MODELLING FARMER SUICIDES IN INDIA

A Project Report

on Text analysis using Latent Dirichlet Allocation

Under The Guidance

Of

Prof. U. Dinesh Kumar
(Chair, Data Centre and Analytics Lab
Decision Sciences and Information Systems, IIM Bangalore)

By

Ipsita Praharaj,
Research Intern
Fourth Year Undergraduate, IIT Kharagpur

Abstract

The objective of the project was to converge to the major topics that contribute to the **Farmer Suicides in India using News Articles.**

Latent Dirichlet allocation is a generative statistical model that allows sets of observations to be explained by unobserved groups. Documents are represented as random mixtures over latent topics, where each topic is characterised by a distribution over all the words. It solves the problem to classify data according to the context it is used in the News articles on the basis of automatic detection of the likelihood of term co-occurrence.

It uses **Python NLTK library** to tokenise the article body and extract valuable information using these tokens. It further employs **Bag of Words** algorithm to form the Vectorised corpus and converges where the document topic and topic term distributions are fairly good**.**

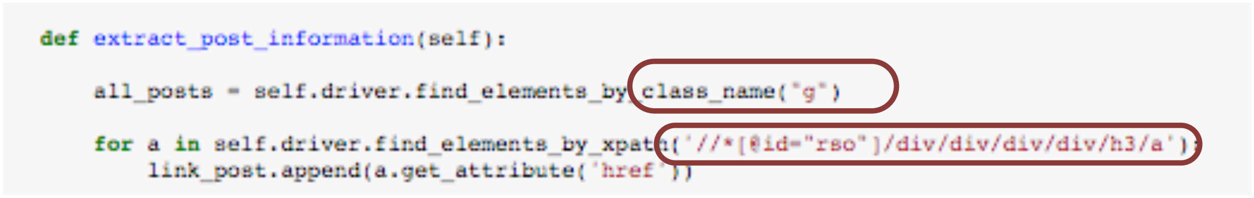
Table of Contents

1. Introduction

This section gives a scope description and overview of everything included in this Project Report. Also, the overview along with goal and vision are listed.

1.1. Automated Web Crawler using Selenium

The purpose of this section is automate click action and scrape multiple pages of the search result at once. Selenium web driver takes the user input and scrapes out the articles from the page(GOOGLE). After every page, it clicks the next button with a try-except mechanism. The tags can be modified here:



```
def extract_post_information(self):

    all_posts = self.driver.find_elements_by_class_name("g")

    for a in self.driver.find_elements_by_xpath('//*[@id="rso"]/div/div/div/div/h3/a'):
        link_post.append(a.get_attribute('href'))
```

1.2. Multi-Threading to Parse the Content

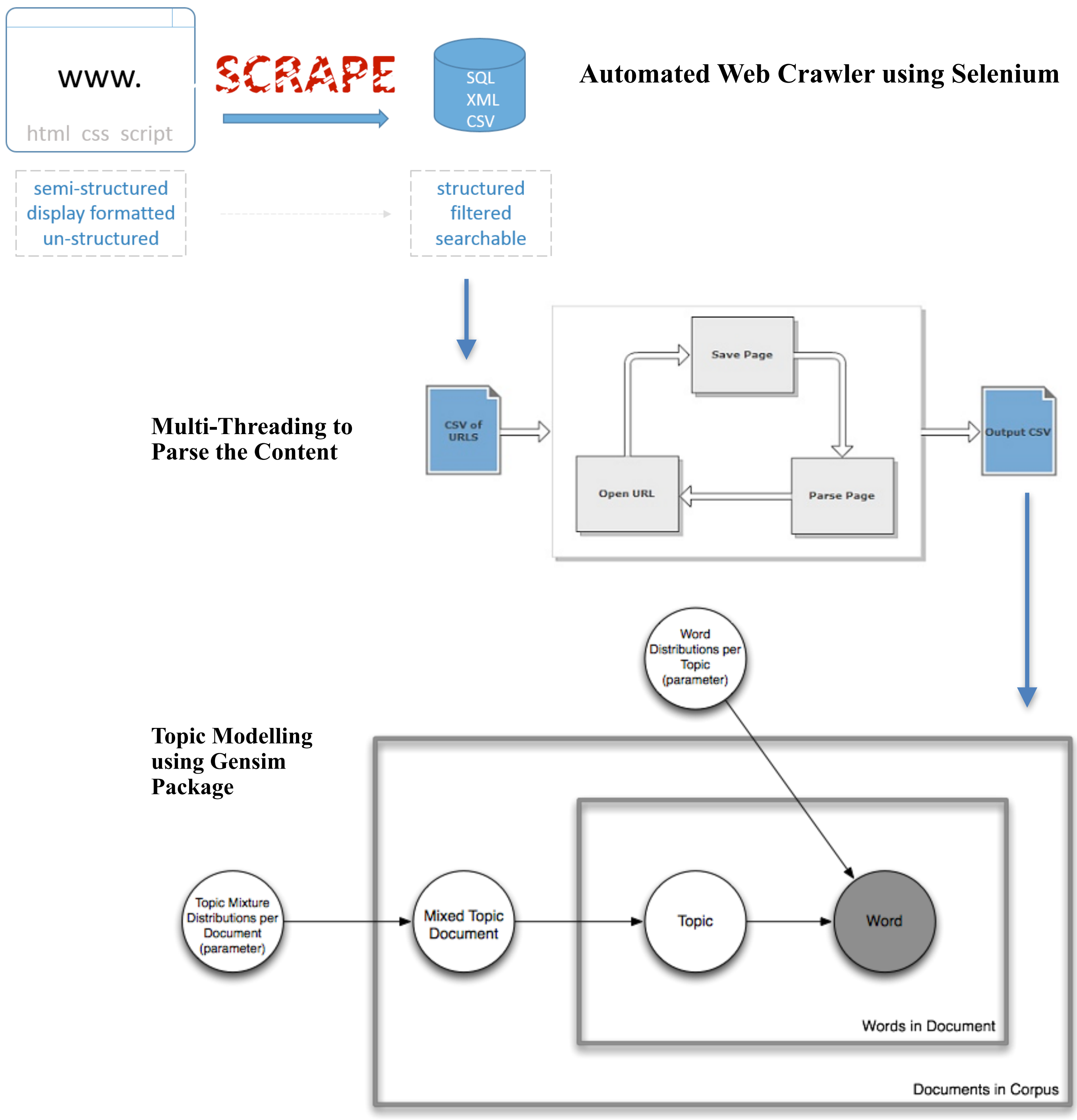
Multithreading achieves multitasking using **threads.** The processor to execute multiple threads concurrently by context switching, i.e. switching to state of another thread whenever anyone interrupts. This helps in quick requests parsing.

