Text Analytics

Applications of Natural Language Processing

Akash Gupta
Internship Mentor: Dr. Sowmya Dhanaraj
Madras School of Economics (PGDM)
09/08/2020

*Motivated by the concepts of **Natural Language Processing**, This report consists of analyses carried out on various datasets from domains like - **Finance, Politics, Journalism and Social-Media**. The unstructured text data, selected and mined from websites, is primarily in the form of - **Articles and Tweets**. We go through applications of **NLP** concepts like - Text Classification, Clustering and Latent Semantic Analysis. Classification and Clustering have been carried out on articles and Tweets corresponding to the **Migrant labour crisis in India**. Word Vectorization concepts have been implemented on articles corresponding to **Financial Sector News (India)**. Finally, an **Ngrams** analysis is presented on articles about **China**. Our central aim is to present the uses of NLP concepts in crafting predictive text classification models and Information extraction.*

Text Analytics

Akash Gupta
Madras School of Economics (PGDM)

Contents

Prelude

This report provides a detailed analysis on text data primarily utilizing concepts from:

- **Unsupervised Learning Methods (Clustering)**
- **Supervised Learning Methods (Classiciation)**
- **Natural Language Processing**
- **Linear Algebra**

The purpose of each of the above elements, within the framework of this analysis is as follows:

1. **Clustering** - To discover similarities between documents and their defining characteristics.
2. **Classification** - To train a classification model so as to predict the class label for a particular document.
3. **NLP** - To enable mathematical computation on text data by essentially breaking down text into fundamental elements like - **Tokens, Ngrams, Sentences** - and essentially **Vectorize** documents and words.
4. **Linear Algebra** - To perform dimensional reduction on **Vectorized text data** which would enable us determine key characteristics of documents and words.

Keywords

Here are some key concepts and definitions that will guide us through this report:

1. **Tokens** - These refer to individual terms and words in a text document. Here is an example of how **NLP** processes are leveraged to **Tokenize** documents. Below we can see how the example document has been broken into a list of words (tokens).

```
document1 = "Example document for demonstration."
nltk.word_tokenize(document1)
['Example', 'document', 'for', 'demonstration', '.']
```

2. **Vectorizing text** - The most fundamental method used to convert text into a numeric vector is as follows - ***Count the number of word occurrences in each document and enter that number as a value corresponding to the word-document pair in a Document-Term matrix***.

3. **Lexicon** - Essentially a dictionary. In this report when we mention **Sentiment lexicon** it refers to a dictionary containing positive and negative words. Here we can see how pre-defined sentiment lexicons have labeled a collection of words as negative and positive.

```r
head(get_sentiments('bing'))
```

```
## # A tibble: 6 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
```

4. **Key subjects** - In the framework of this report, **key subjects** refers to **proper nouns** that form the subject of a sentence and with respect to which, we are often interested to find related sentiments or judgements.

- ***The PM-KISAN scheme is benefitial to the farmers*** –> Here ***PM_KISAN*** would be our primary key subject.

5. **Context** - This refers to the key sentiment containing words present in the vicinity of our **Key subject** words.

- ***The Central Government has handled the crisis quite efficiently*** –> Here the key context words around our key subject ***Central Government***, that contain defining sentiment would be - ***Efficiently***.

5. **Factual Classification** - This is a form of classification wherein we might want to judge whether a sentence is **positive** or **negative**. This is fairly straightforward, in the sense, if there are more number of positive words in say, a review, then that review would be classified as **positive**.

- ***This laptop is really good*** –> ***Positive***
- ***This laptop functions poorly*** –> ***Negative***

6. **Subjective Classification** - This is a more tricky form of classification and essentially what we are dealing with in our analysis as well. Here the classification labels are not as rudimentary as **positive** and **negative**, rather they are what we define them to be, depending on the situation. This kind of classification essentially ***classifies a sentence/document based on the context around Key-subject words***.

- In this analysis, our classification objective is to label documents as ***PRO*** or ***ANTI*** government.

- A ***PRO*** government sentence is defined to be - A sentence having ***positive*** sentiment with respect to ***Government*** related Key-subjects. Alternatively, it is also defined to be - A sentence having ***negative*** sentiment with respect to ***Opposition*** related Key-subjects.
- An ***ANTI*** government sentence is defined to be - A sentence having ***negative*** sentiment with respect to ***Government*** related Key-subjects. Alternatively, it is also defined to be - A sentence having ***positive*** sentiment with respect to ***Opposition*** related Key-subjects.
- ***The opposition has handled the crisis poorly –> PRO***
- ***The government has done a bad job –> ANTI***
- ***The government has done a good job –> PRO***
- ***The opposition has been great at handling this –> ANTI***

The background

The domains under which this analysis has been carried out are essentially - **Politics** and **Journalism**. The aspect of **Politics** relates to the **politicization** of various social and economic issues. The aspect of **Journalism** relates to the newspaper/social-media coverage around various politicized socio-economic issues. Studying these aspects assumes relevance today since it is often observed that opinions and sentiments pertaining to various socio-eonomic issues tend to be quite **polarized**, in the sense that public sentiment around a particular issue is divided to a large extent, into two extremes. In addition, we are aware of the fact that newspapers and social-media often carry opinions that tend to influence a majority of people, given their widespread reach. Therefore, it can be reasonably assumed that opinions carried in newspapers and social-media platforms are quite relevant to observe because they in-turn influence the opinions and political sentiments of large sections of a population.

Given this background, our premise is that if we are able to essentially treat as data, this massive quantum of text in the form of **Articles** and **Tweets**, and conduct thorough analyses, we might obtain useful insights. Many possibilities are capable of emerging from studying and analysing such text data. Further, given our ability to analyse large volumes of textual data using Mathematical models, we can generate information that might help guide policy decisions, as well electioneering.

- ***State and Central governments might want to gather information on the public sentiment in response to various Schemes and policy decisions.***
- ***Since it is reasonable to assume that governments tend to work towards minimizing negative public sentiment, if they can ascertain the reasons or responses that trigger negative sentiment, they can essentially adopt appropriate measures to counter these negative sentimnents.***
- ***It is impossible to read through hundreds of articles and social-media texts in order to gain insights. This presents yet another advantage of text analytics - to***

summarise key insights relating to Key-subjects by scanning hundreds and thousands of articles.

- ***Election manifestos and campaign strategies can be planned by leveraging sentiment analyses and information extraction from text data.***

Our focus

Our focus is primarily on the **Migrant Labour crisis** that unfolded in the months of April, May and June as a consequence of city-wide lockdowns. **Migrant Labourer** typically refers to India's low-income, unskilled workforce mainly engaged in various unorganized sectors - **construction work, domestic help, textiles, brick kilns, transportation services, agriculture**. They are often subject to exploitative labour arrangements and are forced to work low-wage, hazardous jobs [Ajeevika Bureau]. They typically migrate from places like **Bihar, Jharkhand, Uttar Pradesh, West Bengal** to metropolitan cities in search of well paying jobs like **Delhi, Chennai, Mumbai, Bangalore**. Lockdowns, as a Coronavirus response rendered many such workers jobless. Unable to afford living expenses in the metropolitan cities, many wanted to migrate back to their hometowns. This created problems since essential transport services were shut and many were forced to walk to their hometowns, undertaking long, arduous journeys. Recognizing these issues, various governments responded by - **Arranging transport** and **announcing Economic relief packages** for the migrant workers. The migrant labour crisis became a rather large issue, even grabbing attention in international publications.

Upon inital observation, the problematic concerns of the migrant workers seemed to be turning into a political slugfest, with Journalists, Politicians and general public resorting to blame either the Government or Opposition for mismanagement. Our initial observations confirmed the existence of a heavily polarised sentiment pertaining to the migrant labour crisis. Soon, it seemed, that the discussion shifted from focussing on the plight of the migrant's conditions, to political subjects - essentially the issue was **politicized**. Indeed, this was fertile ground for conducting a sentiment analysis and implementing information extraction processes on text data corresponding to this topic. We essentially measure **Pro** and **anti** government sentiments arising from this topic.

