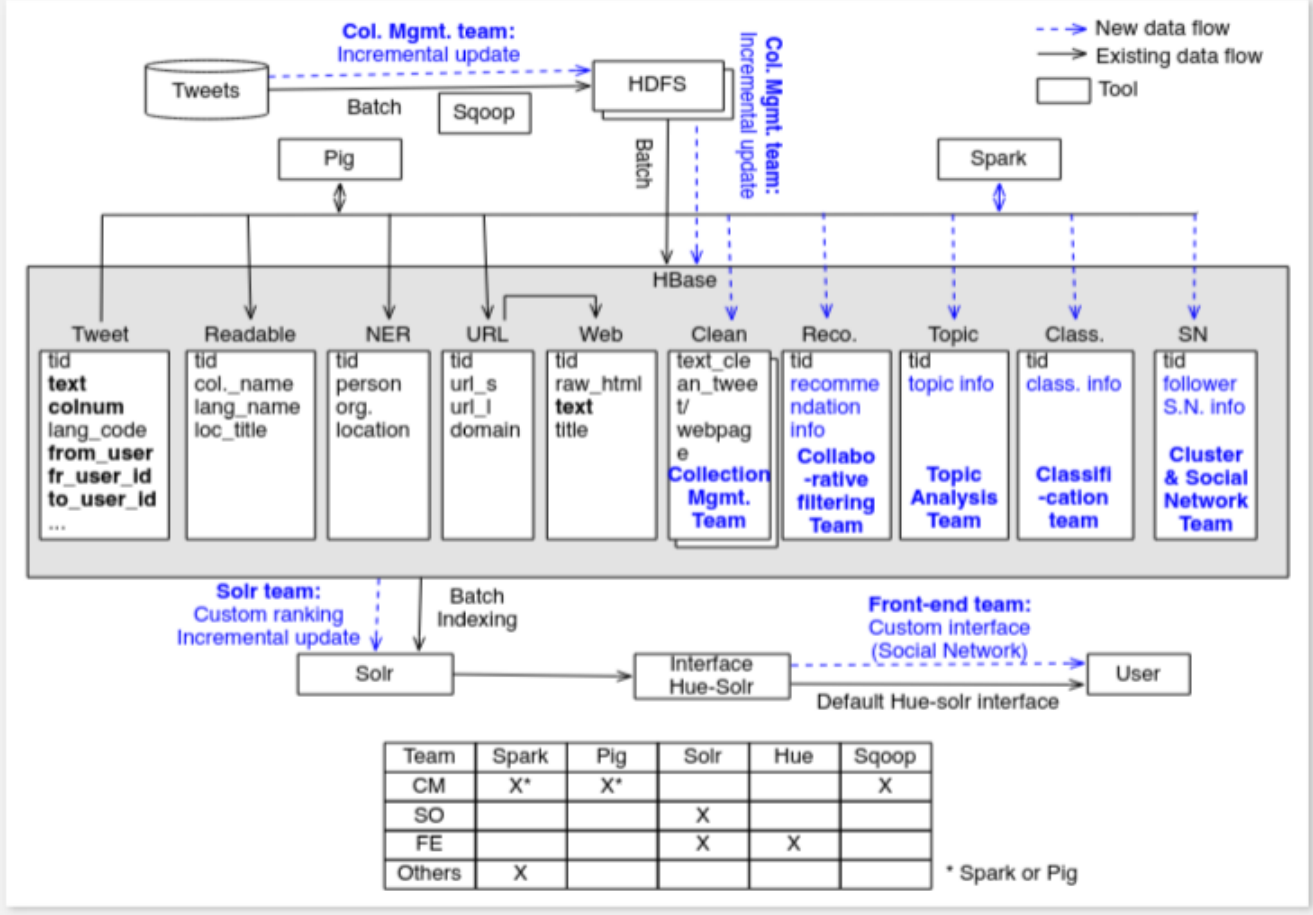


Figure 4: Overview of the IR system (Courtesy: Dataflow diagram 2016)

4.2 Detailed Design

Figure 5 shows the detailed design of the topic model. We are building our LDA model using Apache Spark's machine learning library or MLlib. We will be using cleaned tweets and webpages from the collection. The data for the model comes from HBase which will be processed with Spark. The topic vectors which are generated from the model are stored again in the HBase. This will be indexed and passed to the frontend team for enabling faceted search.

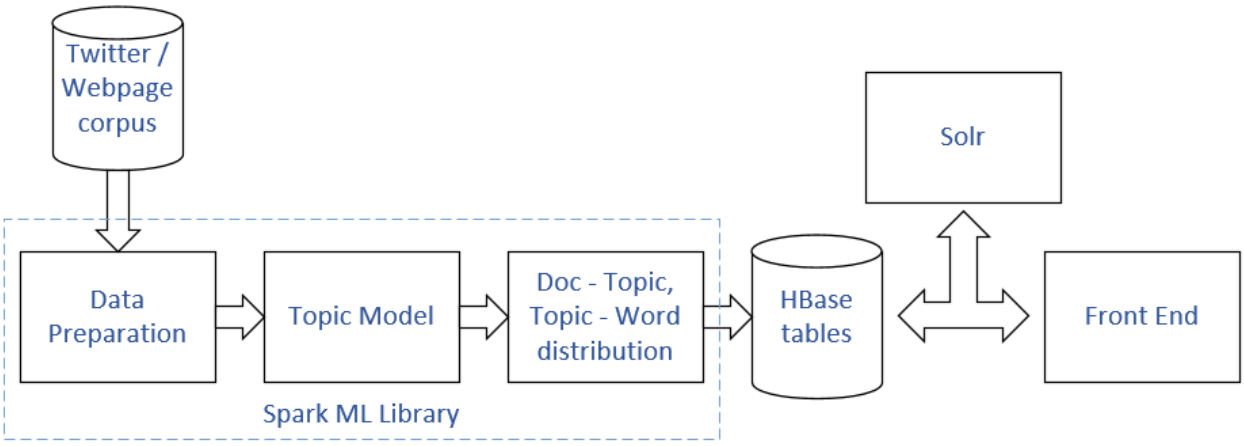


Figure 5: Topic model design

4.3. Implementation

4.3.1 Topic Model

We have implemented the LDA component [11] to extract topics from the shooting collection obtained from the collection management team. As mentioned before, collection management has cleaned the tweets data from the 'Shooting' collection by removing URLs, profanities and non-characters from the raw tweets. We have used Apache Spark 1.5.2 Scala API for implementation and we executed the program in Cloudera CDH 5.5 Hadoop cluster.

First we create a directory `'tweets_data'` in HDFS and then copy the `'cleaned_tweets'` directory provided by the CM team to the HDFS directory.

```
$ hadoop fs -mkdir  tweets_data
$ hadoop fs -put cleaned_tweets tweets_data
```

Data preprocessing

The tweets are loaded and preprocessed. The most important part of preprocessing is choosing a vocabulary. We split text into terms (words) and then remove (a) short terms with < 4 characters, and (b) the most common 25 terms (as "stopwords"). It is important to adjust this preprocessing for the particular dataset in use. We experimented with these parameters and got acceptable results for numStopwords = 25 and removing terms with less than 4 characters. We also created custom stopwords list to remove words which didn't make sense in our trial runs. For instance, figure 6 shows a tweet in which the highlighted words are the ones that will get removed by our preprocessing steps.