1.3. Topic Modelling using Gensim Package (LDA)

We encounter a lot of textual data or web pages during an odd day, and most of the time it is desirable to get a quick overview of what this text of web page is about. Topic Modelling solves this problem by semantically tagging the textual data and also provides a mechanism to classify textual data or web pages.

2. Design

2.1. Flow diagram of the code



2.FD of Application

2.2. Functional Requirements

2.2.1. Input requirements from the User

2.2.1.1. Functional Requirements 1.1

Actor: User

Input: Feed the News topic

2.2.1.2. Functional Requirements 1.2

Actor: User

Input: Change the Directory location and the Name of the Files.

Description: The output will be 2 CSV files.

- **Corpus** - CSV storing the DATE | HEADLINE | NEWSPAPER| NEWS LINK
- **Frequency** - CSV storing the PAPER | FREQUENCY | START INDEX | STOP INDEX | TIMER

2.2.1.2. Functional Requirements 1.2

Actor: User

Input: Specify the Number of topics to be generated in the LDA model

Description: The output will be an interactive chart.

- **Corpus** - CSV storing the DATE | HEADLINE | NEWSPAPER| NEWS LINK
- **Frequency** - CSV storing the PAPER | FREQUENCY | START INDEX | STOP INDEX | TIMER

2.3. Dependencies

This application requires preloaded packages to run.
There are used very modern frameworks for developing its frontend and backend.

- **Python 3 :** Language used
- **Jupyter Notebook :** This is a python framework used for developing backend of this application.

There are many third party python libraries used in this application for performing various tasks, the list is as follows:

1. Beautiful Soup
2. Selenium
3. Numpy
4. Pandas
5. Requests
6. NLTK
7. GENSIM

2.4. Hardware Requirements

To access a web portal of this application, its only need a PC/Laptop with an integrated and updated web browser.

Desktop browser : Chrome

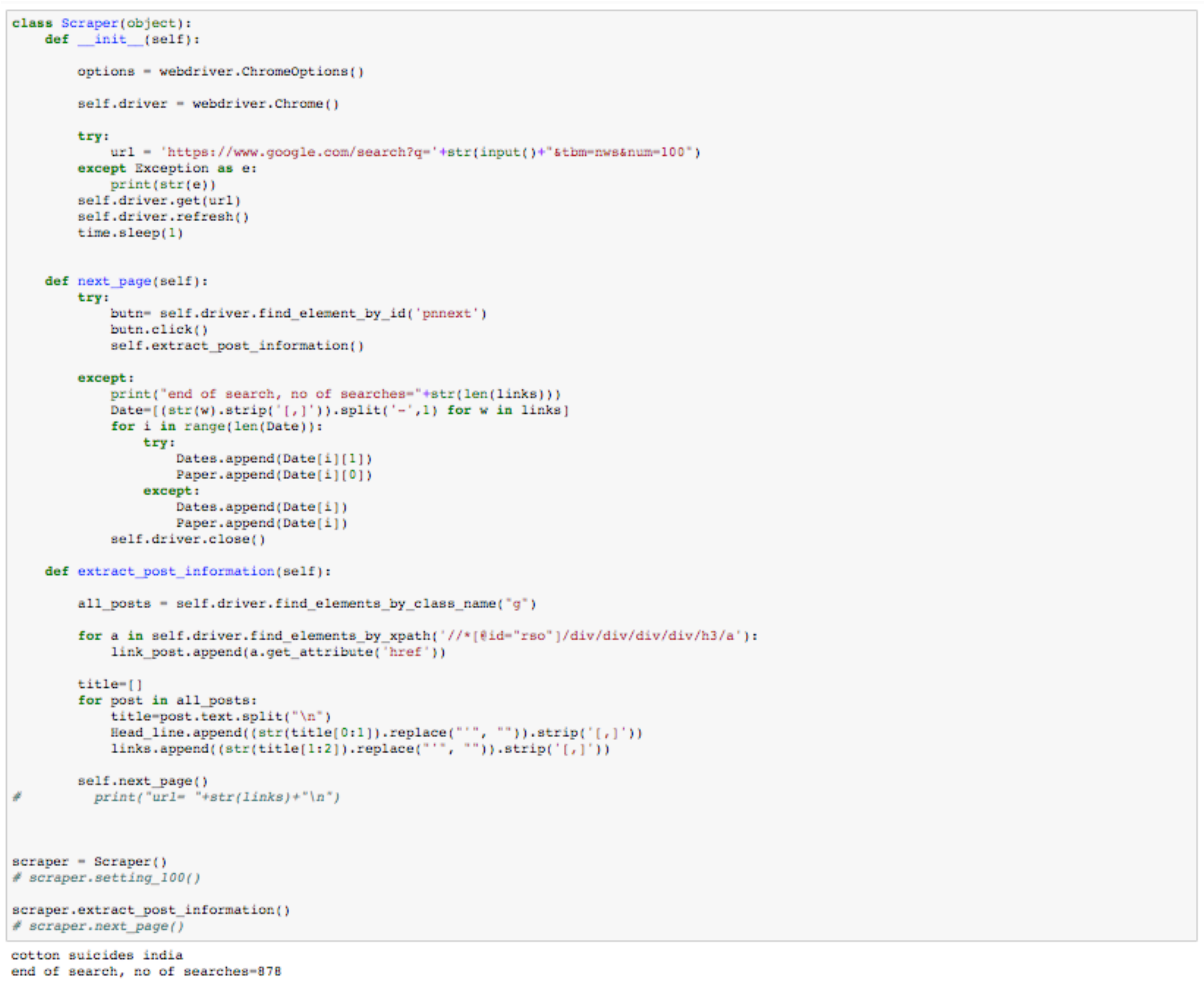
On the server side , a PC/Web Server which meets these specifications:

1. Linux/Ubuntu Operating System
2. At least 8 GB RAM
3. Python Compiler Installed

2.5. Constraints & Assumptions

- Only supports “English” Language and will not work with any other languages.
- It also assumes that the input is meaningful data and not some random characters. Any word in input which is not found in a standard dictionary may result in inaccurate tags. Therefore, corpus needs to be free from metadata
- Proxy rotation might fail in a Protected Network. Limiting amount of requests and increasing the time per requests solves this issue.

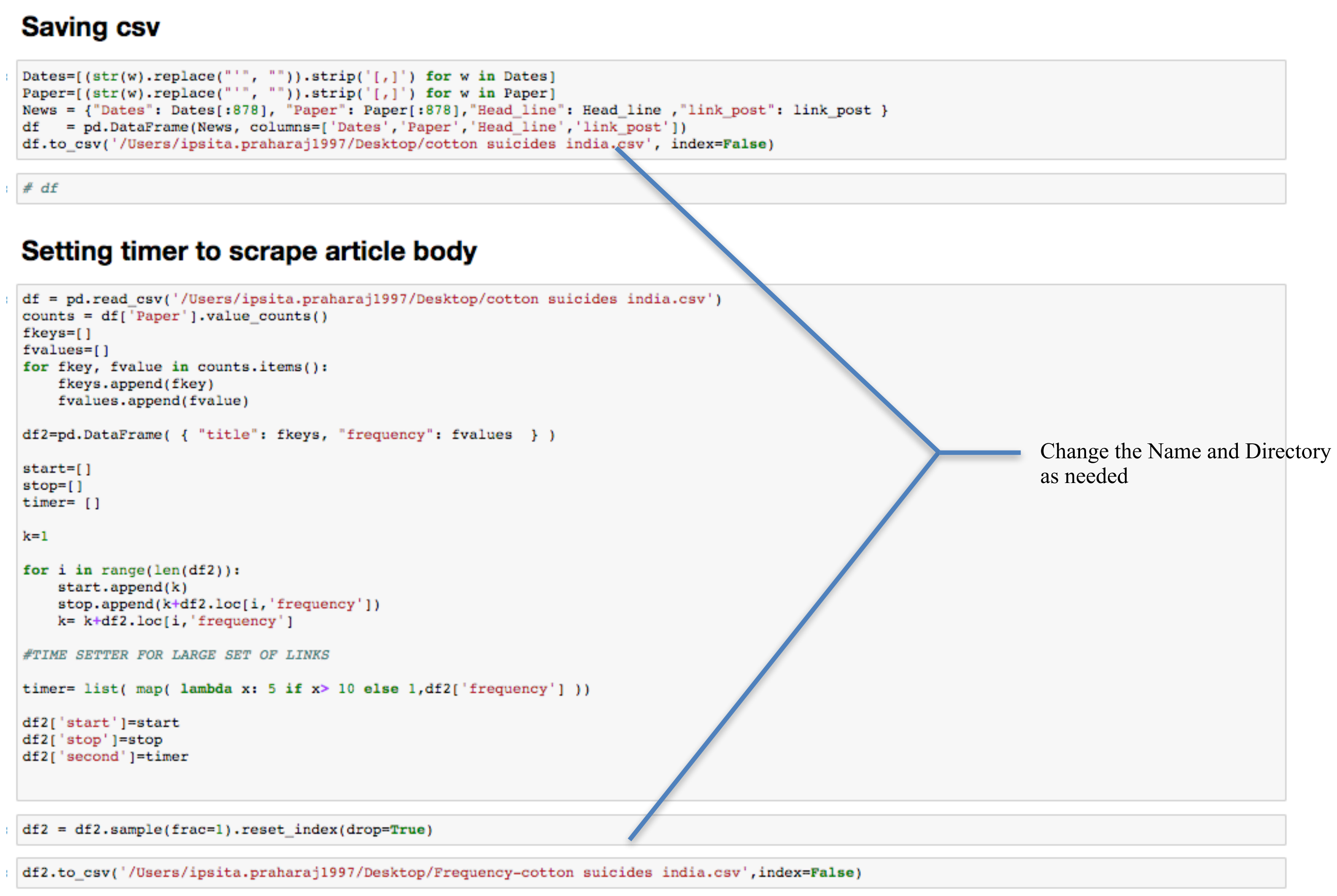
3. Coding



```
class Scraper(object):
    def __init__(self):

        options = webdriver.ChromeOptions()

        self.driver = webdriver.Chrome()

        try:
            url = 'https://www.google.com/search?q='+str(input()+"&tbm=nws&num=100")
        except Exception as e:
            print(str(e))
        self.driver.get(url)
        self.driver.refresh()
        time.sleep(1)


    def next_page(self):
        try:
            butn= self.driver.find_element_by_id('pnnext')
            butn.click()
            self.extract_post_information()

        except:
            print("end of search, no of searches="+str(len(links)))
            Date=[(str(w).strip('[,]')).split('-',1) for w in links]
            for i in range(len(Date)):
                try:
                    Dates.append(Date[i][1])
                    Paper.append(Date[i][0])
                except:
                    Dates.append(Date[i])
                    Paper.append(Date[i])
            self.driver.close()

    def extract_post_information(self):

        all_posts = self.driver.find_elements_by_class_name("g")

        for a in self.driver.find_elements_by_xpath('//*[@id="rso"]/div/div/div/div/h3/a'):
            link_post.append(a.get_attribute('href'))

        title=[]
        for post in all_posts:
            title=post.text.split("\n")
            Head_line.append((str(title[0:1]).replace("'", "")).strip('[,]'))
            links.append((str(title[1:2]).replace("'", "")).strip('[,]'))

        self.next_page()
#         print("url= "+str(links)+"\n")


scraper = Scraper()
# scraper.setting_100()

scraper.extract_post_information()
# scraper.next_page()
```

```
cotton suicides india
end of search, no of searches=878
```

