# Probabilistic Topic Modeling in News Clustering & Representation

Iqra Shahid
Dept. of Computer Science
New York University
is1293@nyu.edu

Rachit Mehrotra
Dept. of Computer Science
New York University
rm4149@nyu.edu

***Abstract—***

**Probabilistic topic models are widely used in many text-mining tasks such as retrieval and extraction, clustering and summarization. We have used this technique in creating news story representation of textual documents. A document is defined to have latent topics or extracted information. We considered the vector space probabilistic model based on Latent Dirichlet Allocation. In this project we present a news search engine that applies this technique of topic detection on NY times and Guardian News corpus. We index and crawl our news using the API's for each, based on the query given to our search engine, categorize these in topical semantic categories (World News, Sports, Environment etc.) by assigning a color encoding for each category and returning a chronological representation of the stories. We used suffix-stripping algorithm by M.F.Porter to remove suffixes from words and Stopwords from sentences in news document corpus by automatic means in order to generate topics. We then use the topical clustering (LDA by Blei et al. 2003) to identify and extract topics from the above words output. These topics are extracted and placed in each category represented as topic summarization above each story, which is reflected by the color-coding. We demonstrate a system that automatically extracts topics from large collection. Each topic associated with news further represents related news when clicked. Our methods for topic detection, for assigning color-coding to reflect a category, and for topic clustering are completely unsupervised.**

***Keywords—LDA, topic, representation, summarization, crawling, news, tokenizing, stopping, stemming, indexing, filtering, color encoding, Gibbs Sampling***

## I. INTRODUCTION

When processing news stories of several accounts of a certain happening, it is often relevant to determine a topic reporting on the event described which deals with a certain set of topics. We developed a system, which captures user query to serve news stories. Our News Search Engine results in Story Representation and Topic Summarization given a query. Using NY Times and Guardian News API as our initial index database, web crawling is done to retrieve and index recent news, generate news summary in chronological representation and topical clustering. Parsing, stemming, filtering and indexing is done to support query search processing. Implemented vector space model i.e. Adaptive LDA to cluster and classify news data as topic summarization. Number of topic clusters is made adaptive to news data. Finally deployed the website to show this representation using Nodejs, JavaScript, jQuery, css and html.

The news that is gathered is segmented into categories, where each story reports on a different event. This is grouped as a preprocessing step for mining and topic clustering and summarization. Models for recognizing the topics in a text are well established. We used the probabilistic topic model; probabilistic Latent Semantic Analysis and Latent Dirichlet Allocation (LDA) (Blei et al. 2003). LDA is a flexible model that estimates the properties of text. We use Gibbs sampling as a means of approximate inference in Bayesian networks. The goal of this project was to study and implement the topic extraction techniques applied to text sources such as News and to use this Gibbs Sampling in order to quantify more accurately the topics from News documents, thereby approximating inferences of LDA and improving topic-based clustering. Our methods are completely unsupervised.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 tells the architecture and flow of our project. Section 4 describes our methodology. Evaluations and results are presented in Section 5. Future work presented in Section 6. In Section 7, we present our conclusions.

## II. RELATED WORK

Parameter estimation methods common with discrete probability distributions, which is of particular interest in text modeling is presented by Gregor.[3] As an application, the model of latent Dirichlet allocation (LDA) is explained by him in detail with a full derivation of an approximate inference algorithm based on Gibbs sampling, including a discussion of

Dirichlet hyper parameter estimation. [3] Similar work has been done. Statistical language models can learn relationships between topics discussed in a document collection. [4] The authors present a novel combination of statistical topic models and named-entity recognizers to jointly analyze entities mentioned and topics discussed in a collection of 330,000 New York Times news articles. They demonstrate an analytic framework, which automatically extracts from a large collection: topics; topic trends; and topics that relate entities. [4] In another similar work, the authors have studied several techniques for creating and comparing content representations of textual documents in the field of event detection. As underlying models they consider the vector space model and probabilistic topic models based on Latent Dirichlet Allocation. Their methods for aspect detection, for learning the importance factors of the aspects, and for event clustering are completely unsupervised. [5] For the process of stemming; removing suffixes by automatic means is an operation, it is especially useful in the field of information retrieval. 6]