Figure 6: Tweet after preprocessing

Running LDA

We then run LDA using 5 topics and 100 iterations. Again, it is often important to choose the number of topics based on the particular dataset in use. In Section 4.4 we have provided detailed analysis of results obtained by varying these parameters. The code for Spark implementation of LDA can be found in the Developer's manual. The code can also be downloaded from GitHub https://github.com/sumehta/ideal_topic_modeling.

From LDA, we obtain Document - Topic and Topic - Word matrices as the output. The Document-Topic matrix describes the topic composition of each document in the corpus. In the Appendix(Section 10.3), we have provided the Document-Topic matrix for the WDBJ shooting collection. As a sample, the Document-Topic distribution for a couple of documents is given below:

Document No.	Topic 1	Topic 2	...	Topic N
Tweet1	0.7581	0.1210	...	0.001
Webpage1	0.5275	0.4214	...	0.012

Table 2: Sample Document - Topic distribution

The other output, Topic - Word matrix contains the top 10 words for each topic. It will also include the probability for each word in the topic. A sample for Topic - Word distribution is given below:

Number of topics = 2

Topic 1		Topic 2	
reporter	0.0234	Survivor	0.0548
heartbroken	0.0220	police	0.0522
condolence	0.0219	moment	0.0393
suspect	0.0178	obama	0.0310
Westandwithwdbj	0.0168	President	0.0302
everytown	0.0164	deaths	0.0270
condition	0.0158	update	0.0263
Youtube	0.0132	Community	0.0253
photographer	0.0130	Spoke	0.0245
fatal	0.0128	suicide	0.0244

Table 3: Sample Topic - Word distribution

4.3.2 Topic labeling

It is useful to provide a semantic label for all the topics obtained from LDA. We can do this either manually or through an automated approach.

4.3.2.1 Manual approach

LDA gives the top 10 words and their probabilities for each topic. By manually interpreting the top 10 words, a semantic label can be assigned to a topic . This method is applicable for both tweets and webpages. Though this is time consuming, it is feasible for small collections.

4.3.2.2 Automated approach

We have developed an automated technique to come up with the labels for the topics. LDA gives us top 10 words for each topic. Using these words, we label each topic with its most probable / relevant word. If a word is already used to refer a topic, then we use next best word.

4.3.3 Output in HBase Schema

There will be one main HBase table where the different teams will store their data in their own column families. The Solr team will index this data and retrieve it when a user makes a query. Our data will be stored in a Column Family(CF) in this table. We will move our *topic - document* and *topic - word* data to HBase and provide it to the Solr team and the frontend team. The *topic - document* distribution could be used by the frontend team to enable faceted search. This output can also be used by the clustering team to cross-check their results. Once all the small collections are cleaned and loaded into HBase, we will run our model using that data and save the results in our column families.

4.3.3.1 Document - Topic matrix

For storing document topics distribution we have two column families. One CF is for storing the topic distribution for tweets and another CF for storing the topic distribution for webpages. In the Tweets CF, there are three columns. The first column stores the topics list as Topic1, Topic2, Topic3 an so on. The second column stores the probabilities for the corresponding topics as 0.76898, 0.20861, 0.0224

and the third column will contain the semantic labels that we assign to our topics based on an automated and/or a manual method. Note that here we show three topics just as an example. The analysis for the optimum number of topics has been shown in the evaluation section for the small data collection 'Shooting' that has been cleaned by the Collection Management team.

Row Key	tweets_topics		webpages_topics	
	Label List	Probability List	Label List	Probability List

Table 4: Document - Topic HBase Schema

Example Output

Row Key	tweets_topics		webpages_topics	
	Label List	Probability List	Topic No. List	Probability List
Tweet #34	WDBJ7, Shooting, Victim	0.76898, 0.20861, 0.02241		
Tweet #9166	WDBJ7, Shooting, Victim	0.17742, 0.20404, 0.61854,		
Webpage #15			WDBJ7, Shooting, Victim	0.52892, 0.39909, 0.07199
Webpage #7608			WDBJ7, Shooting, Victim	0.47392, 0.45991, 0.06617

Table 5: Sample Document - Topic Output

```
hbase(main):007:0> scan 'ideal-tweet-cs5604s16', {COLUMNS => "tweets_topics", "LIMIT" => 5}
ROW                                   COLUMN+CELL

 700-636747110092771328               column=tweets_topics:probability_list, timestamp=1460483638442, value=0.19063803

                                      0.21028375132479396,0.18606402130471783,0.18315940932871153

 700-636747110092771328               column=tweets_topics:topic_number, timestamp=1460483638442, value=Topic1,Topic2,

 700-636858740902203392               column=tweets_topics:probability_list, timestamp=1460483638442, value=0.18111492

                                      0.22331073637388515,0.16876572657999178,0.20285922415486024

 700-636858740902203392               column=tweets_topics:topic_number, timestamp=1460483638442, value=Topic1,Topic2,

 700-637105099270045696               column=tweets_topics:probability_list, timestamp=1460483638442, value=0.18019529

                                      0.17748928548346826,0.24868820104673106,0.20453240194470268

 700-637105099270045696               column=tweets_topics:topic_number, timestamp=1460483638442, value=Topic1,Topic2,

 700-642024509902483456               column=tweets_topics:probability_list, timestamp=1460483638442, value=0.18537115

                                      0.1929401133528662,0.21770476919925866,0.20786868010031592
```

Figure 7: Document - Topic Output from HBase

4.3.3.2 Topic-word matrix

We will create a separate HBase table for storing information about top words in each topic. This topic-word table will contain two column families: one for tweets and another one for webpages. Each column family will contain 3 columns: topic label, word list and probability list. Label for each topic will be stored in the column "Topic Label". This will be used by the frontend team in displaying facets. The column "Word List" will contain the top 10 words for each topic. The corresponding probabilities for each word in the word list will be stored in the column "Probability List". The number of rows in the table will be equivalent to the number of topics in each collection. The row key for this table will be "Collection ID_Topic ID".

Row Key	**Topics**			
	Label (String)	**Collection_ID (String)**	**Words (CSV String)**	**Probabilities (CSV String)**

Table 6: Topic-Word HBase Schema

Example Output

Row Key	**Topics**			
	Label	**Collection_ID**	**Words**	**Probabilities**

	Shooting	700	Shooting, Reporter, heartbroken, condolence...	0.0497,0.0491,0.047..
	Victim	700	Victim, remembrance, violence,..	0.021,0.009,0.005..