In later sections, we shift our focus to **Financial Analysis** where we attempt to extract information and sentiment from Financial News articles concerning the **Automobile, Finance/Banking and Healthcare** sectors. Lastly, we will also demonstrate the concept of obtaining key insights by performing an **Ngrams** analysis on articles pertaining to **China** as the central topic.

Data

Having established the central aim of our analysis - **Sentiment classification and information extraction from text** - we now begin to understand our data. Primarily, data has been mined from newspaper websites and social-media platform **Twitter**. Articles have been mined from **The**

Indian Express, Times of India, Hindustan Times. With respect to mining text data from articles and Twitter, here is a series of steps explaining the process:

1. Digital articles in newspapers websites are mostly in **HTML** format and directly importing such data also brings with it, unnecessary HTML tags. Our job then is to write functions using **Regular Expressions** that essentially remove HTML tags from our data. Below is an example whererin we have essetially **Parsed** html tags from our sentence. Similar functions and procedures, albeit more sophisticated, have been applied on web articles to obtain a pure text form.

```
test_sentence = "this is a sentence with https://t.co/AbsC html tags"
replace_reg1 <- "https://t.co/[A-Za-z\\d]+|"
str_replace_all(test_sentence, replace_reg1, "")
```

```
## [1] "this is a sentence with  html tags"
```

2. After **HTML parsing** from the relevant articles pertaining to the Migrant Labour crisis, here is what the article data looks like:

```
head(corp_news,5)
```

```
## # A tibble: 5 x 2
##   title    text
##   <chr>    <chr>
## 1 toi_mig1 "as i travelled to nearby villages during the lockdown due to unavoi~
## 2 toi_mig2 "uttarakhand has 13 districts in total viz the plain districts of de~
## 3 toi_mig3 "forget mgnrega the government s flagship anti poverty programme mig~
## 4 toi_mig4 "ram parsad returned to his village in parsendi block of sitapur dis~
## 5 toi_mig5 "the covid pandemic crisis has led to the unfortunate displacement o~
```

3. Permission was obtained from **Twitter** after making a ***Twitter developer's account*** and Tweets pertaining to the Migrant issue were obtained:

```
head(tweet_df %>% select(text, num))
```

```
## # A tibble: 6 x 2
##   text                                                                    num
##   <chr>                                                                 <dbl>
## 1 #PMModi launches #AtmaNirbharUttarPradeshRojgarAbhiyan to provide jobs ~     1
## 2 #AatmaNirbharBharatAbhiyan in Uttar Pradesh will provide motivation and~     2
## 3 GoI has launched Garib Kalyan Rojgar Abhiyaan on 20th June 2020 to boos~     3
## 4 During the virtual launch of the #AtmaNirbharUttarPradeshRojgarAbhiyan,~     4
```

```
## 5 The massive rural public works scheme #GaribKalyanRojgarAbhiyaan will e~     5
## 6 PM Narendra Modi Launches 'Atma Nirbhar UP Rojgar Abhiyan', Over 1.25 C~     6
```

Methodology

We are aware of the fact that a typical classification task is essentially a **supervised learning method** which involves pre-labeled datasets. The class labels basically supervise the model training. However, in our case, the data is extremely unstructured and unlabeled as well. So, in order to be able to actually use classification processes, we have to first label a part of the dataset based on certain conditions. This poses the central challenge in our analysis - **assigning class labels to the training dataset for the task of subjective classification**. The following steps describe the methodology followed for assigning class labels on our cleaned dataset.

Context Extraction

1. In a large article of more than 1000 words, there are bound to be negative and positive words present involving various contexts. But, the contexts that particularly interest us are those that contain certain specific Key-subject words. We are interested in these sentences since they are the ones that carry the most significant sentiment evidence. For Example:-

- ***The situation of the Migrants is quite grim. They have had to undertake long, painful journeys to their hometowns and tend to have bad economic conditions. But, the central government announced a wonderful package called Garib-Kalyan-Yojana which will empower them.***
- We notice that in the above paragraph there are various negative and positive words, but the most relevant to our case are the words around the Key-subject ***central government, Garib-Kalyan-yojana***. This entire document displays various opinions, but the opinions most important to us are contained in essentially one sentence - But, the central government announced a wonderful package called Garib-Kalyan-Yojana which will empower them

2. Keeping the previous point in mind, we first attempt to extract out key sentences pertaining to a **check-list** of Government and Opposition related Key-subjects.

- Function that extracts the key ***Proper Nouns***. In this code block, ***NNP*** refers to proper nouns:

```
big_propers = []
for i in range(90):
    text = nltk.word_tokenize(df['text'][i])
    tagged_news = nltk.pos_tag(text)
    big_propers.extend(nltk.FreqDist(word for (word,tag) in
    tagged_news if tag.startswith("NNP")
                                    and and len(word)>1))

['ATMs', 'Aadhaar', 'Aadhar', 'Aajeevika', 'Aayog',...]
```

- Check-list that contains ***Government*** related terms:

```
check_list = ["Aadhaar","Aadhar","Aayog","Abhiyaan","Abhiyan","Adityanath","Amitabh",
"Atal","Atma","AtmaNirbhar","Atmanirbhar","Ayushman","BJP","BJP-RSS",
"Bhagwat","CARES","Central","Centre","Dhan","Dhan-Aadhaar-Mobile","Divider-in-chief",
"Gadkari","Garib","Hindu","Hindutva","India","JAM","LOCKDOWN","Lockdown",
"Lockdowns","MUDRA","Modi","NITI","Nirbhar","Nirmala","Nitish","PAEG",
"PM","PM-Cares","PM-KISAN","PMJAY","PMJDY","PMO","PMUY","Package",
"Railways","Railway","Rajnath","Sevak","Shivraj","Shramik","Yediyurappa","Yogi"]
```

- Check-list that contains ***Opposition*** related terms:

```
check_list = ["Banerjee","Arvind","Congress","Dynasty","Maharashtra",
"Mamata","Mamta","Manmohan","Mumbai","Opposition","Rahul","Shiv","Sonia","UPA"]
```

- Function that ***extracts sentences*** that contain words in the ***Check-list***:

```
sent_list = []
for i in range(90):
    for sent in nltk.sent_tokenize(df['text'][i]):
        for word in nltk.word_tokenize(sent):
            if (word in check_list) and (sent not in sent_list):
                sent_list.append((sent,df['title'][i]))
```

- ***Sonia Gandhi and West Bengal Chief Minister Mamta Banerjee had even an iota of concern for the migrants, they would have allowed the states ruled by them to bring back migrant workers, and all the misery migrants are having to endure could have been avoided.***
- ***However, I am sorry to say that the full capacity of these trains is still not being utilised by some states, especially the ones controlled by Congress and the TMC***
- ***The Opposition parties have both stoked and used migrant workers' conundrum to critique the government's Covid-19 strategy.***
- ***The government of India through the Ministry of Home Affairs and Ministry of Health and Family Welfare have been at the forefront of this battle against the Covid-19 virus.***

A Customized Lexicon

The general purpose Lexicon avaialable in Software packages tends to be fairly comprehensive however there are some important negative and positive words that they miss out on. After a quick scan through our data, many words were appended to the existing Lexicon to essentially give rise to a new Lexicon with the added words. Some of these new words are mentioned below:

- ***forerunner* –> *Positive***
- ***imporverishment* –> *Negative***
- ***assistance* –> *Positive***
- ***abandoned* –> *Negative***
- ***maligned* –> *Negative***
- ***provision* –> *Positive***
- ***encouraged* –> *Positive***
- ***stranded* –> *Negative***

Defining Key Features

After compiling our extracted sentences into a **DataFrame** object, we begin the primary task of assigning class labels - defining **key features** in the data. Features are essentially characteristics of certain documents that set them apart from other documents, perhaps with a different set of features. Here, we define our features to be - **The proportion of positive and negative words**. We adopt this approach since by now we are certain that the key judgemental words are infact, with respect to the Key-subject of our interest. Proportion has been taken instead of word count, since it essentially accounts for sentence length and gives us a measure of intensity of sentiment as well. For example, a short sentence with many negative words will have a high proportion of **negative-word to total-word count**.