## III. DESIGN AND ARCHITECTURE

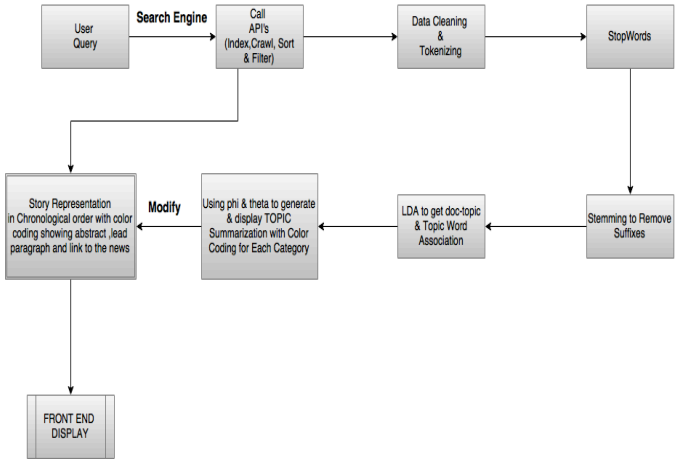


Figure 1: Flow Diagram Of The System

## IV. METHODOLOGY

This section explains in detail our project from crawling the news, application of suffix stripping algorithm and LDA with a full derivation of an approximate inference algorithm based on Gibbs sampling and representation of new stories and topic summarization.

### 1. *Indexing, Crawling and Filtering*:

Our system first takes in a query from the user. Based on the query, the search engine makes a call to the Guardian API as well as NY Times API to make a web crawl. The q parameter indexes, crawls, sorts and filters the results to only those that include the search term for e.g.: NYU. The publication date, abstract, headline and lead paragraph is extracted and appended as Story Representation which is then displayed in chronological order by sorting the results by date. Before the display, few factors are considered and implemented. The category of the news articles is inferred from its abstract and each separate category is assigned a different color encoding to differentiate the genre of the news. To read the specific detailed article, a click to link redirects to the full original article on the Guardian/NY Times Website. Once this preprocessing is done, we topicise our generated stories by first removing Stopwords and then using the suffix splitting algorithm aka Porter Stemmer to remove suffixes. We then apply LDA algorithm to generate topics and finally display topic summarization with color encoding (discussed in the steps below) along with Story Representation modified with the topic color associated with it for each category.

### 2. *Data Cleaning, Tokenizing and Stopping:*

In order to generate a useful topic model, data cleaning is absolutely crucial. We follow the common Natural Language Processing methods to do so.

(i)Tokenizing: Converting a document to its atomic elements.

We have a collection of documents, each described by the words in the document title and document abstract. A vector of words or terms represents documents. The headlines and lead paragraphs from the filtered news are pushed into an array whose length is analyzed to generate documents. The sentences are split on space into words to perform Stopping and then Stemming. A for loop makes sure we traverse all documents to get a list of all tokens/words.

(ii)Stopping: removing meaningless words.

Certain parts of English speech are meaningless to the topic model such as ‘the’ and conjunctions. These words are called stop words and need to be removed from our list of words. We have constructed our own stopword list, which we include in our code. To remove Stopwords, we loop through our words list comparing each word to our stopword list.