Table 7: Sample Topic-Word Output

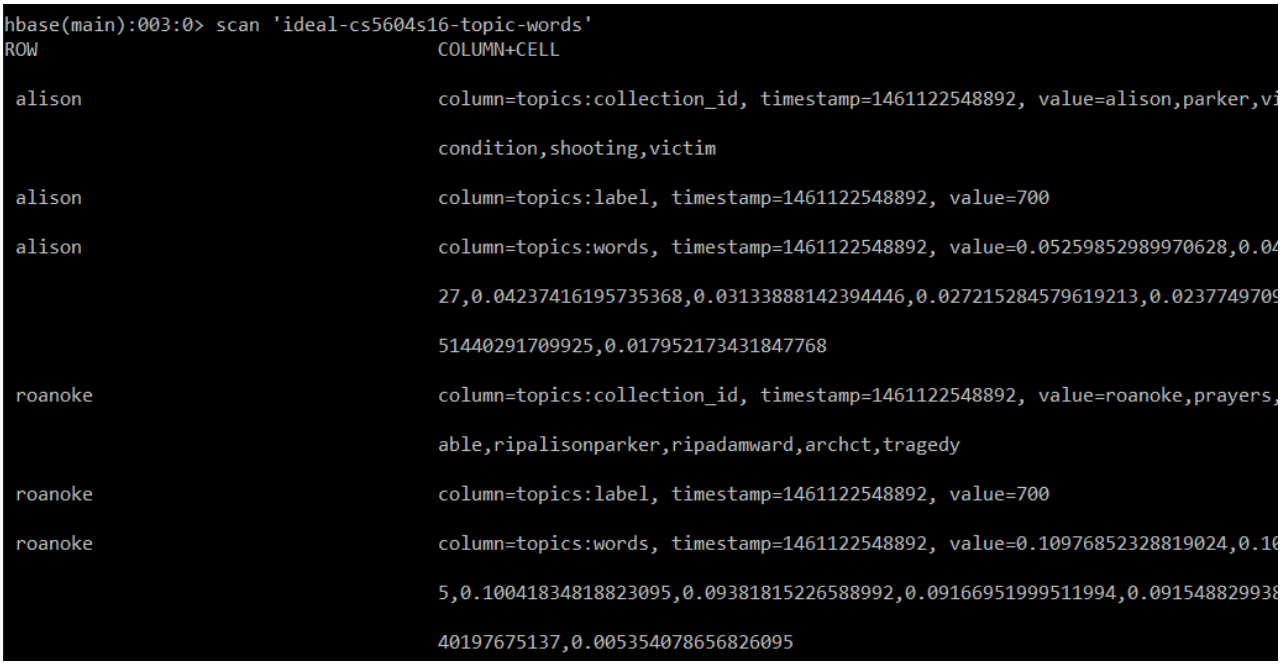


Figure 8: Topic-Word Output from HBase

4.4. Experiments and Results

4.4.1 Cleaned Collections

Using topic model described in the report, we ran experiments on IDEAL tweet and web page collections. We got 6 small cleaned tweet collections and 1 cleaned web page collection from the Collection Management team. The details of these collections are given in table 8 below.

Collection Name	Has Tweets?	Has Web pages?
WDBJ Shooting	Yes	No
NAACP Bombing	Yes	No
4th of July	Yes	No
Germanwings	Yes	No

Houston	Yes	No
Obamacare	Yes	Yes

Table 8: Cleaned collections

4.4.2 Experiments on Collections

We have implemented LDA in Scala using Spark MLlib. We have written the program in such a way that it could be launched in both Spark standalone mode as well as cluster mode. Spark standalone mode supports 100 to 200 iterations of LDA with the available computational resources. We tested the code on cluster as well, but when trying to increase the number of iterations to more than 500, it broke the Spark Context. Thus we recorded all our results in standalone mode. In table 9, we have recorded the time it took to run LDA for each collection. Figure 9 represents the time vs. size for each collection.

Tweet Collection	Size	Time (in seconds)
Houston Flood	14813	860
4thofJuly	31743	2220
NAACP Bombing	33865	2094
Germanwings	90040	12253

Table 9: Running time for tweets

Webpage Collection	Size	Time (in Seconds)
Obamacare	22393	1615

Table 10: Running time for webpage collection

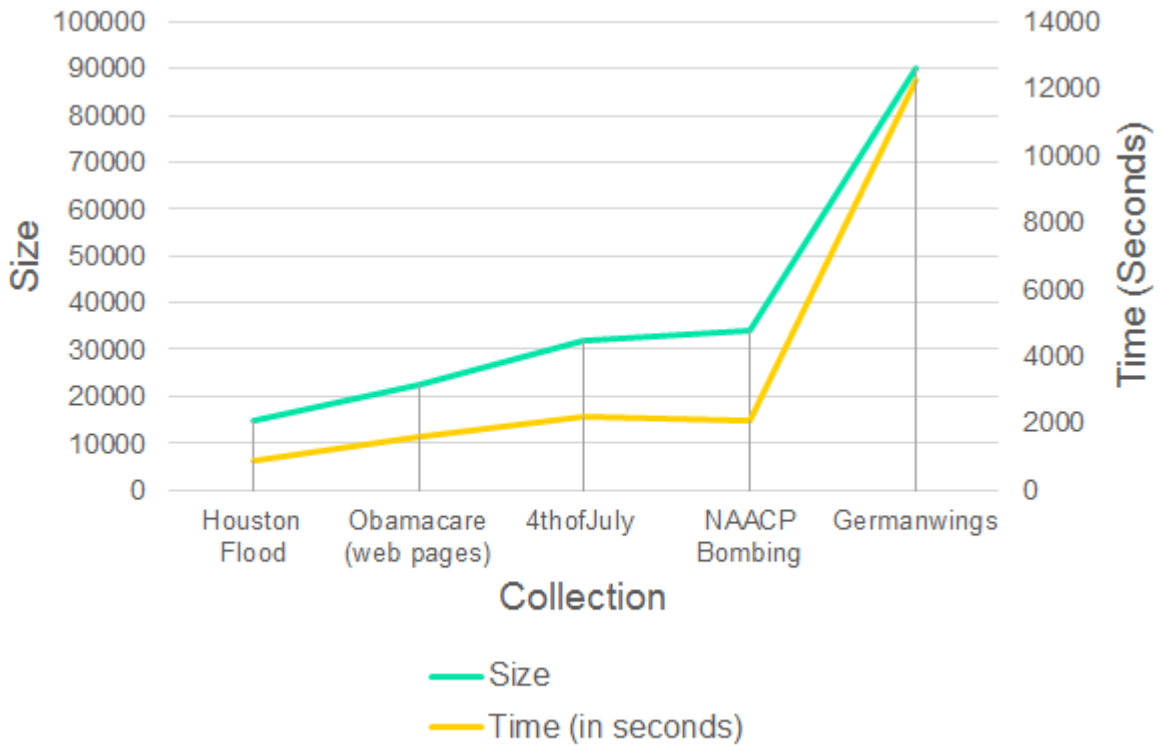


Figure 9: Time, Size plot for each collection

4.4.3 Results for Tweet Collections

On each of the cleaned tweet collection we ran LDA for 100 iterations to identify 5 topics. Table 11 shows the topics that we identified for each of these collection through our automated topic labeling technique. Top 10 words for all these topics are given in the Appendix 10.4 section.

Collection Name	Topics
WDBJ Shooting	journalists, shooting, victims, Roanoke, virginia
NAACP Bombing	naacp, naacpbombing, terrorism, michellemalkin, deray
4thofJuly	american, independenceday, fireworks, happy, america
Germanwings	germanwings, ripley, world, copilot, crash
Houston Flood	houstonflood, texas, texasflood, flood, billbishopkhou

Table 11: Topics for tweet collections

4.4.3 Comparison of Tweet and Web page results

For the IDEAL Obamacare collection, CM team provided both the cleaned tweets and cleaned webpages separately. To identify relevant topics, individually we ran LDA on Obamacare tweets and web page collections. For the tweet collection, we ran LDA for 100 iterations to identify 5 topics. For the webpage collection, we ran LDA for 200 iterations to identify 4 topics. The identified topics are given in Table 12.

Collection Name	Webpage_Topcis	Tweet_Topics
Obamacare	insurance, plans, companies, court	obamacare, uninsured, health, arrived, pjnet

Table 12: Topics in Obamacare collection

Table 13 shows the top words for each of the topics in tweet and webpage collections. Note that the topics are quite comparable in tweet and webpage results. For instance, Topic 1 is sharing words like health, insurance and people in both tweet and webpage results.