1. The negative word ratio is obtained by:

$$\frac{num(Negativewords)}{num(totalwords)}$$

2. The positive word ratio is obtained by:

$$\frac{num(Positivewords)}{num(totalwords)}$$

3. These two ratios are subtracted to get the final score or the defining ***feature*** of a particular sentence:

$$Score = \frac{num(Positivewords)}{num(totalwords)} - \frac{num(Negativewords)}{num(totalwords)}$$

4. The condition is that if the ***Score*** is positive, then we assign that sentence a class label of ***PRO***, whereas if the ***Score*** is negative, then we assign that sentence a class label of ***ANTI***. Here is a look at this computation:

```r
head(total_arts,5)
```

```
## # A tibble: 5 x 5
##   title    poss_r negs_r    cats lab
##   <chr>     <dbl>  <dbl>   <dbl> <chr>
## 1 ht_mig73 0.102  0.0625  0.0391 pro
## 2 ht_mig74 0.0968 0.145  -0.0484 anti
## 3 ht_mig76 0.0442 0.0265  0.0177 pro
## 4 ht_mig77 0.04   0.16   -0.12   anti
## 5 ht_mig78 0.0625 0.188  -0.125  anti
```

5. We note that the ***title*** column refers to the article from which the key sentences were extracted for the task of class labeling. Now that we have labeled our sentences, we extend the same labeling to the entire article as well. After appending these class labels to the articles we get the following dataset:

```r
head(corp_marked, 5)
```

```
## # A tibble: 5 x 5
##   title   text                                                  num label target
##   <chr>   <chr>                                               <int> <chr>  <dbl>
## 1 toi_mi~ "as i travelled to nearby villages during the lock~     1 pro        1
## 2 toi_mi~ "uttarakhand has 13 districts in total viz the pla~     2 pro        1
## 3 toi_mi~ "forget mgnrega the government s flagship anti pov~     3 pro        1
## 4 toi_mi~ "the covid pandemic crisis has led to the unfortun~     5 pro        1
## 5 toi_mi~ "lockdown and reverse movement the reverse movemen~     7 pro        1
```

6. Here is an example of some sentences that were assigned appropriate class labels:

- ***to its credit the modi government has ticked all the right boxes –> PRO***
- ***ever since the nationwide lockdown began on march 25 unending miseries have been visited on the migrant workers stranded in india s cities –> ANTI***
- ***fiscal packages and monetary stimulus will have limited impact unless india creates a demand pull that alleviates impoverishment –> ANTI***
- ***not only has prime minister narendra modi emphasised unlock unlock unlock as the way forward many industries have restarted in some form –> PRO***

Classification

The previous section outlines our methodology regarding assigning class labels such that we are able to create a classification model for the data. The purpose of doing this is so that we are able to

predict whether a given article would be **PRO** or **ANTI** government, without even looking at it. At the outset we would however state that, this analysis does not give us near-perfect prediction accuracies of the order of **90%**. We get a lower accuracy of around **70%** to **74%**. However, given that this is essentially a **subjective classification** with many subtle nuances in forms of natural language, that level of accuracy is not bad to start with. At the same time, it is definitely worth mentioning that possibly with various other permutations and combinations of feature characteristics, we might arrive at slightly better accuracy levels. We primarily apply the **Naive Bayes** and **Support Vector Machine** learning models to our dataset.

Naive Bayes Classification

The following results were obtained after training a **Naive Bayes Classifier** that gave us a **70%** accuracy.

```
Most Informative Features
     contains(local) = True              pro : anti   =     10.4 : 1.0
contains(infrastructure) = True          pro : anti   =      6.2 : 1.0
 contains(transport) = True             anti : pro    =      5.6 : 1.0
    contains(market) = True              pro : anti   =      5.4 : 1.0
  contains(activity) = True              pro : anti   =      4.6 : 1.0
 contains(economies) = True              pro : anti   =      4.6 : 1.0
 contains(education) = True              pro : anti   =      4.6 : 1.0
contains(healthcare) = True              pro : anti   =      4.6 : 1.0
 contains(infection) = True              pro : anti   =      4.6 : 1.0
contains(productivity) = True            pro : anti   =      4.6 : 1.0
contains(uncertainty) = True             pro : anti   =      4.6 : 1.0
contains(humanitarian) = True           anti : pro    =      4.5 : 1.0
contains(opportunities) = True           pro : anti   =      4.3 : 1.0
       contains(bus) = True             anti : pro    =      4.0 : 1.0
contains(livelihoods) = True            anti : pro    =      4.0 : 1.0
contains(accommodation) = True           pro : anti   =      3.8 : 1.0
 contains(agricultural) = True           pro : anti   =      3.8 : 1.0
    contains(bharat) = True              pro : anti   =      3.8 : 1.0
     contains(cards) = True              pro : anti   =      3.8 : 1.0
contains(constitution) = True            pro : anti   =      3.8 : 1.0
   contains(migrant) = False             pro : anti   =      3.8 : 1.0
   contains(minimum) = True              pro : anti   =      3.8 : 1.0
 contains(ministers) = True              pro : anti   =      3.8 : 1.0
```

The above results can be interpreted as follows:

1. If an article contains the word ***Infrastructure*** then the odds of ***PRO to ANTI*** are **6.2:1.0**. What this essentially means is that if that word is present in an article then it is almost **6** times more likely to be a **PRO** government article rather than having **ANTI** government sentiments.

- An example of a ***PRO*** government sentence correctly identified by our method is as follows - ***The Modi government has launched a Garib Kalyan Rojgar Abhiyan to provide livelihood opportunities through focused public infrastructure works in 116 districts with large returnee migrant worker populations.***.

2. If an article contains the word ***Bharat*** then it is almost **3.2** times more likely that the article is ***PRO*** government as opposed to ***ANTI*** government.

- Example of correct identification - ***Targeted and ecosystem support for migrant workers is a major thrust of the Atmanirbhar Bharat package.***
- Example of incorrect identification - ***This time those same old promises were packaged as Atma Nirbhar Bharat Abhiyan.***. Capturing these subtle language structures would indeed require further investigation into designing features.

3. The **SVM** classifier has performed quite poorly as compared to the **NaiveBayes** classifier in case of this data, giving an accuracy of about **42%** signifying the fact that the data is probably not linearly separable in higher dimensions.

Preparing the Twitter Data

In case of **Tweets** as the primary form of text data, we find that setting the proportion of negative/positive words might not be necessary since most Tweets have similar length, as opposed to article sentences, which come in a high degree of size variability. So basically, our primary feature characteristic in this regard is:

$$Score = num(Positivewords) - num(Negativewords)$$

Note that in this case, we treat the **hashtags** as tokens, since they also tend to carry key sentiment evidence. A similar process has been adopted here as well wherein we first extract the relevant **proper nouns** and the Tweets associated with them so as to assign class-labels.