### 3. *Suffix Splitting Algorithm- Porter Stemming:*

Stemming is also an NLP technique to reduce topically similar words to their root. This is important in topic modeling, which would otherwise view those terms as separate entities and reduce their importance in the model. Common stem words having similar meaning are conflated into a single term by removal of various suffixes –ing, ion etc. Stemming is performed by suffix stripping algorithm by M.F.Porter. By merging words equivalent in meaning helps in reduction of data size to be given to LDA algorithm to extract suitable topics, therefore in turn improving Information Retrieval performance. The suffix-stripping program will be given the word list extracted after Stopping stage and an explicit list of suffixes. With each suffix, the criterion is checked under which it may be removed from a word to leave a valid stem.

The first step of the algorithm deals with plurals and past participles. The subsequent steps are much more straightforward.[6]

**(a) *Implementation of the algorithm***

*Step: 1a*

Any word, or part of a word, has the form:

***[C] VCVC … [V]***

***[C](VC){m}[V]***

where C denotes a list of consonants, V denotes a list of vowels, square brackets denote arbitrary presence of their contents. 'm' is the measure of any word in this form. E.g.:

*m=0 TR, EE, TREE, Y, BY.*
*m=1 TROUBLE, OATS, TREES, IVY.*
*m=2 TROUBLES, PRIVATE, OATEN, ORRERY*

In a set of rules, the one with longest matching S1 for given word is picked. (Condition) S1 -> S2. This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. E.g.:

*SSES -> SS*
*caresses -> caress*

*IES -> I*
*ponies -> poni*

*Step: 1b*

The condition is usually given in terms of m, e.g. (m > 1) EMENT -> Here S1 is `EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2. The `condition' part may also contain *v* - the stem contains a vowel.

*(m>0) EED -> EE*
*feed -> feed*
*agreed -> agree*

*(*v*) ED ->*
*plastered -> plaster*
*bled -> bled*

*Step: 1c*

*(*v*) Y -> I*
*happy -> happi*
*sky -> sky*

*Step: 2*

In this step a program switch is done on the penultimate letter of the word being tested to make the test on it faster. The S1-strings are presented here in alphabetical order of their penultimate letter.

*(m>0) ENCI -> ENCE*
*valenci -> valence*

*(m>0) ANCI -> ANCE*
*hesitanci -> hesitance*

*(m>0) IZER -> IZE*
*digitizer -> digitize*

*Step: 3*

Similar techniques as in Step 2 are applied in further steps.

*(m>0)*
*ICATE -> IC*
*triplicate -> triplic*

*(m>0) ATIVE ->*
*formative -> form*

*Step: 4*

*(m>1) AL ->*
*revival -> reviv*

*(m>1) ANCE ->*
*allowance -> allow*

*(m>1 and (*S or *T)) ION ->*
*adoption -> adopt*

*Step: 5*

This is the tidying up step after the suffixes have all been removed.

*(m>1) E ->*
*probate -> probat*
*rate -> rate*

*(m=1 and not *o) E ->*
*cease -> ceas*

*(m > 1 and *d and *L) -> single letter*
*controll -> control*

*roll* -> *roll*

Measure m, helps decide whether to remove suffix thus being careful to not remove suffix if the word is too short.

**(b) *Issues***

The algorithm clearly explains that when a set of rules of the type *(condition) S1* → S2 are presented together, only one rule is applied, the one with the longest matching suffix *S1* for the given word. Despite this, the rules are sometimes simply applied in turn until either one of them succeeds or the list runs out. This leads to small errors in various places, for example in the Step 4 rules

*(m>1)ement* →
*(m>1)ment* →
*(m>1)ent* →

to remove final *ement*, *ment* and *ent*

Properly, *argument* stems to *argument*. The longest matching suffix is *-ment*. Then stem *argu-* has measure *m* equal to 1 and so *-ment* will not be removed at end of Step 4. But if the three rules are applied in turn, then for suffix *-ent* the stem *argum-* has measure *m* equal to 2, and *-ent* gets removed.