Topics	Tweet_Topic_Words	Webpage_Topic_Words
Topic 1	health, insurance, people, healthcare, medicaid, congress…	plans, fixed, indemnity, insurance, people, judge, health, administration…
Topic 2	uninsured, barackobama, americans, thanks, realdonaldtrump, president …	percent, increases, federal, health, insurers, affordable, state, claims ..
Topic 3	obamacare, repeal, scottwalker, prolife, little, supports, sisters…	court, religious, mandate, supreme, schools, baptist, organizations, christian…
Topic 4	arrived, abcnews, antichrist, doomsday, cometcoreinc, comet, republicans …	companies, books, playlist, cancer, medical, health, medicine, problem…
Topic 5	pjnet, cruzcrew, benghazi, teaparty, tedcruz, healthcare, uniteblue …	

Table 13: Top words in Obamacare collection

4.4. LDA Evaluation

This section outlines our evaluation strategies for the topics.

4.4.1 Choosing number of topics

One of the input parameters for LDA is the number of latent topics in the corpus. If we a choose large number of topics, LDA will forcefully separate the documents into different topics, even though they discuss a single topic. On the other hand, if we choose small number of topics as input, the documents that are not likely to be in the same group can be clustered together. The decision for the number of topics thus depends on the size and the nature of the collection.

For the shooting collection, we executed LDA for different numbers of topics and manually verified the quality of the results. After executing LDA for different numbers of topics, for the small collection, we found that if we choose more than 5 topics, many of the words are shared between topics.

4.4.2 Choosing number of iterations

Number of iterations to run LDA for is another important input parameter. If we stop LDA after only a few iterations, the quality of the results is poor. Choosing the correct number of iterations depends on the size of the collection. By running LDA for different numbers of iterations we found that for 100 iterations we get good quality results. If we run for a smaller number of iterations, different topics tend to share many words. In the Section 10.2, we have provided the Topic-Word results for 20 iterations and 100 iterations. In the result for 20 iterations a lot of words are shared between topics but for 100 iterations the words are clearly distinctive. So, based on the size of the collection, we pick the number of iterations.

4.4.3 Evaluation of output

It is a standard practice to manually evaluate the results of the topic model [12]. In our preliminary evaluation, we manually evaluated the results by checking for word intrusion. Word intrusion is a measure to check for the coherence of topics. If different topics have clearly separable words that explain the corresponding topics well, then the result of the LDA is considered to be good. Since we ran LDA on a single collection which talks about the WDBJ shooting, all the topics in our result are related to that specific shooting incident and they do not contain very distinctive words. Below we present the results of the user study that we performed in CS 5604 class. The study consisted on users answering a questionnaire (attached in Appendix 10.5). At the start of the study the participants were distributed a plain sheet of paper and the questionnaire was displayed on the screen. The participants were given background and instructions about the study and then were asked to write answers to each of the questions in the questionnaire. The entire study lasted for about 10 minutes. There were total six questions split into two parts. The first part consisted of five questions requiring the participant to pick a word that did not belong to the group. The words were actually the top ten words for each of the five topics extracted by our topic model. The sixth question asked the participant to give a score on 1 to 10 on the quality of words describing the WBDJ shooting. Total 15 participants participated in the study. The table below describes the votes for each of the words for every topic. The words highlighted in red got the most votes and can be considered as

intruder words. Except topic 4, majority of the participants were able to agree on the intruder word for each topic. This represents the coherence of the topics.

Topic 1	**Topic 2**	**Topic 3**	**Topic 4**	**Topic 5**
word/ votes	word /votes	word /votes	word /votes	word/votes
Virginia / 4	Alison / 6	Roanoke / 1	Shooting / 0	Victims / 0
Journalists / 1	Shooting / 0	Prayers / 0	Vicki / 7	Shooting / 0
Shooting / 0	Video / 4	Families / 1	Gardner / 4	Gunman / 0
Newscast / 3	Parker / 5	Friends / 1	Survivor / 0	Husband / 9
Heartbroken / 2	Reporter / 0	Unbelievable / 2	Condition / 4	Blame / 2
Condolence / 0	Police / 0	Ripalisonparker /4	People / 2	Foxnews /4
Tribute / 1	Watch / 5	Ripadamward / 7	Victim / 0	westandwithwdbj/ 3
Everytown / 11	Christiansburg / 2	Archct / 11	Today / 8	Brianstelter / 4
Community / 0	Moment / 9	Shooting / 0	Thoughts / 4	Tragic / 0
Support / 2	Fatal / 0	Suspect / 0	Virginia / 2	Onair / 6

Table 14: Evaluation results

For the sixth question, one participant did not answer the question. Out of the remaining 14 participants, 12 participants gave a score of 8 or above and 2 participants gave a score of 7. Figure 10 represents the number of votes vs scores for 6th question.

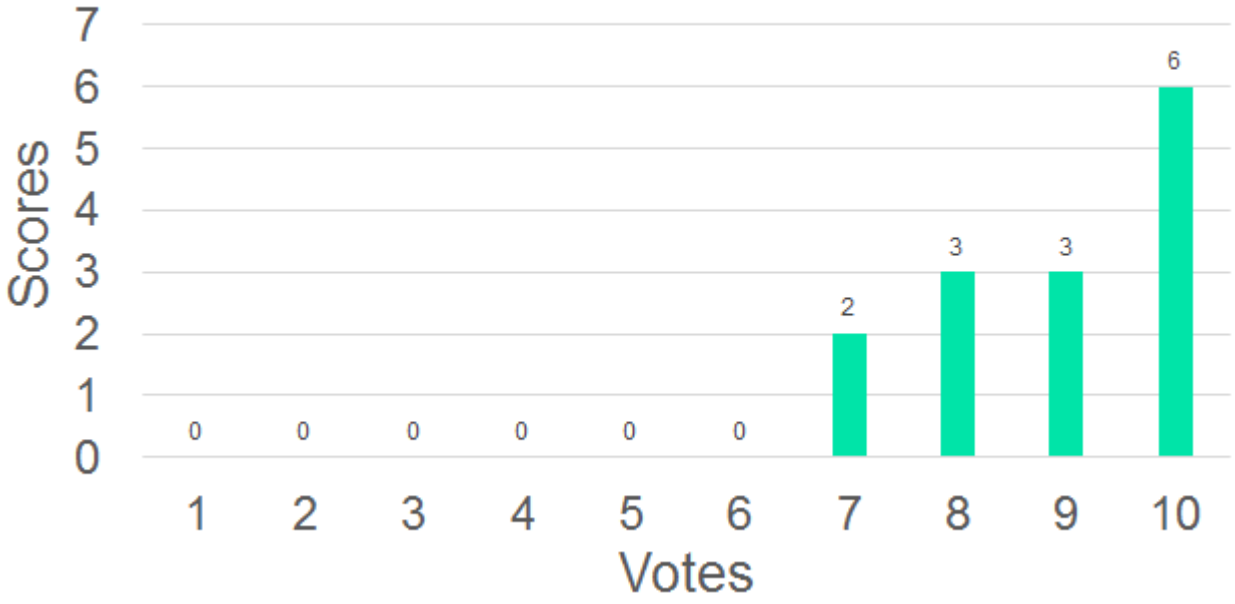


Figure 10: Evaluation result

5. User Manual

This section contains Apache Spark installation tutorials and basics of LDA in Spark. This section will help new users get kick-started with programming in Spark and using the MLlib library especially the LDA API. This section will also be useful to users for understanding our LDA code provided in the Developer's Manual (Section 4) and run LDA on their own datasets.