1. Function to extact Key-subjects:

```
big_propers = []
for i in range(1339):
    text = nltk.word_tokenize(df['text'][i])
    tagged_news = nltk.pos_tag(text)
    big_propers.extend(nltk.FreqDist(word for (word,tag) in
                                     tagged_news if tag.startswith("NNP")
                                     and len(word)>1))
```

```
['Mumbai',
'Muzaffarpur',
'Samstipur',
'PMModi',
'AtmaNirbharUttarPradeshRojgarAbhiyan',
'AatmaNirbharBharatAbhiyan',...]
```

2. Example of the extracted Tweets:

- ***#PMModi launches Rs 50,000 crore campaign - #GaribKalyanRojgarAbhiyaan - to boost opportunities for #MigrantWorkers in rural India***
- ***I think #Lockdown is failing in India due to #MigrantLabour crisis created by #Modi.***
- ***Garib kalyan Rojgar Yojna, A prominent step taken by Modi Cabinet. It will surely engage lacs of migrants***

3. A Custom-Lexicon was created for the Tweets data. Some of the new words included were:

- ***sarkari*** –> ***Negative***
- ***wtf*** –> ***Negative***
- ***eyewash*** –> ***Negative***
- ***lockdownwithoutplan*** –> ***Negative***
- ***speakupsonia*** –> ***Negative***
- ***abdication*** –> ***Negative***

4. After defining the features and assigning class-labels, the following dataset was obtained:

```
head(gov_train1,5)
```

```
## # A tibble: 5 x 4
##   text                                                    num tn    label
##   <chr>                                                 <int> <chr> <chr>
## 1 #PMModi launches #AtmaNirbharUttarPradeshRojgarAbhiyan to p~     1 tn1   pro
## 2 #AatmaNirbharBharatAbhiyan in Uttar Pradesh will provide mo~     2 tn2   pro
## 3 GoI has launched Garib Kalyan Rojgar Abhiyaan on 20th June ~     3 tn3   pro
## 4 During the virtual launch of the #AtmaNirbharUttarPradeshRo~     4 tn4   pro
## 5 The massive rural public works scheme #GaribKalyanRojgarAbh~     5 tn5   pro
```

Classification on the Twitter Data

The following results were obtained after training a **Naive Bayes Classifier** that gave us a **65%** accuracy.

```
Most Informative Features
     contains(launch) = True              pro : anti    =      8.0 : 1.0
contains(migrantworkers) = True           pro : anti    =      8.0 : 1.0
  contains(launches) = True               pro : anti    =      6.1 : 1.0
   contains(provide) = True               pro : anti    =      5.2 : 1.0
  contains(lockdown) = True              anti : pro     =      3.8 : 1.0
contains(employment) = True               pro : anti    =      3.5 : 1.0
     contains(rural) = True               pro : anti    =      3.5 : 1.0
      contains(fund) = True              anti : pro     =      3.1 : 1.0
    contains(issues) = True              anti : pro     =      3.1 : 1.0
    contains(plight) = True              anti : pro     =      3.1 : 1.0
  contains(stranded) = True               pro : anti    =      2.9 : 1.0
    contains(ruling) = True              anti : pro     =      2.4 : 1.0
     contains(party) = True              anti : pro     =      2.3 : 1.0
contains(responsibility) = True          anti : pro     =      2.3 : 1.0
       contains(bus) = True               pro : anti    =      2.2 : 1.0
    contains(cities) = True               pro : anti    =      2.2 : 1.0
  contains(returned) = True               pro : anti    =      2.2 : 1.0
    contains(scheme) = True               pro : anti    =      2.2 : 1.0
    contains(worker) = True               pro : anti    =      2.2 : 1.0
```

We can gather the following insights from our model:

1. If a Tweet contains the word ***launch*** then it is almost **8** times more likely to be a **PRO** goverment Tweet as opposed to an **ANTI** government Tweet.

- Correctly identified as a positive Tweet - ***#PMModi to launch '#GaribKalyanRojgarAbhiyaan' on 20 June to boost livelihood opportunities in rural India.***.

2. If a Tweet contains the word ***lockdown*** then it is almost **3.8** times more likely to be an **ANTI** goverment Tweet as opposed to a **PRO** government Tweet.

- Correctly identified as a negative Tweet - ***We are electing leaders to manage systems and people in place to foresee the pro's and cons of every major government action and formulate policies. Who decided the lockdown without any thought to the migrant labour problems? PM should answer***.

3. Example of model predictions - **0** implies an **ANTI** government Tweet and **1** implies a **PRO** government Tweet:

- ***why so much anger madam? what's wrong talking? union government take responsibility whole country. you know what's happening state borders. don't runaway responsibility. #federal government #migrantlabour #migrantsontheroad #nirmalasithraman #rahulgandhi' –> 0 –> ANTI***

- ***pm totally silent intra country migrant labour crisis . that's total failure indian role state " #jairamramesh' –> 0 –> ANTI***
- ***the massive rural public works scheme #garibkalyanrojgarabhiyaan empower #migrantworkers ; rural citizens.' –> 1 –> PRO***

4. There are examples of incorrect classification here as well owing to the rather loose language used in Tweets and subjects not clearly specified. For example it presents quite a challenge to accurately measure the correct prediction for something like :

- ***600 train received UP govt migrants labour of various states including maharastra ...bengal is crying with 47 train migrants labour ...before amphan migrants.... tolabaji....?.... shame***

5. We feel that given the generalized nature of our features, it is quite hard to reach accuracy levels of around **90%** given the complex interplay of natural language, without **overfitting** the data.

Latent Semantic Analysis and KMeans (Articles)

In this section, we will discuss how documents and words were essentially **vectorized** and Linear Algebra algorithms like **SVD** were applied to the **Document-Term matrix** to give us the dimensionally reduced form of data. This is primarily done to visualize typically high dimensional data in 2-dimensions and identify similarities between documents and words. Basically document-word vectors are created using a **Tfidf** weighting. The process for achieving numeric valued vectors from textual data is given below:

1. First we count the number of unique terms across all the documents, essentially forming our ***Vocabulary***. Here is a demonstrative function that implements this procedure.

```
unique_words = []
for sent in doc_array:
    for word in sent:
        if word not in unique_words:
            unique_words.append(word)
```

2. Now we calculate the ***inverse document frequency (idf)*** for all the words. This essentially gives us the inverse log value of - the number of occurrences of a particular word across all documents. This measure is taken, to account for the extremely high occurrence of what are known in ***NLP*** as ***Stop_words***. These are words like -***is, of, by, to***- which typically have large values, so this score essentially tones these values down since they are not really important and do not carry much information. It is given by:

$$Idf_{word_i} = \log\left(\frac{num(corpus)}{num(docFrequency_{word_i})}\right)$$

Here is a ***function*** that achieves this:

```
doc_freq = {}
for voc in unique_words:
    doc_freq[voc] = 0
    for sent in doc_array:
        for word in sent:
            if word == voc:
                doc_freq[voc] += 1
    doc_freq[voc] = np.log(7/doc_freq[voc])
```

3. Now we compute the final score of a particular word as its ***term frequency*** - which is the measure of the number of occurrences of a particular word in a document - times its ***inverse document frequency***.

$$Score_{word_i,doc_j} = Tf_{word_i,doc_j} \times Idf_{word_i}$$

Here is a ***function*** that achieves this to get the final score:

```
tdm_array = {}

for s in range(len(doc_array)):
    name = "d"+str(s)
    tmp_array = {}
    for v in unique_words:
        tmp_array[v] = 0
        for word in doc_array[s]:
            if v == word:
                tmp_array[v] += 1
        tmp_array[v] = tmp_array[v] * doc_freq[v]
    tdm_array[name] = list(tmp_array.values())
```

4. After getting these scores, we essentially have our high dimensional ***Document-Term matrix***. We then move on to apply ***SVD*** on this that essentially re-expresses our data in terms of the ***top two Principal components*** - which are basically Eigendirections capturing most amount of variability in the data. In the below formula, ***A*** is our original matrix, ***U*** and ***V*** are Eigenvector matrices and ***Sigma*** denotes the matrix of ***Singular Values***. The subscript ***k*** refers to the effective rank of the dimensionally reduced matrix.