4. ***LDA (Latent Dirichlet Allocation) Gibbs sampler:***

Once our cleaning stage is done, we get a tokenized, stopped and stemmed list of words from all documents appended to this one list of vocabulary. We have used a Machine Learning algorithm for language processing called LDA. LDA by Blei et al. is a probabilistic model that estimates multinomial observations by using unsupervised learning. It performs latent semantic analysis (LSA) by using a convex combination of a set of component distributions to model observations. To generate an LDA model, we need to understand how frequently each term occurs within each document. To do that we constructed a document-term matrix. Given this assumption of how documents are created, LDA backtracks and tries to figure out what topics would create those documents in the first place. LDA assumes documents are produced from a mixture of topics. Those topics then generate words based on their probability distribution. In other words, the main objectives of LDA inference: to find

(1) The term distribution p (t|z=k) = ϕ~k for each topic k and
(2) The topic distribution p (z|d=m) = ϑ~ m for each document m.

The estimated parameter sets Φ = {ϕ~k} K k=1 and Θ = {ϑ~ m} M m=1 are the basis for latent-semantic representation of words and documents.

The Bayesian network of LDA is interpreted as follows:
LDA generates a stream of observable words wm,n, partitioned into documents w~ m. For each of these documents, a topic proportion ϑ~ m is drawn, and from this, topic-specific words are emitted. That is, for each word, a topic indicator zm,n is sampled according to the document-specific mixture proportion, and then the corresponding topic-specific term distribution ϕ~zm,n used to draw a word. The topics ϕ~k are sampled once for the entire corpus.

The exact inference of LDA is difficult. So we use Gibbs sampling as an approximate inference algorithm. Gibbs sampling is a special case of Markov-chain Monte Carlo (MCMC) simulation. MCMC methods emulate high-dimensional probability distributions p (x) by the stationary behavior of a Markov chain, which happens after a so-called "burn-in period". The algorithm works as follows-
1. Choose dimension i (random or by permutation16)
2. Sample xi from p (xi |~x¬i).

***Gibbs sampling algorithm->***
Using the equations for full conditional and theta and phi, the Gibbs sampling procedure is run.
***Full conditional.*** From the joint distribution,[3]

$$\mathbf{p(\sim z,w\sim|\alpha,\sim\beta) = \prod Kz=1 \Delta(\sim nz+\sim\beta)/\Delta(\sim\beta)\cdot \prod Mm=1 \Delta(\sim nm+\alpha\sim)/\Delta(\alpha\sim)}$$

Derivation of the full conditional distribution for a word token with index i=(m, n), given that w~ = {wi=t,w~ ¬i} and ~z = {zi=k,~z¬i}yields:

$$\mathbf{p(zi=k|\sim z\neg i, w\sim) = p(w\sim,\sim z) / p(w\sim,\sim z\neg i)}$$
$$\mathbf{= p(w\sim|\sim z) /p(w\sim \neg i|\sim z\neg i)p(wi)\cdot p(\sim z) /p(\sim z\neg i)}$$
$$\mathbf{\propto \Delta(\sim nz+\sim\beta)/\Delta(\sim nz,\neg i+\sim\beta)\cdot \Delta(\sim nm+\alpha\sim)/\Delta(\sim nm,\neg i+\alpha\sim)}$$

$$\mathbf{\propto \Gamma(n(t)k+\beta t)\,\Gamma(\Sigma V\, t=1\, n(t)k,\neg i+\beta t)/\Gamma(n(t)k,\neg i+\beta t)\,\Gamma(\Sigma V\, t=1\, n(t)k+\beta t)\cdot \Gamma(n(k)m+\alpha k)\,\Gamma(\Sigma K\, k=1\, n(k)m,\neg i+\alpha k)/\Gamma(n(k)m,\neg i+\alpha k)\,\Gamma(\Sigma K\, k=1\, n(k)m+\alpha k)}$$

$$\mathbf{\propto n(t)k,\neg i+\beta t\ \Sigma V\, t=1\, /n(t)k,\neg i+\beta t\cdot n(t)m,\neg i+\alpha k\,/ [\Sigma K\, k=1\, n(k)m+\alpha k]-1}$$

Where the counts n (t) k,¬i indicate that the token i is excluded from the corresponding document or topic.