5.1 Spark Installation

There are two ways to get kickstarted with writing programs using Apache Spark. First, install Apache Spark on your personal computer and second, install Cloudera virtual machine in a virtualization software such as VirtualBox or VMWare which comes pre-installed with Apache Spark. We illustrate both ways below.

5.1.1 Spark on PC

Spark is Hadoop's sub-project. Therefore, it is better to install Spark into a Linux based system. The following steps show how to install Apache Spark.

Step 1: Verifying Java Installation

Java installation is one of the mandatory things in installing Spark. Try the following command to verify the JAVA version.

```
$java -version
```

If Java is already installed on your system, you get to see the following response –

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

In case you do not have Java installed on your system, then Install Java before proceeding to the next step.

Step 2: Verifying Scala installation

Spark has API's in Java, Python and Scala, but use of Scala is preferred. So let us verify Scala installation using the following command.

```
$scala -version
```

If Scala is already installed on your system, you get to see the following response –

```
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
```

In case you don't have Scala installed on your system, then proceed to the next step for Scala installation.

Step 3: Downloading Scala

Download the latest version of Scala by visit the following link http://www.scala-lang.org/download/ . After downloading, you will find the Scala tar file in the download folder.

Step 4: Installing Scala

Follow the steps given below for installing Scala.

Extract the Scala tar file

Type the following command for extracting the Scala tar file.

```
$ tar xvf scala-2.11.6.tgz
```

Move Scala software files

Use the following commands to move the Scala software files to a suitable directory (/usr/local/scala).

```
$ su –
Password:
# cd /home/Hadoop/Downloads/
# mv scala-2.11.6 /usr/local/scala
# exit
```

Set PATH for Scala

Use the following command for setting PATH for Scala.

```
$ export PATH = $PATH:/usr/local/scala/bin
```

Verifying Scala Installation

After installation, it is better to verify it. Use the following command for verifying Scala installation.

```
$scala -version
```

If Scala is already installed on your system, you get to see the following response –

```
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
```

Step 5: Downloading Apache Spark

Download the latest version of Spark by visiting the following link https://spark.apache.org/downloads.html. For this tutorial, we are using spark-1.3.1-bin-Hadoop 2.6 version. After downloading it, you will find the Spark tar file in the download folder.

Step 6: Installing Spark

Follow the steps given below for installing Spark.

Extracting Spark tar

The following command for extracting the spark tar file.
$ tar xvf spark-1.3.1-bin-hadoop2.6.tgz

Moving Spark software files

The following commands for moving the Spark software files to respective directory (/usr/local/spark).

```
$ su -
Password:

# cd /home/Hadoop/Downloads/
# mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark
# exit
```

Setting up the environment for Spark

Add the following line to ~/.bashrc file. That adds the location, where the spark software files are located to the PATH variable.

```
export PATH = $PATH:/usr/local/spark/bin
```

Use the following command for sourcing the ~/.bashrc file.

```
$ source ~/.bashrc
```

Step 7: Verifying the Spark Installation

Write the following command for opening the Spark shell.

```
$spark-shell
```

If Spark is installed successfully then you will find the following output.

```
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/06/04 15:25:22 INFO SecurityManager: Changing view acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: Changing modify acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: SecurityManager: authentication disabled;
   ui acls disabled; users with view permissions: Set(hadoop); users with modify permissions:
Set(hadoop)
15/06/04 15:25:22 INFO HttpServer: Starting HTTP Server
15/06/04 15:25:23 INFO Utils: Successfully started service 'HTTP class server' on port 43292.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.0
```

```
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71)
Type in expressions to have them evaluated.
Spark context available as sc
scala>
```

5.1.2 Spark on Cloudera virtual machine

1. Go to http://hadoop.dlib.vt.edu/cloudera_vm/ . Download what you want for KVM, VirtualBox, or VMWare.

2. Run VirtualBox and then Run "Import Appliance…" in the File menu

3. Find "cloudera-quickstart-vm-5.3.*-virtualbox" in your unzipped virtual machinefolder and click "open"

4. Click 'continue'

5. Click 'Import'

6. Click 'Start'

Once you are able to successfully deploy cloudera on VirtualBox or the like, it is easy to get started with Scala programs.

Open a terminal in the virtual machine and type this:

```
$ spark-shell --master yarn-client
```

Your should start seeing output like below on your screen, and get a Scala shell back.

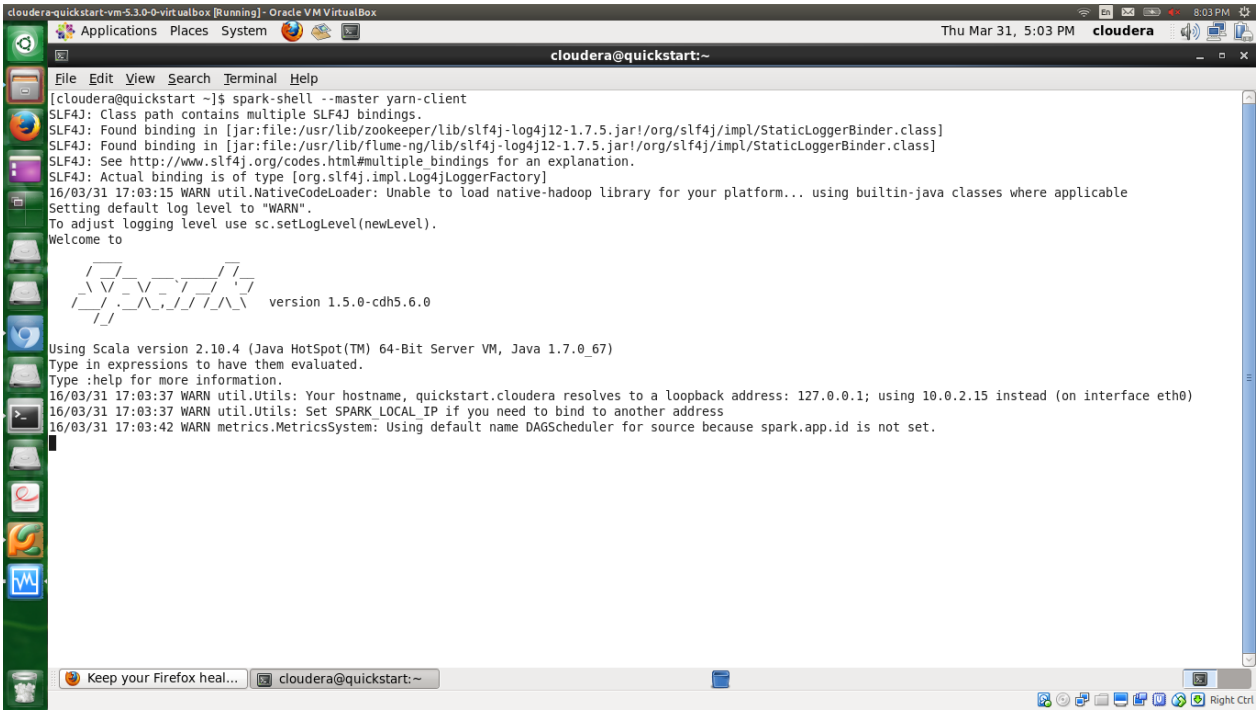


Figure 11: Starting Spark Shell

If you are unhappy with the version of Spark available on Cloudera, it is possible to upgrade the version by this command. Remember to install a Spark version >= 1.3, because LDA API was first introduced only in version 1.3.

```
$ sudo yum install spark-core spark-master spark-worker spark-history-server spark-python
```

The above command will work with RHEL compatible systems. Cloudera uses CentOS which is a RHEL compatible system. But if you want to upgrade your Spark version on Ubuntu or Debian systems, below is the command to use.

```
$ sudo apt-get install spark-core spark-master spark-worker spark-history-server spark-python
```

5.2 Apache Spark

Below we provide a brief introduction to Apache Spark and its core concepts and later we give an introduction to the Spark's Machine Learning library (MLlib).