Figure 1. Input the news topic to be scraped



Saving csv

```
Dates=[(str(w).replace("'", "")).strip('[,]') for w in Dates]
Paper=[(str(w).replace("'", "")).strip('[,]') for w in Paper]
News = {"Dates": Dates[:878], "Paper": Paper[:878],"Head_line": Head_line ,"link_post": link_post }
df   = pd.DataFrame(News, columns=['Dates','Paper','Head_line','link_post'])
df.to_csv('/Users/ipsita.praharaj1997/Desktop/cotton suicides india.csv', index=False)
```

```
# df
```

Setting timer to scrape article body

```
df = pd.read_csv('/Users/ipsita.praharaj1997/Desktop/cotton suicides india.csv')
counts = df['Paper'].value_counts()
fkeys=[]
fvalues=[]
for fkey, fvalue in counts.items():
    fkeys.append(fkey)
    fvalues.append(fvalue)

df2=pd.DataFrame( { "title": fkeys, "frequency": fvalues  } )

start=[]
stop=[]
timer= []

k=1

for i in range(len(df2)):
    start.append(k)
    stop.append(k+df2.loc[i,'frequency'])
    k= k+df2.loc[i,'frequency']

#TIME SETTER FOR LARGE SET OF LINKS

timer= list( map( lambda x: 5 if x> 10 else 1,df2['frequency'] ))

df2['start']=start
df2['stop']=stop
df2['second']=timer
```

```
df2 = df2.sample(frac=1).reset_index(drop=True)
```

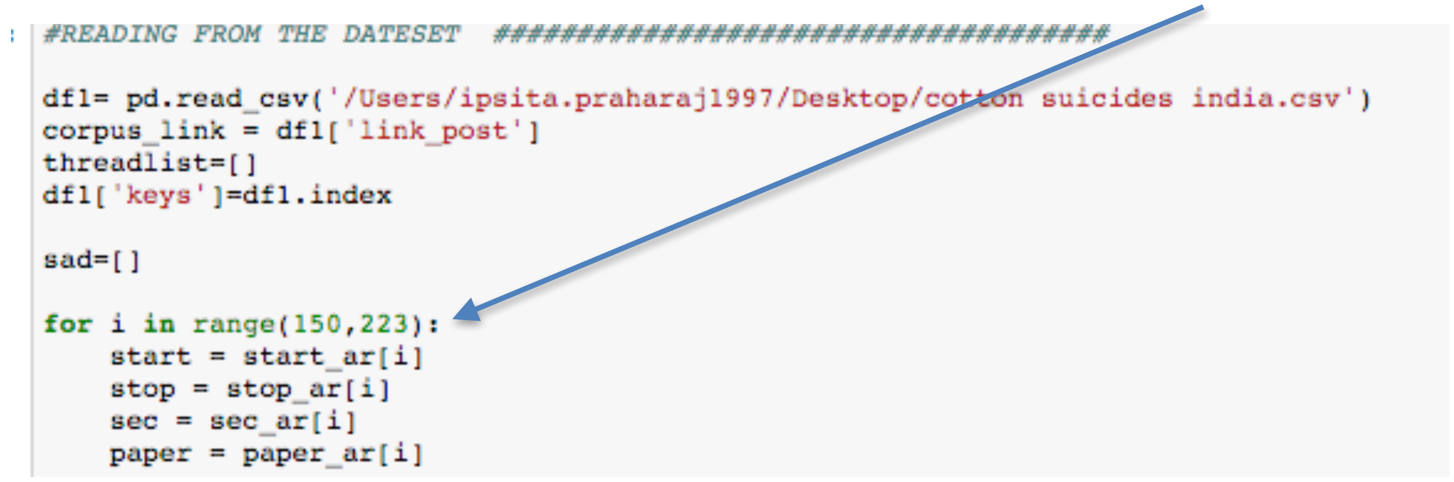
```
df2.to_csv('/Users/ipsita.praharaj1997/Desktop/Frequency-cotton suicides india.csv',index=False)
```