$$A_k = U\Sigma_k V^T$$

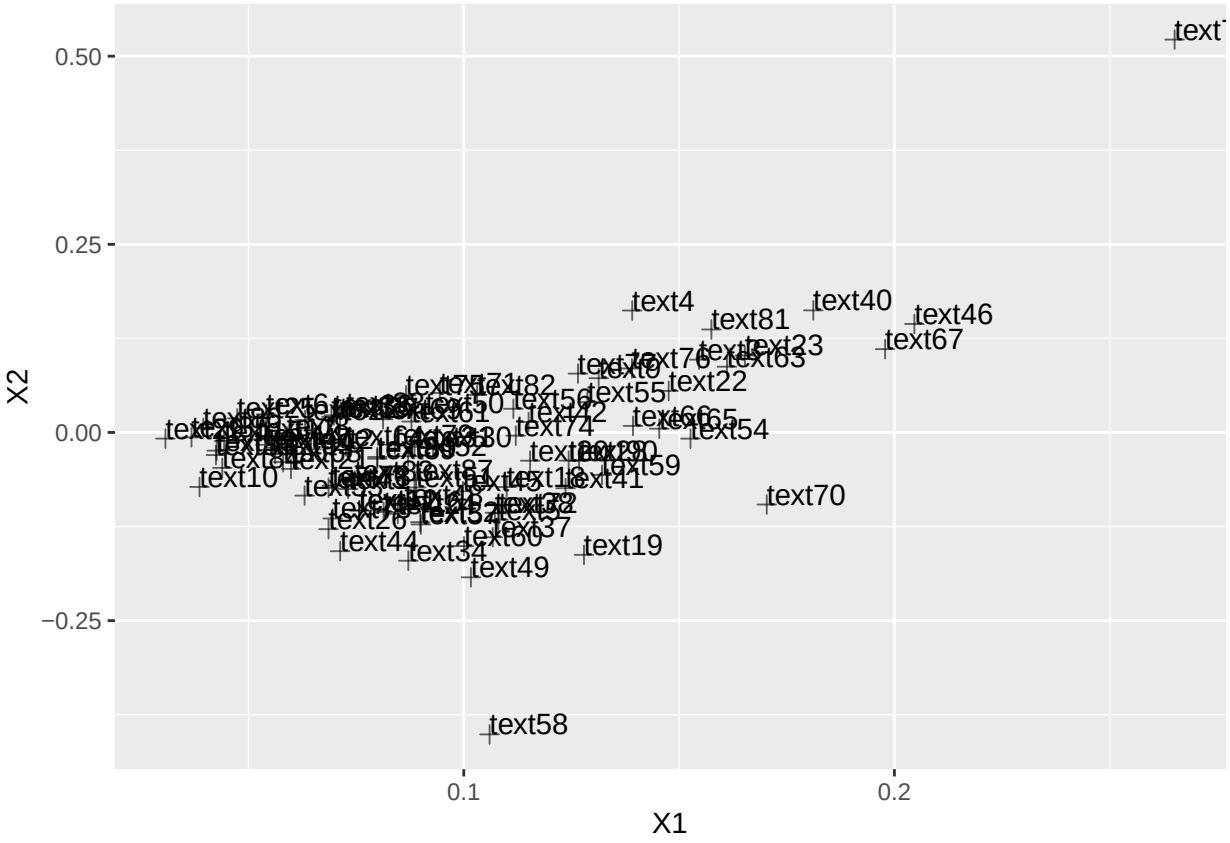
5. This is how our ***Document vectors*** look after ***SVD***. Note that these correspond to our ***Article*** data.

```
corp_vect[1:2,]
```

```
##              [,1]        [,2]
## text1 0.07314612 -0.07507370
## text2 0.06454998 -0.02237673
```

6. We can now plot all our ***Article vectors*** to visualize similarities between documents. The closer two documents are in this vector space essentially tells us their degree of similarity.

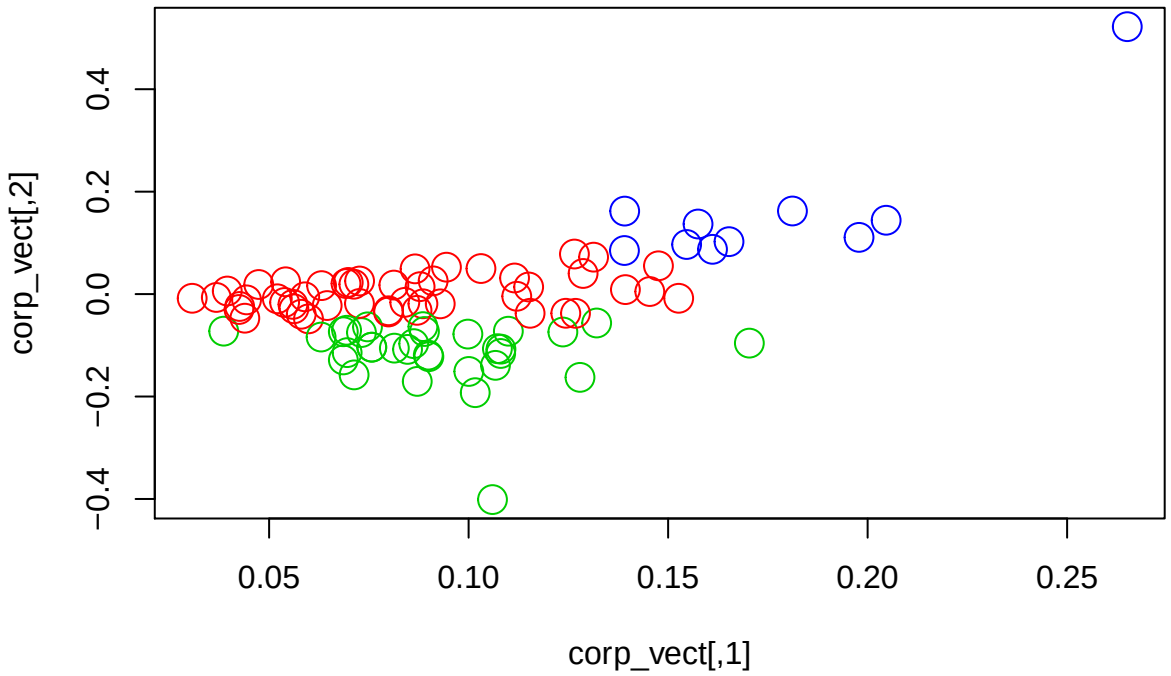
```
corp_transformed %>%
  ggplot(aes(x = X1, y = X2, label = .rownames)) +
  geom_point(pch = 3, size = 2, alpha = 0.6) +
  geom_text(aes(label=.rownames),hjust=0, vjust=0)
```



7. Upon an initial viewing of this data, we can somewhat see that 3 clusters might exist in this data. With this premise, we then move on to use ***Unsupervised clustering methods*** like the ***KMeans*** algorithm to essentially form centroid based clusters of the documents. We will attempt to ascertain as to how these clusters differ in characteristics. This is essentially a form of ***Topic modeling*** wherein key characteristics of clusters give us information on the various sub-topics of discussion within the larger topic of ***Migrant Labourers***.