***Multinomial parameters.*** Finally, we need to obtain the multinomial lda parameter sets Θ and Φ; per document distributions over topics and per-topic distributions over words respectively. They correspond to the state of the Markov chain, M = {w~,~z}.[3]

$$\mathbf{\phi k,t = n(t)k+\beta t\ /\Sigma V\, t=1\, n(t)k+\beta t\ ,}$$
$$\mathbf{\vartheta m,k = n(k)m+\alpha k\ /\Sigma K\, k=1\, n(k)m+\alpha k}$$

The Gibbs sampling algorithm runs over the three periods: initialization, burn-in and sampling. To obtain the resulting model parameters from a Gibbs sampler, several approaches exist. One is to just use only one read out, another is to average a number of samples, and often it is desirable to leave an interval of L iteration between subsequent read-outs to obtain de correlated states of the Markov chain. This interval is often called "thinning interval" or sampling lag.
The Pseudo Code for Gibbs Sampling Algorithm that we have implemented is –

$M$ number of documents to generate (const scalar).
$K$ number of topics / mixture components (const scalar).
$V$ number of terms $t$ in vocabulary (const scalar).
$\vec{\alpha}$ hyperparameter on the mixing proportions ($K$-vector or scalar if symmetric).
$\vec{\beta}$ hyperparameter on the mixture components ($V$-vector or scalar if symmetric).
$\vec{\vartheta}_m$ parameter notation for $p(z|d=m)$, the topic mixture proportion for document $m$. One proportion for each document, $\underline{\Theta} = \{\vec{\vartheta}_m\}_{m=1}^{M}$ ($M \times K$ matrix).
$\vec{\varphi}_k$ parameter notation for $p(t|z=k)$, the mixture component of topic $k$. One component for each topic, $\underline{\Phi} = \{\vec{\varphi}_k\}_{k=1}^{K}$ ($K \times V$ matrix).
$N_m$ document length (document-specific), here modelled with a Poisson distribution [BNJ02] with constant parameter $\xi$.
$z_{m,n}$ mixture indicator that chooses the topic for the $n$th word in document $m$.
$w_{m,n}$ term indicator for the $n$th word in document $m$.

Figure 2: Quantities in the model of LDA[3]

⊡ initialisation
zero all count variables, $n_m^{(k)}, n_m, n_k^{(t)}, n_k$
**for** all documents $m \in [1, M]$ **do**
  **for** all words $n \in [1, N_m]$ in document $m$ **do**
    sample topic index $z_{m,n}=k \sim \text{Mult}(1/K)$
    increment document–topic count: $n_m^{(k)} + 1$
    increment document–topic sum: $n_m + 1$
    increment topic–term count: $n_k^{(t)} + 1$
    increment topic–term sum: $n_k + 1$
  **end for**
**end for**
⊡ Gibbs sampling over burn-in period and sampling period
**while** not finished **do**
  **for** all documents $m \in [1, M]$ **do**
    **for** all words $n \in [1, N_m]$ in document $m$ **do**
      ⊡ for the current assignment of $k$ to a term $t$ for word $w_{m,n}$:
      decrement counts and sums: $n_m^{(k)} - 1; n_m - 1; n_k^{(t)} - 1; n_k - 1$
      ⊡ multinomial sampling acc. to Eq. 79 (decrements from previous step):
      sample topic index $\tilde{k} \sim p(z_i|\vec{z}_{\neg i}, \vec{w})$
      ⊡ use the new assignment of $z_{m,n}$ to the term $t$ for word $w_{m,n}$ to:
      increment counts and sums: $n_m^{(\tilde{k})} + 1; n_m + 1; n_{\tilde{k}}^{(t)} + 1; n_{\tilde{k}} + 1$
    **end for**
  **end for**
  ⊡ check convergence and read out parameters
  **if** converged and $L$ sampling iterations since last read out **then**
    ⊡ the different parameters read outs are averaged.
    read out parameter set $\underline{\Phi}$ according to Eq. 82
    read out parameter set $\underline{\Theta}$ according to Eq. 83
  **end if**
**end while**