Figure 2. Save content from the scraper

4. Testing

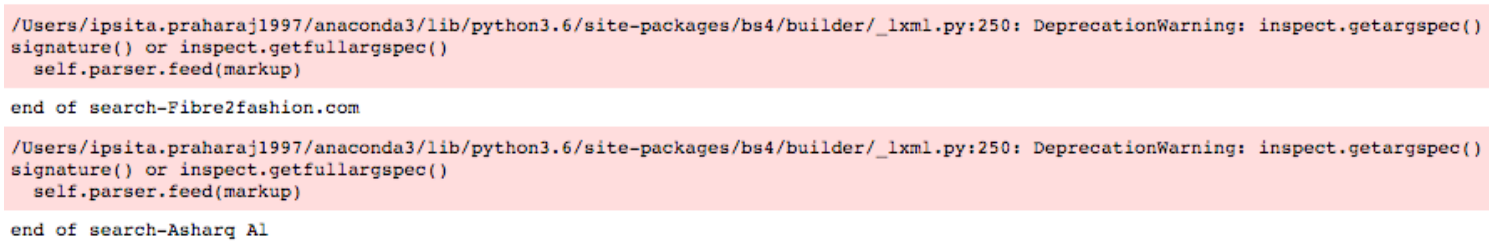
4.1. Test Plan

Unit Testing:
After generating the frequency CSV, set the range for parsing here:



```
#READING FROM THE DATESET   #######################################

df1= pd.read_csv('/Users/ipsita.praharaj1997/Desktop/cotton suicides india.csv')
corpus_link = df1['link_post']
threadlist=[]
df1['keys']=df1.index

sad=[]

for i in range(150,223):
    start = start_ar[i]
    stop = stop_ar[i]
    sec = sec_ar[i]
    paper = paper_ar[i]
```

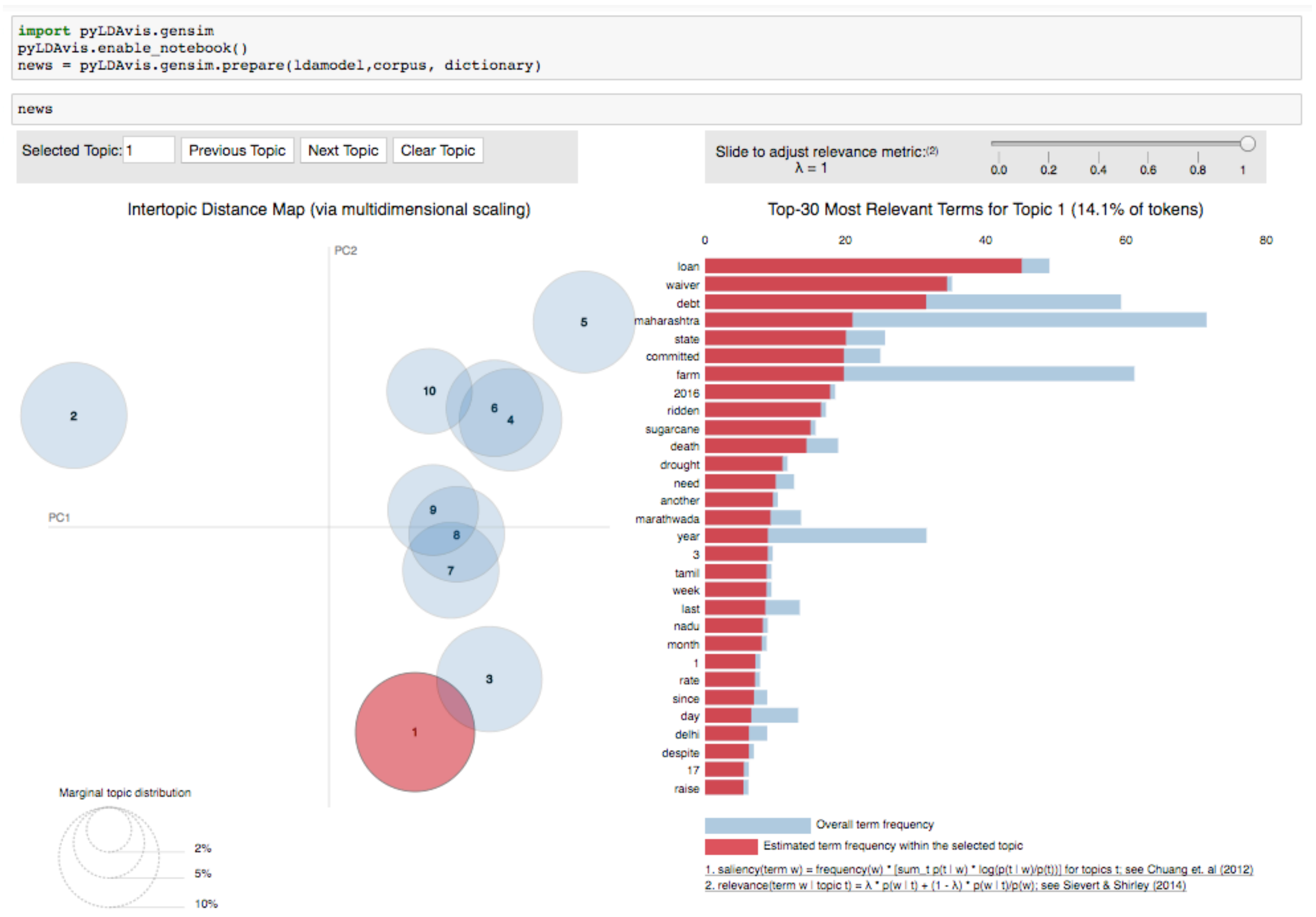
4.2. Test Report



```
/Users/ipsita.praharaj1997/anaconda3/lib/python3.6/site-packages/bs4/builder/_lxml.py:250: DeprecationWarning: inspect.getargspec()
signature() or inspect.getfullargspec()
  self.parser.feed(markup)

end of search-Fibre2fashion.com

/Users/ipsita.praharaj1997/anaconda3/lib/python3.6/site-packages/bs4/builder/_lxml.py:250: DeprecationWarning: inspect.getargspec()
signature() or inspect.getfullargspec()
  self.parser.feed(markup)

end of search-Asharq Al
```