8. ***KMeans*** clustering is hence performed with a preset of ***3*** as the number of clusters to be obtained. Here is how the data is separated now.

```
plot(corp_vect, col=(km_corp$cluster+1), cex = 2)
```



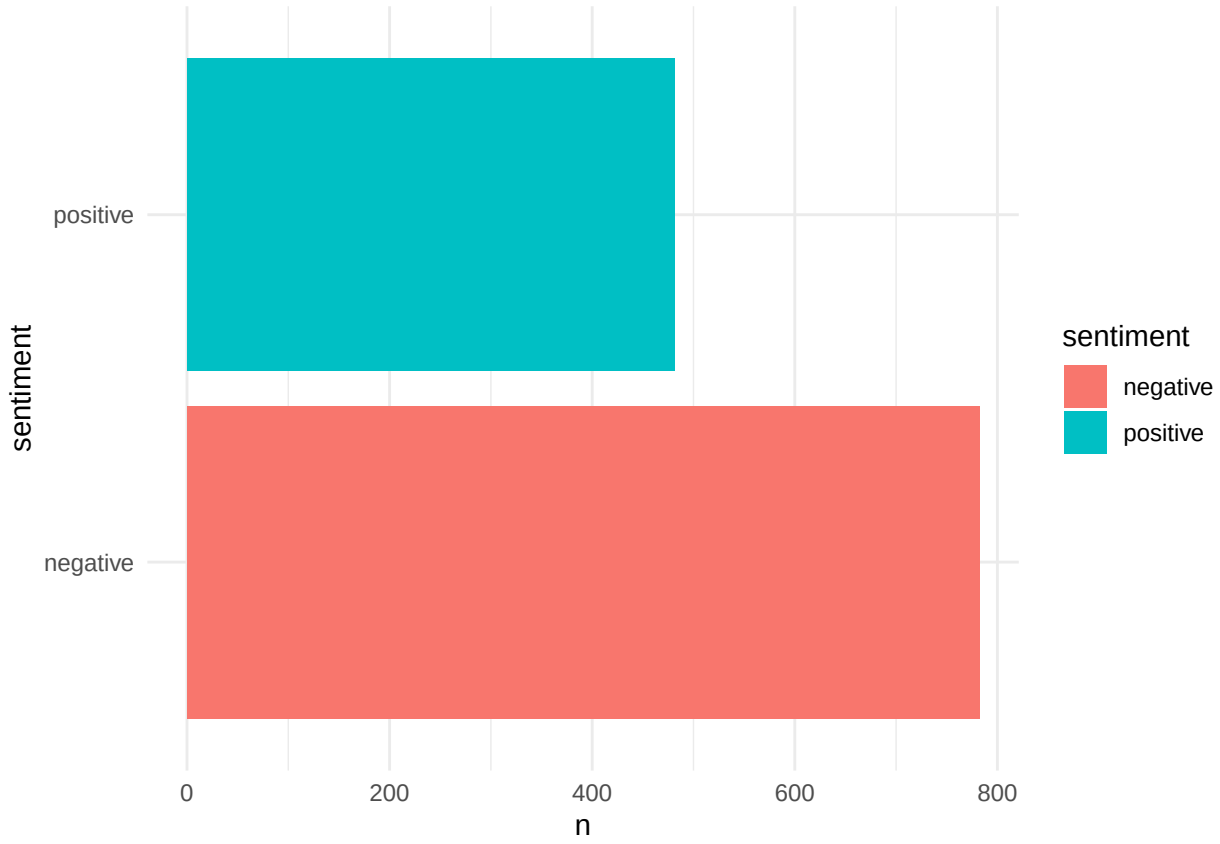
9. Even though ***KMeans*** is essentially a hard-clustering algorithm, we believe that the data actually exhibits a soft-clustering. This is because various words are common across documents in each cluster. Upon examining the defining words in each cluster, we found the following:

- ***Cluster 1*** contained words like - ***migrants, government, lockdown, trains, economic*** - implying a set of articles that discuss the Migrant issue primarily with respect to - Government imposed lockdowns, Economic Relief packages and the condition of the Migrants.
- ***Cluster 2*** contained words like - ***standard, scorching, marginal, help, food*** - implying a set of articles that discuss the Migrant issue primarily with respect to - Highlighting the sufferring of the Migrants.
- ***Cluster 3*** contained words like - ***host states, railways, contagion, special trains, atmanirbhar*** - implying a set of articles that discuss the Migrant issue primarily with respect to - Shramik Trains, seeing Migrant travel as a means of spreading Coronavirus, specific packages like AtmaNirbhar Yojana.

10. Lastly, we present the ***publication specific*** sentiment to see how do different publications differ in their general sentiment while writing about the Migrant issue.

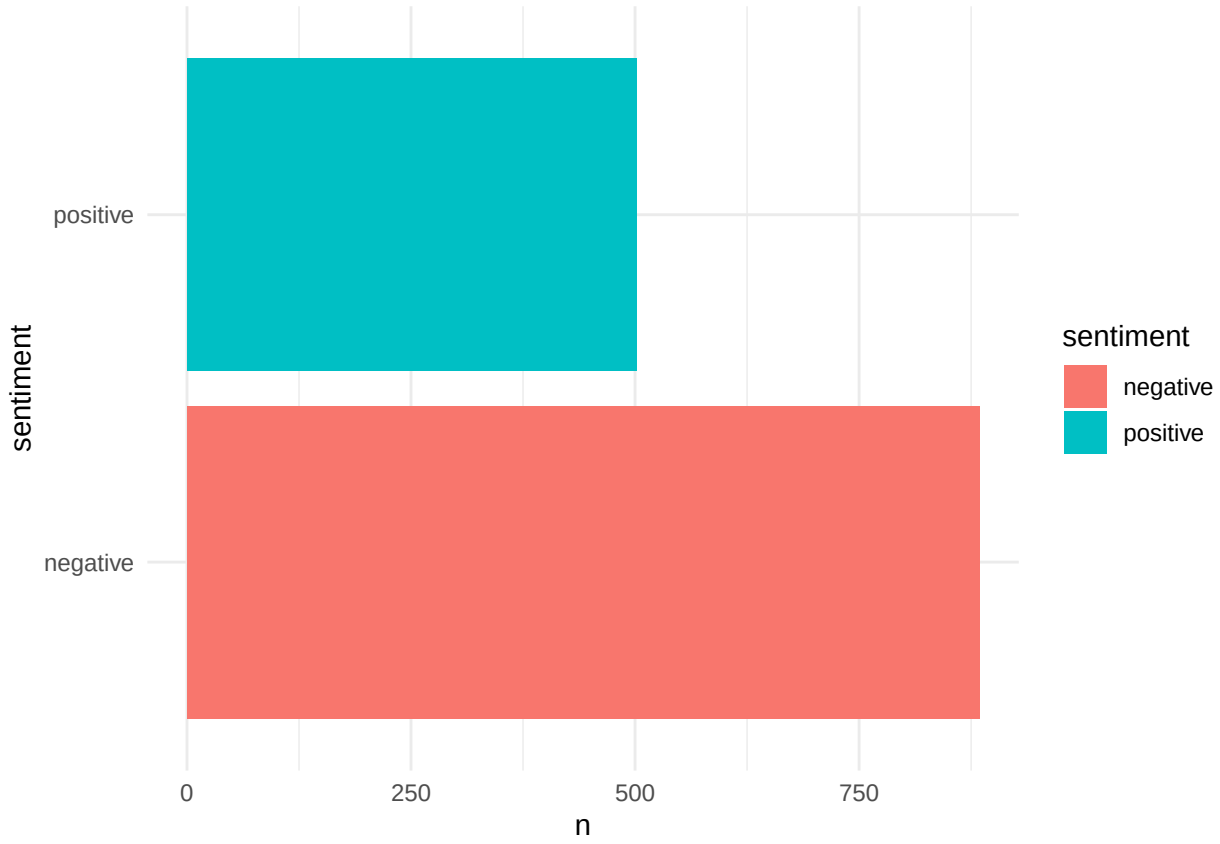
- The ***TOI*** sentiment is given by the below figure. The positive to negative ratio is **0.61**.

```
toi_sentiment %>%
  ggplot(aes(sentiment, n, fill = sentiment)) +
  geom_col() +
  theme_minimal() +
  coord_flip()
```