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types

of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.
Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Apache Spark - Core programming

Spark Core is the base of the whole project. It provides distributed task dispatching, scheduling, and basic I/O functionalities. Spark uses a specialized fundamental data structure known as RDD (Resilient Distributed Datasets) that is a logical collection of data partitioned across machines. RDDs can be created in two ways; one is by referencing datasets in external storage systems and second is by applying transformations (e.g; map, filter, reducer, join) on existing RDDs.

The RDD abstraction is exposed through a language-integrated API. This simplifies programming complexity because the way applications manipulate RDDs is similar to manipulating local collections of data.

Spark Shell

Spark provides an interactive shell − a powerful tool to analyze data interactively. It is available in either Scala or Python. Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD). RDDs can be created from Hadoop Input Formats (such as HDFS files) or by transforming other RDDs.

Open Spark Shell

The following command is used to open the Spark shell.

```
$ spark-shell
```

Create simple RDD

Let us create a simple RDD from the text file. Use the following command to create a simple RDD.

```
scala> val inputfile = sc.textFile("input.txt")
```

The output for the above command is

```
inputfile: org.apache.spark.rdd.RDD[String] = input.txt MappedRDD[1] at textFile at
<console>:12
```

Machine Learning Library (MLlib)

We use Spark MLlib's API for LDA for topic modeling. Below we give a brief introduction of MLlib and then introduce the LDA API.

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.

It divides into two packages:

`spark.mllib` contains the original API built on top of `RDDs`.
`spark.ml` provides higher-level API built on top of `DataFrames` for constructing ML pipelines.
Using `spark.ml` is recommended because with DataFrames the API is more versatile and flexible.

Latent Dirichlet allocation (LDA)

Latent Dirichlet allocation (LDA) is a topic model which infers topics from a collection of text documents. LDA can be thought of as a clustering algorithm as follows:

Topics correspond to cluster centers, and documents correspond to examples (rows) in a dataset.
Topics and documents both exist in a feature space, where feature vectors are vectors of word counts (bag of words).

1. Rather than estimating a clustering using a traditional distance, LDA uses a function based on a statistical model of how text documents are generated.
2. LDA supports different inference algorithms via the `setOptimizer` function. `EMLDAOptimizer` learns clustering using expectation-maximization on the likelihood function and yields comprehensive results, while `OnlineLDAOptimizer` uses iterative mini-batch sampling for online variational inference and is generally memory friendly.
3. LDA takes in a collection of documents as vectors of word counts and the following parameters (set using the builder pattern):

- k: Number of topics (i.e., cluster centers)
- `optimizer`: Optimizer to use for learning the LDA model, either `EMLDAOptimizer` or `OnlineLDAOptimizer`
- `docConcentration`: Dirichlet parameter for prior over documents' distributions over topics. Larger values encourage smoother inferred distributions.
- `topicConcentration`: Dirichlet parameter for prior over topics' distributions over terms (words). Larger values encourage smoother inferred distributions.
- `maxIterations`: Limit on the number of iterations.
- `checkpointInterval`: If using checkpointing (set in the Spark configuration), this parameter specifies the frequency with which checkpoints will be created. If maxIterations is large, using checkpointing can help reduce shuffle file sizes on disk and help with failure recovery.

All of spark.mllib's LDA models support:

- `describeTopics`: Returns topics as arrays of most important terms and term weights
- `topicsMatrix`: Returns a vocabSize by k matrix where each column is a topic

Note: LDA is still an experimental feature under active development in Spark. As a result, certain features are only available in one of the two optimizers / models generated by the optimizer. Currently, a distributed model can be converted into a local model, but not vice-versa.

Now we present a sample of our code for LDA on the tweets dataset. Full code with all dependencies is uploaded on VTechWorks.

```
import scala.collection.mutable
import org.apache.spark.mllib.clustering.LDA
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.rdd.RDD

val corpus: RDD[String] = sc.textFile("tweets_data")

// Split each document into a sequence of terms (words)
val tokenized: RDD[Seq[String]] = corpus.map(_.toLowerCase.split("\\s")).map(_.filter(_.length
> 4).filter(_.forall(java.lang.Character.isLetter)))

// Choose the vocabulary
//   termCounts: Sorted list of (term, termCount) pairs
val termCounts: Array[(String, Long)] = tokenized.flatMap(_.map(_ -> 1L)).reduceByKey(_ +
_).collect().sortBy(-_._2)

//   vocabArray: Chosen vocab (removing common terms)
val numStopwords = 25
val vocabArray: Array[String] = termCounts.takeRight(termCounts.size - numStopwords).map(_._1)

//   vocab: Map term -> term index
val vocab: Map[String, Int] = vocabArray.zipWithIndex.toMap

// Convert documents into term count vectors
val documents: RDD[(Long, Vector)] =
  tokenized.zipWithIndex.map { case (tokens, id) =>
    val counts = new mutable.HashMap[Int, Double]()
```

```
    tokens.foreach { term =>
      if (vocab.contains(term)) {
        val idx = vocab(term)
        counts(idx) = counts.getOrElse(idx, 0.0) + 1.0
      }
    }
    (id, Vectors.sparse(vocab.size, counts.toSeq))
  }

// Set LDA parameters
val numTopics = 5
val lda = new LDA().setK(numTopics).setMaxIterations(20)
val ldaModel = lda.run(documents)
val avgLogLikelihood = ldaModel.logLikelihood / documents.count()

// Print topics, showing top-weighted 10 terms for each topic.
val topicIndices = ldaModel.describeTopics(maxTermsPerTopic = 10)
topicIndices.foreach { case (terms, termWeights) =>
  println("TOPIC:")
  terms.zip(termWeights).foreach { case (term, weight) =>
    println(s"${vocabArray(term.toInt)}\t$weight")
  }
  println()
}
```

The above sample code can be executed in Spark standalone mode using Spark shell. Also, by adding class and main function to the code, you can run it in Spark cluster mode. To run in cluster, first we need to create a jar file. We can use SBT to create a jar file from Scala code.

SBT (Scala build tool)

First, store the Scala code in a directory. And in the same directory create a config.sbt. This file should contain all dependencies of the Scala program. The class name of the Scala program should be mentioned in the name field of SBT.

```
name := "LDAClass"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies ++= Seq(
"org.apache.spark" %% "spark-core" % "1.0.0",
"org.apache.spark" %% "spark-mllib" % "1.5.0"
)
```

Now the Scala program could be executed on the Spark cluster using `spark submit` command. Remember to pass all the required parameters for the program to `spark submit.`

```
spark-submit --master spark://node1.dlrl:7077 --conf "spark.broadcast.blockSize=64m" --conf
"spark.reducer.maxSizeInFlight=2g" --conf "spark.shuffle.file.buffer=512k" --conf
"spark.shuffle.io.numConnectionsPerPeer=4"  --class lda_web --jars lda-web22_2.10-1.0.jar lda-
web22_2.10-1.0.jar hdfs://nameservice1/user/cs5604s16_to/z686-1 z686-1-topicwords.txt
hdfs://nameservice1/user/cs5604s16_to/z686-1-doctopicfile
hdfs://nameservice1/user/cs5604s16_to/z686-1-doctopicfilecombined z686-1-doctopicfinal.txt 5
200 686
```

6. Lessons Learnt

Throughout the course of this project we faced numerous challenges and developed solutions to overcome those challenges. This section details those aspects of our work for future reference.