Figure 10. Test CSV

4.3. Topic Modelling Deployment



```
import pyLDAvis.gensim
pyLDAvis.enable_notebook()
news = pyLDAvis.gensim.prepare(ldamodel,corpus, dictionary)
```

```
news
```

Selected Topic: 1

Slide to adjust relevance metric:(2) λ = 1

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 3. Visualisation of content

Alpha Hyperparameter

Alpha represents document-topic density.
Higher the value of alpha, documents are composed of more topics and lower the value of alpha, documents contain fewer topics.

<u>Circles indicate topics</u> : topics(circles) in same quadrant indicate similar topics.

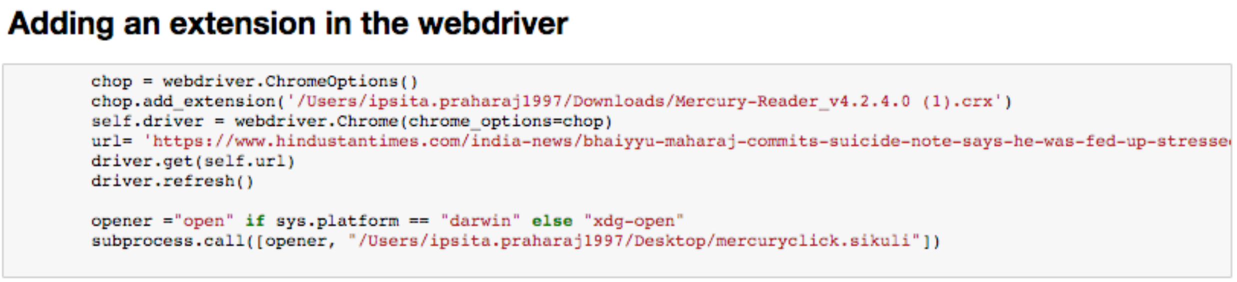
5. Future Work

Some of the possible amendments and improvements in this system are:

- Enabling Mercury Reader and MozBar extension in Selenium web driver to scrape out cleaner content
- Employing **Machine Learning** techniques to iterate the search based on the keywords from LDA.

Mercury Reader is a browser extension that cleans up your reading experience in Chrome. Firefox users have a reader option built right into the browser

MozBar is a browser extension that exports your search engine results page (SERP) analysis details to a CSV file.



Adding an extension in the webdriver

```
        chop = webdriver.ChromeOptions()
        chop.add_extension('/Users/ipsita.praharaj1997/Downloads/Mercury-Reader_v4.2.4.0 (1).crx')
        self.driver = webdriver.Chrome(chrome_options=chop)
        url= 'https://www.hindustantimes.com/india-news/bhaiyyu-maharaj-commits-suicide-note-says-he-was-fed-up-stresse
        driver.get(self.url)
        driver.refresh()

        opener ="open" if sys.platform == "darwin" else "xdg-open"
        subprocess.call([opener, "/Users/ipsita.praharaj1997/Desktop/mercuryclick.sikuli"])
```

By employing **Machine Learning** techniques this system may further be enhanced for better results. **Iterating the search based on the keywords from LDA** will help us to get to the bottom of the pipeline pinpointing the exact topics responsible for a broader topic. Supervised learning is the most preferable approach for the same it's easy to implement and train. Stop-words set can be improved from this.