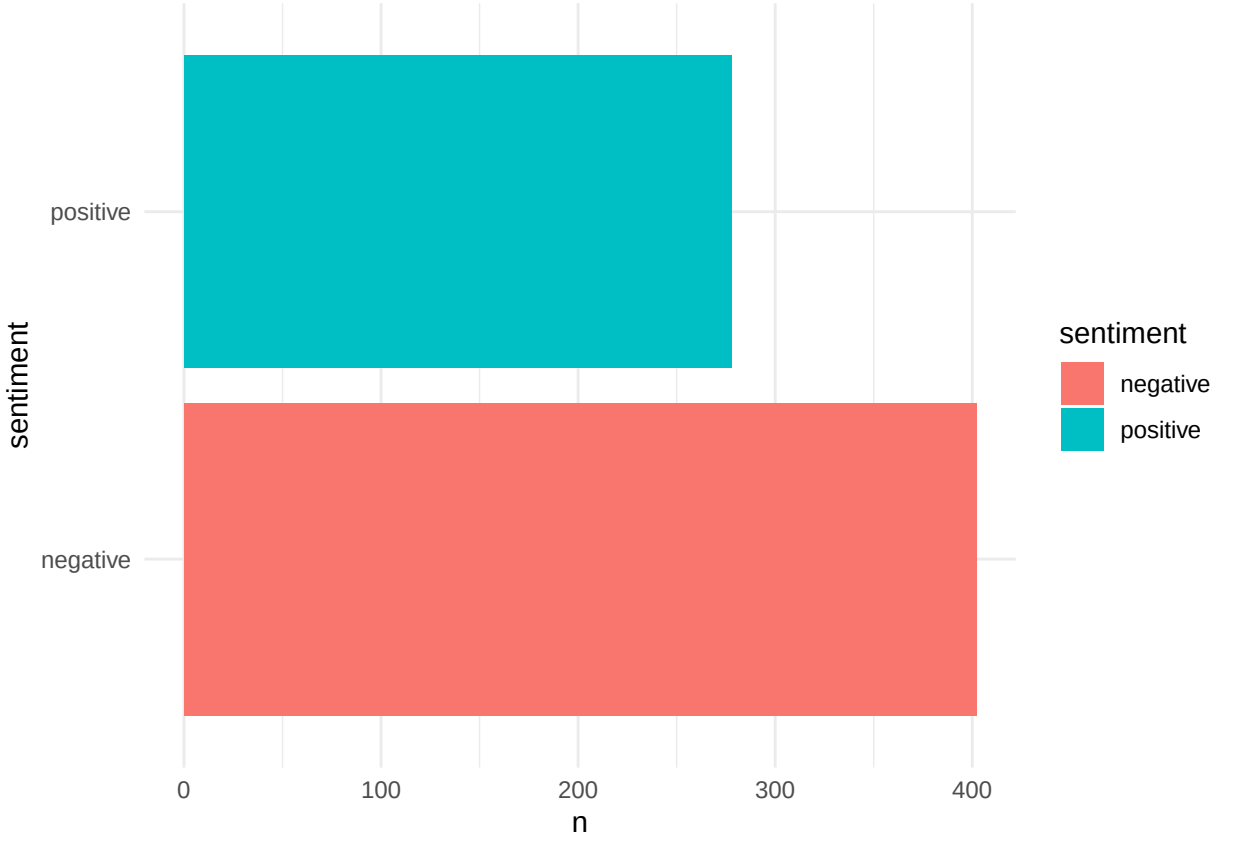
- The ***IE*** sentiment is given by the below figure. The positive to negative ratio is **0.56**.

```
ie_sentiment %>%
  ggplot(aes(sentiment, n, fill = sentiment)) +
  geom_col() +
  theme_minimal() +
  coord_flip()
```



- The ***HT*** sentiment is given by the below figure. The positive to negative ratio is **0.69**.

```
ht_sentiment %>%
  ggplot(aes(sentiment, n, fill = sentiment)) +
  geom_col() +
  theme_minimal() +
  coord_flip()
```



11. We can imply from the above sentiment counts is that - ***Even though the overall coverage has been negative in all three publications, HT has had the most positive number of articles, followed by TOI and then IE, which has the most negative coverage around the issue.***

Financial Analysis

With the exponential growth of unstructured textual data generated everyday in the form of Articles, Reviews and Social-media statements, we feel that **NLP** concepts are absolutely crucial to apply, in order to gain meaningful insights from such a vast pool of data. With this perspective, we demonstrate some more **NLP** concepts and gain meaningful insights from other categories of text data.

we now present another completely different dataset to the one we used previously. This is a dataset composed of a **Finance Corpus**. Articles have been mined from **The Economic Times** for the month of July. Articles pertaining to - **The Automobile Sector, The Healthcare Sector and The Finance/Banking Sector** - have been used. We will now see how concepts of **Sentiment analysis** and **Cosine Similarity measures** between word-vectors can be leveraged to get useful insights that might inform our **Investment decisions**.

The Auto dataset:

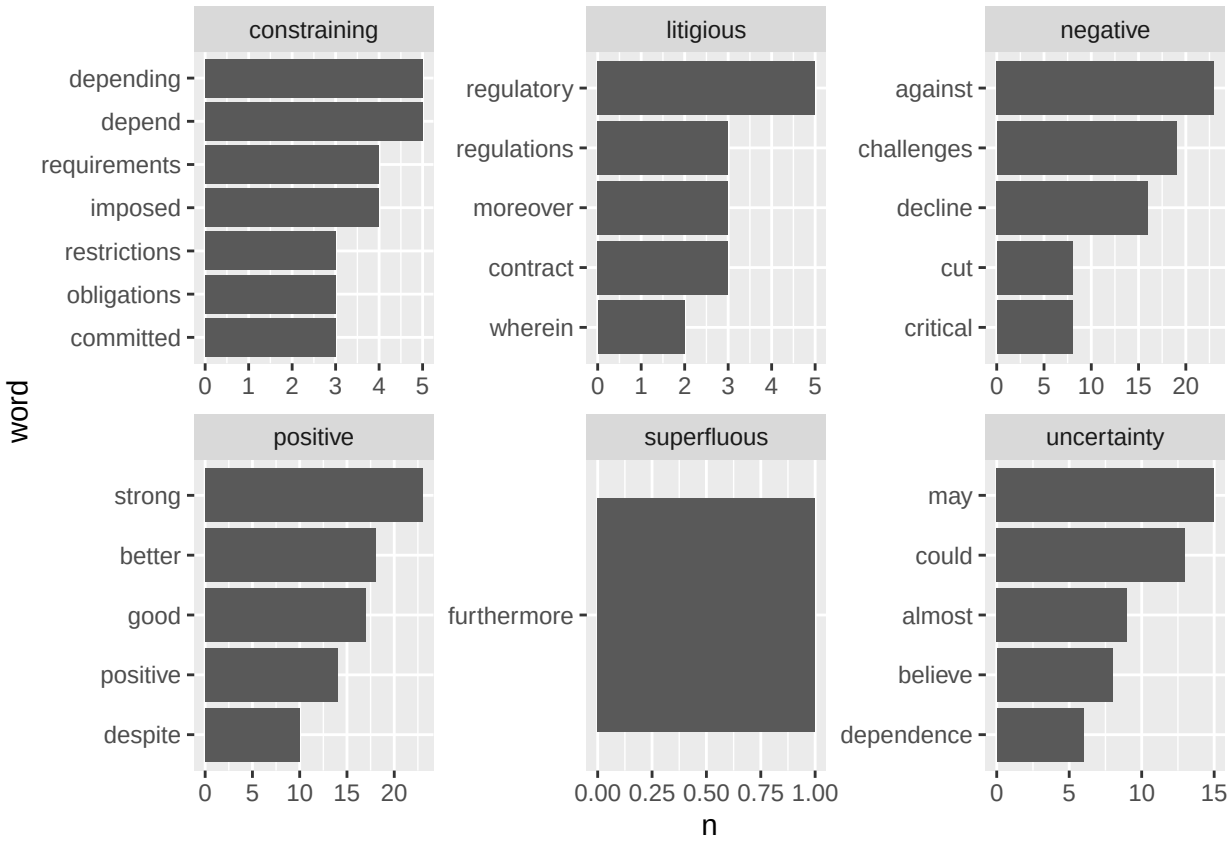
```
head(scauto, 5)
```

```
## # A tibble: 5 x 2
##   title   text
##   <chr>   <chr>
## 1 auto_et "MUMBAI: Top officials at the Department of Heavy Industry recently m~
## 2 auto_et "Mumbai: Demand for two-wheelers is recovering at a fast pace and mos~
## 3 auto_et "Passenger vehicle retail sales in June fell 38.34 per cent to 1,26,4~
## 4 auto_et "Mumbai: N Chandrasekaran, the chairman of Tata Sons, has stressed on~
## 5 auto_et "MUMBAI|NEW DELHI: Having assured employees of no salary cuts or job ~
```