Figure 3: Gibbs Sampling Algorithm for LDA [3]

The breakdown is such that once we have our vocabulary list and document list of word indices, we configure our LDA to be set to 10,000 iterations, 2000 burn-in, 100 thin-intervals and 10 sample-log. We then call the Gibbs Sampling algorithm using 6 topics and 0.1 alpha and beta each. The Gibbs algorithm first initializes the sampler statistics (theta sum and phi sum) and then initializes state of the Markov chain. This starts with an assignment of observations to topics. Random assignments with equal probabilities is chosen and an element is selected. Conditional on other elements is updates i.e. sample a topiz z from the full conditional distribution then update the parameters by adding to the statistics the values of theta and phi for the current state.

### *5. Topic Summarization:*

Finally we retrieve –
(i) Theta- estimated document-topic associations. If sample lag > 0 then the mean value of all sampled statistics for theta [][] is taken. We get theta multinomial mixture of document topics (M x K).

(ii) Phi- estimated topic-word associations. If sample lag > 0 then the mean value of all sampled statistics for phi [][] is taken. We get phi multinomial mixture of topic words (K x V).

To get topics and present a topic summarization for each document, we run a for loop for the length of phi and traversing for each document/category, we set an array of tuples with the word from vocabulary and its topic word association probability to a precision of 2 each time. Each probability is separately stored in another array-cluster. We then reverse sort our array of tuples. Setting our top terms to our vocabulary length and traversing, we split the tuples to generate topic terms and probability for it. Provided the probability is greater than 0.0001, we add the topic terms to an array and sort it. The maximum similarity between these topic words calculated. If the number of similar words is less than 15, then we set our flag to 1 and loop to print the topic summarization for each document/category.

### *6. Color Encoding and User Interface:*

*Framework:* HTML, jQuery, Nodejs, CSS, JavaScript

The topics are displayed for each document and differentiated with the other by a different color encoding. For e.g.: blue color is assigned to US News. Once we have the color for each category, we modify our results from API and represent the news story for that category with the same color encoding. All bag of words in topic summarization are links which when clicked further displays its related news.

## V. RESULTS

The above methodology is implemented and displayed as News Story Representation in chronological order and Topic Summarization with color encoding for each category in the Web Application as follows:


Figure 4: News Search Engine displays results for query passed


Figure 5: Topic Summarization displayed in the form of bag of words

Figure 6: Story Representation for each category in chronological order with link to full story

Example of Project Succeeding / Failing:

Success: Almost all queries result in the application working absolutely fine

E.g.:

- Barack Obama
- White House
- Hillary Clinton
- Manhattan

Failure: Queries that fail to get news results from the API, breaks the LDA Topicise function, which results in the Bag of Words Coloring to fail.

E.g.:

- NYU

Couldn't find any other examples where the API does not return an empty array, it is surprising that NYU results in an empty array whereas New York University works perfectly fine

## VI. CHALLENGES

*1)* If the Guardian/NY Times API does not return any results on the query , the application does not have anything to display and topicise resulting in an empty result page, due to which the results are displayed from only 1 API ( considering one of them return some results) , but the Color Coding fails as it requires all the results from both the API to work.

*2)* Due to JavaScript Canvas constrains the bag of words being used as a onClick to search that terms only works once, as once it reloads the page the onClick function stops working and requires the user to Manually Reload the page for it to work, this does not affect the functionality of the system in any way, just the bag of words no longer behave as hyperlinks after the first query search.