6. Results

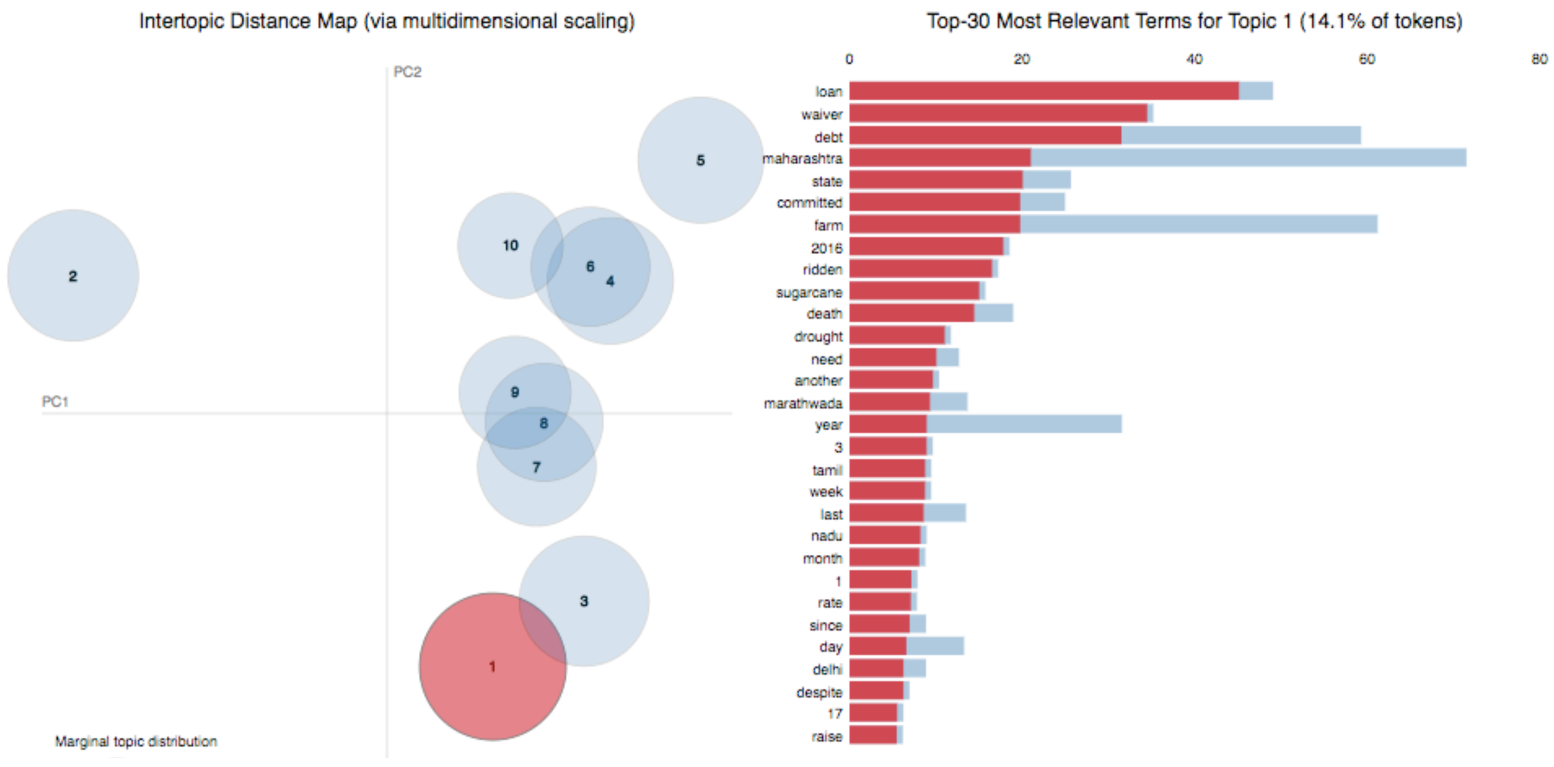
The model was carried out 3 times to narrow down the search and pinpoint the cause.

LDA on the headlines on the result of "farmer suicides in India"

Helps in prioritising the articles without processing on their corpus

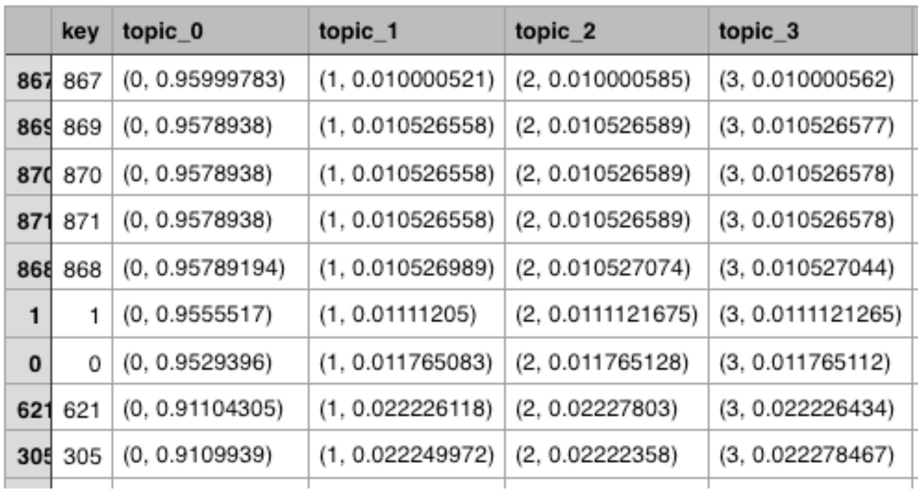
Maharashtra takes up the largest share in the document-topic and topic-word ratio