1) In the initial testing stages of our algorithm we observed a lot of repeating words in topics. This was because we were running the algorithm for only 20 iterations. The model was only 'half-baked'. Increasing the number of iterations effortlessly alleviated the problem.

2) Sometimes, we observed words in the topic that did not make semantic sense. For instance we observed 'lgtlgtg' in the 'Houston Floods' (541) small tweets collection. We include such type of words in custom stop-words list by manual inspection.

3) Is is very important to fine-tune the number of topics based on the data collection as detailed in the Evaluation section. At times, when the topics were not very discriminative we decreased the number of topics which resulted in overall better representation. At other times, when the topics were clear and well defined we increased the number of topics to see if there was possibility for more fine-grained topic representation.
4) It is our empirical observation that when the amount of data is more, example webpage data, the topics are well defined and clearly separable from each other.

7. Conclusion and Future work

In contribution to the IDEAL project for CS 5604 Spring 2016, we implemented Latent Dirichlet Allocation (LDA) for topic modeling. We performed topic analysis on tweets and web page collections. Using our topic model, we extracted meaningful topics from all tweet and web page collections. Our evaluation results confirm the good quality of our topic model. We also devised an automated approach to label the topics that we obtained from LDA. In this project report, we have given complete information on background, implementation and evaluation details of our approach. We have developed an extensive developer's manual to facilitate someone to extend our work in future. User's manual will help the readers deploy and test our work.

Future Work:

The topic models for tweets and web page collections could be implemented in a couple of ways:

1. Independent topic model for tweets and web pages
2. Develop a topic model for tweets and use the same to analyze web pages
3. Develop a topic model for web pages and use the same to analyze tweets
4. Treat the tweets and web pages together to develop a single topic model

We have developed independent topic models for tweets and web pages, but in future, we can also try implementing other ways to come up with the best approach.

For this project, we have implemented the basic LDA which works best for static data collections. In literature, there are also other variants of LDA like Online LDA / hierarchical LDA which is suitable for the streaming data / the collections which get constantly updated. In future, our work could be extended in this direction to implement Online LDA algorithm.

We have also developed an automated technique to label the topics. Our approach uses the top words for each topic to come up with labels. In future, we can also label the topics using tweet's/ webpage's metadata information like hashtags, titles etc. In [13] Lau et. al have discussed this approach in detail. We can also try the possibilities of other sophisticated Machine Learning techniques like "Word to Vec".

8. Acknowledgements

We would like to thank Prof. Edward Fox for helping us in the successful completion of the project. We also thank the Graduate Research Assistant's Mr. Sunshin Lee and Mr. Mohamed Magdy Gharib Farag for their useful discussions and advice. We also like to express sincere gratitude to all the students of CS 5604 Spring class in extending their best cooperation in the successful completion of the project. We also acknowledge and thank NSF in supporting the Integrated Digital Event Archiving

and Library (IDEAL) project with the grant IIS - 1319578. Last but not least, we acknowledge the success of the problem based learning approach of CS 5604 class.

9. References

[1] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993-1022.

[2] Sarunya Pumma and Xiaoyang Liu, Extracting Topics from Tweets and Webpages for IDEAL: LDA project in CS 5604, Spring 2015. http://vtechworks.lib.vt.edu/bitstream/handle/10919/52343/LDAReport.pdf?sequence=6&isAllowed=y

[3] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. Introduction to Information Retrieval. New York: Cambridge UP, 2008.

[4] Wei, Xing, and W. Bruce Croft. 2006. LDA-based document models for ad-hoc retrieval. In Proc. SIGIR, pp. 178–185. ACM Press.

[5] Wikipedia LDA: http://en.wikipedia.org/wiki/ Latent_Dirichlet_allocation

[6] Tom White. 2015. *Hadoop: The Definitive Guide*, 4th Edition. O'Reilly Media, Inc.

[7] Lam, Chuck. *Hadoop in Action*. Greenwich, CT: Manning Publications, 2011.

[8] Ryza, Sandy, Uri Laserson, Sean Owen, and Josh Wills. *Advanced Analytics with Spark*.

[9] Solr Manual. http://lucene.apache.org/solr/quickstart.html

[10] Ilker Yildirim. 2012. *Bayesian Inference: Gibbs Sampling*. http://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm

[11] "Topic Modeling with LDA: MLlib Meets GraphX." *Databricks*. 25 Mar. 2015. Web. 11 Mar. 2016. https://databricks.com/blog/2015/03/25/topic-modeling-with-lda-mllib-meets-graphx.html

[12] Chang, Jonathan, et al. "Reading tea leaves: How humans interpret topic models." Advances in neural information processing systems. 2009.

[13] Lau, Jey Han, et al. "Automatic labelling of topic models." *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011.

[14] Topic analysis for NIPS 2014 papers. https://cs.stanford.edu/people/karpathy/nips2014/

[15] The scala programming language. http://www.scala-lang.org/

10. Appendix

10.1 Sample words after pre-processing

We cleaned the data for the small 'Shooting' collection provided by the Collection Management team. We split text into terms (words) and then removed short terms with < 4 characters and the most common 25 terms (as "stopwords"). A sample of the resulting data has been provided below. The text is now just a collection of words (bag of words) which can be ingested by our LDA algorithm.

```
reporter ,heartbroken ,condolence ,survivor ,police ,watch ,suspect ,everytown
,westandwithwdbj ,moment ,condition ,christiansburg ,youtube ,people ,fatal ,community
,brownjenjen ,photographer ,support ,victim ,killed ,obama ,today ,president ,control ,slain
,thoughts ,terrorism ,deaths ,update ,emotional ,suicide ,spoke ,colleagues ,debate ,remember
,family ,conference ,remembered ,wards ,killing ,footage ,reporters ,autoplay ,stable
,breaking ,recovering ,hospital ,attack ,silence ,harrowing ,ignites ,television ,cbsnews
,employees ,yoware ,father ,during ,onair ,brianstelter ,tragic ,could ,office ,house
,describes ,released ,interview ,gunshot ,cnnbrk ,wounds ,medical ,examiners ,suffered
,mashable ,investigating ,waffle ,filmed ,coworkers ,virginiashooting ,shocking
,londonandreams ,mourn ,guardian ,upgraded ,special ,county ,looking ,todays ,deadly ,eonline
,brighten ,lockdown ,finale ,anyone ,commits ,mental ,share ,gunwestandwithwdbj ,liked
,violence ,bridgewater ,thursday ,attempted ,deputy ,story ,cameraman ,shooter...
```

10.2 Topic-Word output for different input parameters

Number of Topics: 5 Number of words: 7 Number of iterations: 20

Topic 1		Topic 2		Topic 3	
condolence	0.0123	reporter	0.0114	reporter	0.0120
heartbroken	0.0122	survivor	0.0113	condolence	0.0118
police	0.0122	condolence	0.0112	heartbroken	0.0117
survivor	0.0119	heartbroken	0.0110	survivor	0.0108
reporter	0.0119	watch	0.0108	police	0.0104
watch	0.0100	police	0.0092	Watch	0.0102
moment	0.0095	moment	0.0088	westandwithwdbj	0.0094