## VII. FUTURE WORK

As this is a project which features a search engine and topic summarization / depiction, it can be extended and scaled to include different genres rather than just news.

Anything ranging from Music, Movies or anything that can be divided into different categories can be searched and sorted through the above principles by modifying the implementation of LDA. For example, the same logic can be applied on Movies where you can basically search for a query and the results can be topicised based on the genre of the movie and then displayed accordingly.

Apart from being used as a Search Engine, this can also be used as a Trainer for Natural Language Processing as the system basically reads large amount of data and extracts topics out of it and then divides the data into those topics. Thus, making it a small machine-learning project if used properly.

We tried to work with solr and lucene and had the whole solr backend setup and working, but as the NY Times Corpus was paid and not available to us even through Bobst Library accounts, we didn't use it in the final project, but if the dataset is given even the solr part works perfectly with indexing, and getting results through solr collections of news, news is parsed and stored in solr by a java indexer with reads the meta tags from corpus and stores the news accordingly. So in the future we can acquire the NY Times dataset and make the solr part work, which makes the search engine even work offline and with much higher response time and accuracy.

## VIII. Conclusion

We wanted an approach to automatically separate out topics in news documents. This is essentially a clustering problem. LDA is a probabilistic model with a corresponding generative process. A topic is a distribution over a fixed vocabulary. Querying the LDA model is the operation to retrieve documents relevant to a query document. A query is simply a vector of words ˜w~, and we can find matches with known documents by estimating the posterior distribution of topics ˜~z given the word vector of the query ˜w~ and the LDA Markov state M = {~z,w~}: p( ˜~z| ˜w~;M).

On searching a query in our system, our call to API sorts and filters out news data (documents). We extract words from these documents and remove stop words as well as suffixes. The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and in flexional endings from words in English. We have presented the topical model of Latent Dirichlet Allocation (LDA) and a complete derivation of approximate inference via Gibbs sampling. To generate a document:

1. Randomly choose a distribution over topics
2. For each word in the document

a. Randomly choose a topic from the distribution over topics

b. Randomly choose a word from the corresponding topic (distribution over the vocabulary).

Finally we generated topics that were most frequent and assigned color encoding to each category of topics as well as their corresponding news story presented in chronological order. We clustered documents based on maximum probability for a topic.

The model of Latent Dirichlet allocation can be considered the basic building block of a general framework of probabilistic modeling of text and be used to develop more sophisticated and application-oriented models, such as hierarchical models, models that combine content and relational data.


## Acknowledgment

We would like to thank NY Times and Guardian News API for providing us with the data without which we couldn't have had algorithms on it. We would like to thank Prof. Ernest Davis for giving us the opportunity to explore these new technologies in the industry and equipping us with proper skillset to use these technologies. Lastly, we would thank the CIMS Support team for regularly helping out whenever we were stuck with hosting the engine on the cs.nyu.edu site.

External Libraries used- jQuery v.1.7.2


## References


[1] http://tartarus.org/~martin/PorterStemmer/ , Jan 2006.

[2] StephenClark. Topic Modelling and Latent Dirichlet Allocation. 2013

[3] Gregor Heinrich. Parameter estimation for text analysis.August 2008.

[4] David Newman, Chaitanya Chemudugunta, Padhraic Smyth, Mark Steyvers. Analayzing Entities and Topics in News Articles using Statistically Topic Models.

[5] Wim De Smet, Marie-Francine Moens. An Aspect Based Document Representation for Event Clustering.

[6] M.F.Porter. An algorithm for suffix stripping .1980

[7] https://rstudio-pubs-static.s3.amazonaws.com/79360_850b2a69980c4488b1db95987a24867a.html

[8] David M.Blei, Andrew Y. Ng, Michael I.Jordan.Latent Dirichlet Allocation.Jan 2003.

[9] http://xpo6.com/list-of-english-stop-words/

[10] http://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html