Collection of articles regarding farmer suicides in Maharashtra was collected

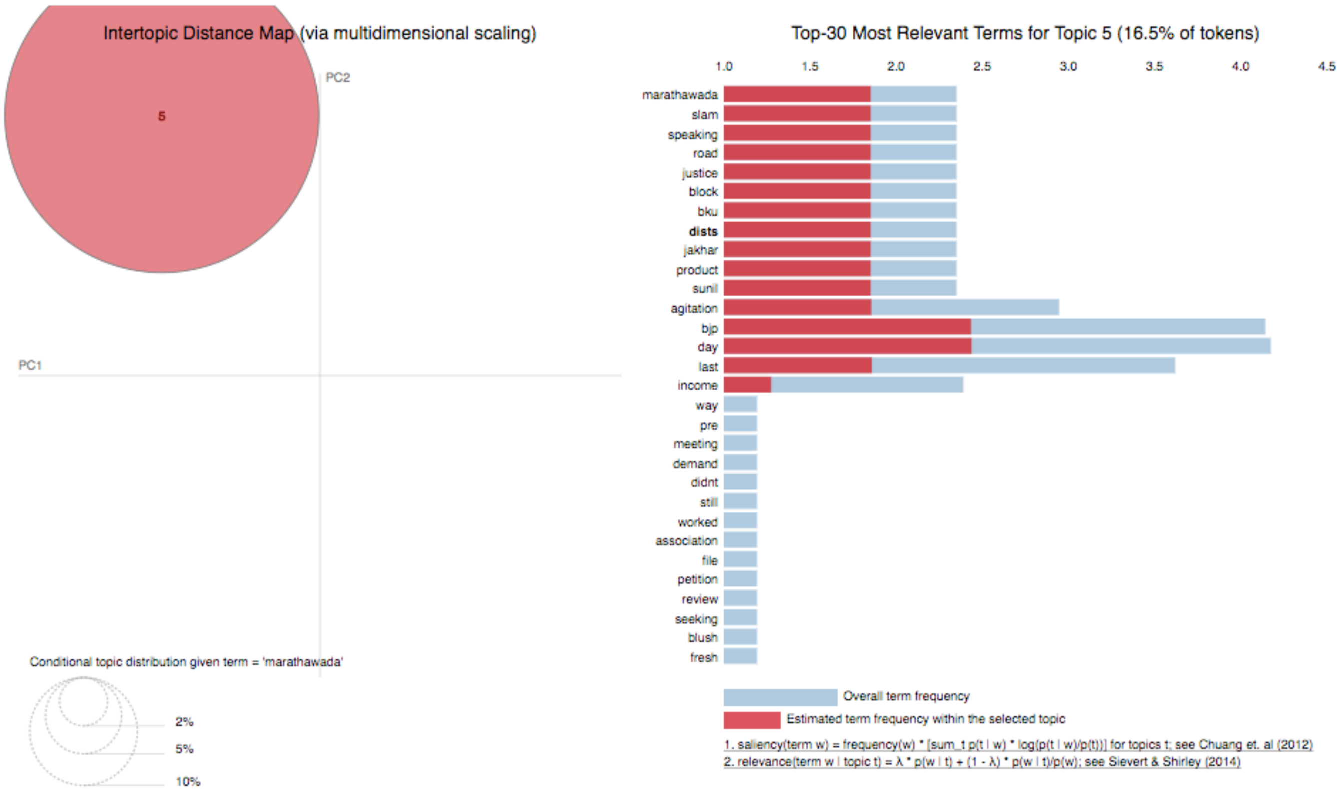


Arrange the corpus(document) in the descending order of the probabilities of topic-1(Maharashtra)

Setting a threshold and taking the keys to rerun the model



	key	topic_0	topic_1	topic_2	topic_3
867	867	(0, 0.95999783)	(1, 0.010000521)	(2, 0.010000585)	(3, 0.010000562)
869	869	(0, 0.9578938)	(1, 0.010526558)	(2, 0.010526589)	(3, 0.010526577)
870	870	(0, 0.9578938)	(1, 0.010526558)	(2, 0.010526589)	(3, 0.010526578)
871	871	(0, 0.9578938)	(1, 0.010526558)	(2, 0.010526589)	(3, 0.010526578)
868	868	(0, 0.95789194)	(1, 0.010526989)	(2, 0.010527074)	(3, 0.010527044)
1	1	(0, 0.9555517)	(1, 0.01111205)	(2, 0.0111121675)	(3, 0.0111121265)
0	0	(0, 0.9529396)	(1, 0.011765083)	(2, 0.011765128)	(3, 0.011765112)
621	621	(0, 0.91104305)	(1, 0.022226118)	(2, 0.02227803)	(3, 0.022226434)
305	305	(0, 0.9109939)	(1, 0.022249972)	(2, 0.02222358)	(3, 0.022278467)



---

major keywords: vidarbhas, marathawada, jakhar, bjp, climate, minister

On running LDA after setting a threshold we get the keywords for our next search.

Major keywords obtained after running LDA on the corpus of the filtered Keys are **vidarbhas, marathawada, bjp, climate, minister**

7. Summary

Systems like LDA are more efficient over LSA as the former one usage probabilistic latent semantic analysis (pLSA). It further provides a basic structure to develop larger systems utilising the similar concepts to classify data all over the internet and textual documents. Obtaining cleaner corpus over various newspapers poses a challenge as customising the code for each would increase the time of execution. Keywords obtained provide intuitions and the leads to further research.

8. References

- https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/
- https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
- http://www.nltk.org/
- https://www.geeksforgeeks.org/multithreading-python-set-1/
- https://github.com/vprusso/youtube_tutorials/tree
- http://www.awesomestats.in/python/
- https://github.com/ipsita-praharaj