Topic 4		Topic 5	
reporter	0.0118	heartbroken	0.0125
heartbroken	0.0113	reporter	0.0122
condolence	0.0113	condolence	0.0118
police	0.0111	survivor	0.0111

survivor 0.0104 Suspect 0.0096 westandwithwdbj 0.0084	police 0.0103 watch 0.0090 condition 0.0089

Num topics = 3, Num iterations = 100

Topic 1	**Topic 2**	**Topic 3**
Brownjenjen 0.0329	Condition 0.0417	reporter 0.0581
Emotional 0.0250	people 0.0389	Westandwithwdbj 0.0413
debate 0.0244	victim 0.0329	christiansburg 0.0391
Conference 0.0230	today 0.0315	fatal 0.0337
autoplay 0.0227	thoughts 0.0297	photographer 0.0322
Harrowing 0.0213	Remember 0.0244	watch 0.0300
Ignites 0.0213	family 0.0241	remembered 0.0226
youtube 0.0160	stable 0.0230	wards 0.0225
Brianstelter 0.0154	Recovering 0.0224	breaking 0.0217
control 0.0142	cbsnews 0.0216	silence 0.0213

Num topics = 5, Num iterations = 100

Topic 1	**Topic 2**	**Topic 3**
police 0.0550	Suspect 0.0457	youtube 0.0388
Christiansburg 0.0416	people 0.0393	slain 0.0298
Brownjenjen 0.0339	victim 0.0332	brianstelter 0.0199
Update 0.0274	Remember 0.0247	onair 0.0198
suicide 0.0259	footage 0.0236	office 0.0185
debate 0.0251	killing 0.0235	gunshot 0.0177
Autoplay 0.0233	Reporters 0.0234	cnnbrk 0.0170
Breaking 0.0229	cbsnews 0.0218	wounds 0.0169
hospital 0.0225	Employees 0.0213	medical 0.0169
ignites 0.0220	tragic 0.0200	suffered 0.0168

Topic 4	**Topic 5**
heartbroken 0.0497	reporter 0.0588
condolence 0.0491	watch 0.0477
survivor 0.0410	Westandwithwdbj 0.0425
everytown 0.0380	moment 0.0416
community 0.0318	condition 0.0396
obama 0.0296	fatal 0.0341
president 0.0289	photographer 0.0326
control 0.0288	today 0.0306
Support 0.0283	thoughts 0.0289
terrorism 0.0256	colleagues 0.0235

10.3 Document - Topic distribution for WDBJ Shooting collection

The topic distribution for the first 10 tweets in the WDBJ Shooting collection is given below:

Tweet ID	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
1	0.2084	0.1792	0.2057	0.2085	0.1980
2	0.2050	0.1915	0.2114	0.1895	0.2050
3	0.2130	0.1795	0.2041	0.2186	0.1845
4	0.1915	0.1971	0.2158	0.1863	0.2091
5	0.1892	0.2080	0.1914	0.1921	0.2191
6	0.1752	0.2014	0.2028	0.2051	0.2152
7	0.2083	0.1889	0.2020	0.1848	0.2158
8	0.2053	0.1967	0.1454	0.2150	0.2373
9	0.1797	0.2134	0.2126	0.1858	0.2082
10	0.1884	0.2013	0.2061	0.2184	0.1855

Table 15: Tweet - Topic distribution for WDBJ Shooting collection

10.4 Top words for each topic

After providing samples of output below we list exhaustively all topic words for all collections we modeled.

Tweets:

541 – NAACP Bombing

```
naacpbombing,suspect,sketch,msnbc,paris,photo,attack,michellemalkin,covering,terrorist

media,black,naacpbombing,coverage,zellieimani,still,massacre,jrehling,deaths,apologizing

naacpbombing,white,because,please,twitter,havent,heard,update,terrorism,bipartisanism

naacp,naacpbombing,bombing,colorado,stories,springs,brownjenjen,another,under,cannot

Naacpbombing,deray,person,looking,respensable,people,shaunking,blacklivesmatter,there,charlie
hebdo
```

602 – Germanwings

germanwings,crash,lubitz,copiloto,andreas,victims,vuelo,actualidadrt,reuters,bomba
germanwings,vctimas,copilot,accidente,plane,french,believe,crash,ripley,sociedad
germanwings,pilot,flight,crash,playing,click,believe,ripley,nowplaying,copilot
germanwings,ripley,believe,world,charliehebdo,lives,garissa,stood,showed,whole
germanwings,lufthansa,bigareveal,pllfinale,believe,ripley,charlesisa,vampsvip,prince,funeral

668 - Houston flood

houstonflood,hounews,please,river,flooding,flooded,still,those,photo,affected
flood,houston,houstonflood,victims,houwx,today,there,shelter,myfoxhouston,animal
houstonflood,billbishopkhou,drive,follow,flooding,people,found,retweet,ground,cattle
houstonflood,texas,disaster,relief,bipartisanism,federal,weather,during,against,everyone
houstonflood,texasflood,insurance,video,great,cambodia,getstupid,houston,online,watched

686 - Obamacare

obamacare,repeal,would,scottwalker,replace,prolife,every,little,supports,sisters
obamacare,uninsured,barackobama,americans,first,thanks,realdonaldtrump,costs,single,president
obamacare,health,insurance,people,healthcare,million,since,medicaid,before,congress
obamacare,arrived,abcnews,antichrist,doomsday,dajjal,smartphone,cometcoreinc,comet,republicans
obamacare,pjnet,obama,cruzcrew,support,benghazi,teaparty,tedcruz,healthcare,uniteblue

694 – 4th of July

american,weekend,clothing,patriotic,fourth,psswstyles,fourthofjuly,chicago,johnspatricc,white

```
independenceday,party,miley,cyrus,musicnewsvocal,family,friends,country,happy,giveaway

fireworks,photo,firework,people,funny,beach,never,music,followme,youre

happy,independence,celebrate,freedom,indepenceday,lebronjames,teamlebron,freeageny,heatnation,
summer

america,watch,askanythingchat,ddlovato,right,everyone,saturdaynightonline,wonairromeo,great,bi
rthday
```

Webpages:

686 - Obamacare

```
plans,fixed,indemnity,insurance,people,judge,health,administration,purchase,central

insurance,percent,increases,federal,health,insurers,affordable,state,claims,increase

court,religious,mandate,supreme,schools,baptist,organizations,christian,support,statement

companies,books,playlist,cancer,medical,health,federal,effectively,medicine,problem
```

10.5 User Study - Topic Modeling

For questions 1 to 5 below please select the words which do not belong to the group.

1. Virginia, journalists, shooting, newscast, heartbroken, condolence, tribute, everytown, community, support

2. Alison, shooting, video, parker, Reporter, police, Watch, Christiansburg, moment, fatal

3. Roanoke, Prayers, families, friends, Unbelievable, ripalisonparker, ripadamward, archct, shooting, suspect

4. Shooting, vicki, gardner, survivor, condition, people, victim, Today, thoughts, virginia

5. Victims, shooting, gunman, husband, blame, foxnews, westandwithwdbj, brianstelter, tragic, onair

Do you think the following set of words can describe the WDBJ shooting incident?

1. Virginia, journalists, shooting, newscast, heartbroken, condolence, tribute, everytown, community, support

- Yes
- No