1. For the purpose of Sentiment analysis on Financial articles we will utilize a Finance specific Lexicon known as ***Loughran***.

   - Some of the characteristic words in each sentiment category is presented below. We can immediately notice some of the characteristic words pertaining to each category:

```
scauto %>%
  unnest_tokens(word, text) %>%
  count(word) %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  group_by(sentiment) %>%
  top_n(5,n) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~ sentiment, scales = "free")
```



2. Word-count for various sentiment categories is presented below. Some of the key remarks that can be made are:

- The degree of ***uncertainty*** is **20%**.
- The degree of ***negativity*** is **35.5%**. Notice the high frequency of the word ***Decline*** in this Sector.
- The degree of ***positivity*** is **34.7%**. At the same time, we also notice a high frequency of the words ***Strong*** and ***Better***.

```
scauto %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments('loughran')) %>%
  count(sentiment)
```

```
## Joining, by = "word"
```

```
## # A tibble: 6 x 2
##   sentiment        n
##   <chr>        <int>
## 1 constraining    42
## 2 litigious       22
## 3 negative       233
## 4 positive       228
```

```
## 5 superfluous      1
## 6 uncertainty    130
```

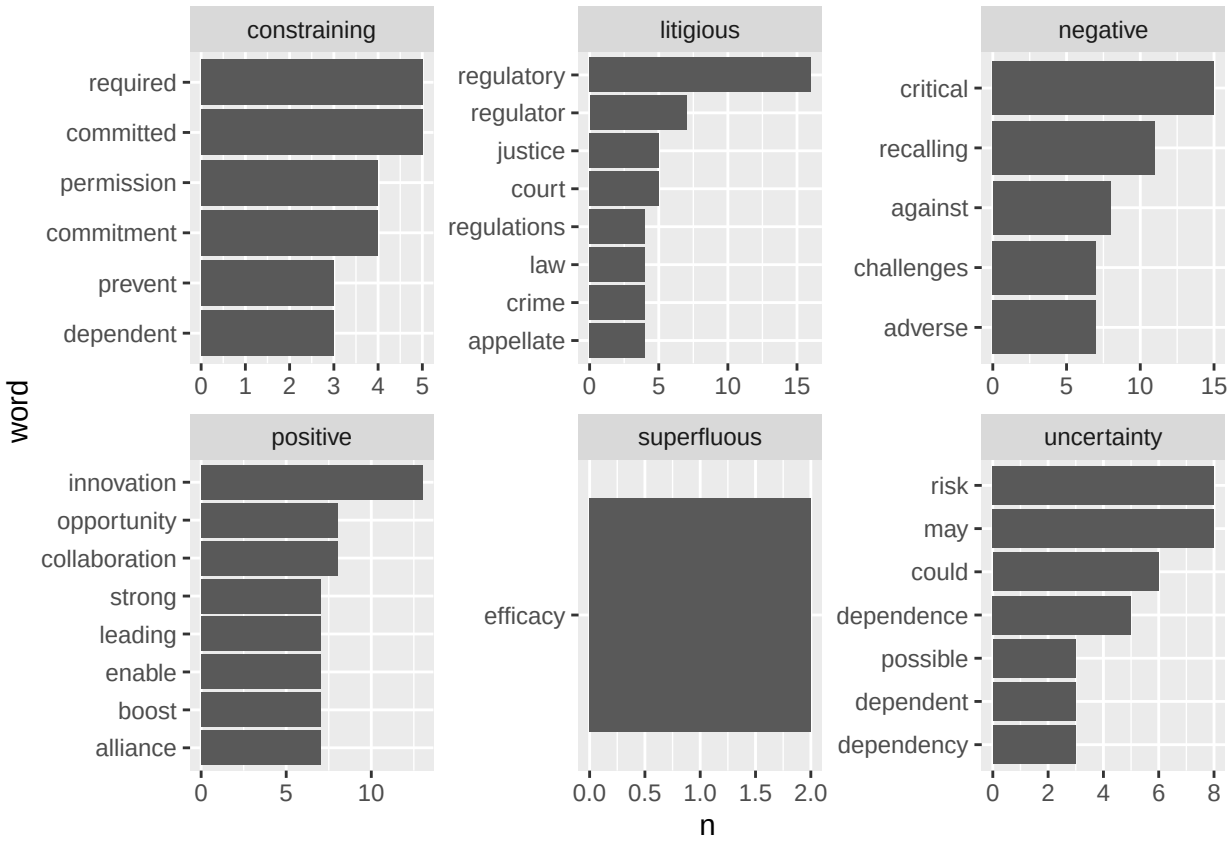
The Healthcare Dataset

```
head(scbio, 5)
```

```
## # A tibble: 5 x 2
##   title text
##   <chr> <chr>
## 1 bio   "HYDERABAD: The Coalition for Epidemic Preparedness Innovations (CEPI) ~
## 2 bio   "HYDERABAD: Vaccine manufacturer Bharat Biotech announced signing an ex~
## 3 bio   "NEW: Biotechnology is an asset heavy industry, said Kiran Mazumdar-Sha~
## 4 bio   "New Delhi: Drug major Lupin is recalling 35,928 bottles of a generic a~
## 5 bio   "LONDON: The international vaccine alliance GAVI has facilitated a new ~
```

1. Some of the characteristic words in each sentiment category is presented below. We can immediately notice some of the characteristic words pertaining to each category:

```
scbio %>%
  unnest_tokens(word, text) %>%
  count(word) %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  group_by(sentiment) %>%
  top_n(5,n) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~ sentiment, scales = "free")
```



2. Word-count for various sentiment categories is presented below. Some of the key remarks that can be made are:

- The degree of ***uncertainty*** is **11%**. Notice that it is lower than that of the ***Auto*** sector.
- The degree of ***negativity*** is **33.8%**. Notice the high frequency of the word ***Recalling*** in this Sector. Although this sector doesn't seem to have a degree of negativity as high as the ***Auto*** sector.
- The degree of ***positivity*** is **34.5%**. At the same time, we also notice a high frequency of the words ***Innovation*** and ***Collaboration***. this is possibly due to various research partnerships happening with respect to ***Covid-Vaccine*** development.

```
scbio %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments('loughran')) %>%
  count(sentiment)
```

```
## Joining, by = "word"
```

```
## # A tibble: 6 x 2
##   sentiment        n
##   <chr>        <int>
## 1 constraining    41
## 2 litigious       72
```

```
## 3 negative       193
## 4 positive       197
## 5 superfluous      2
## 6 uncertainty     66
```

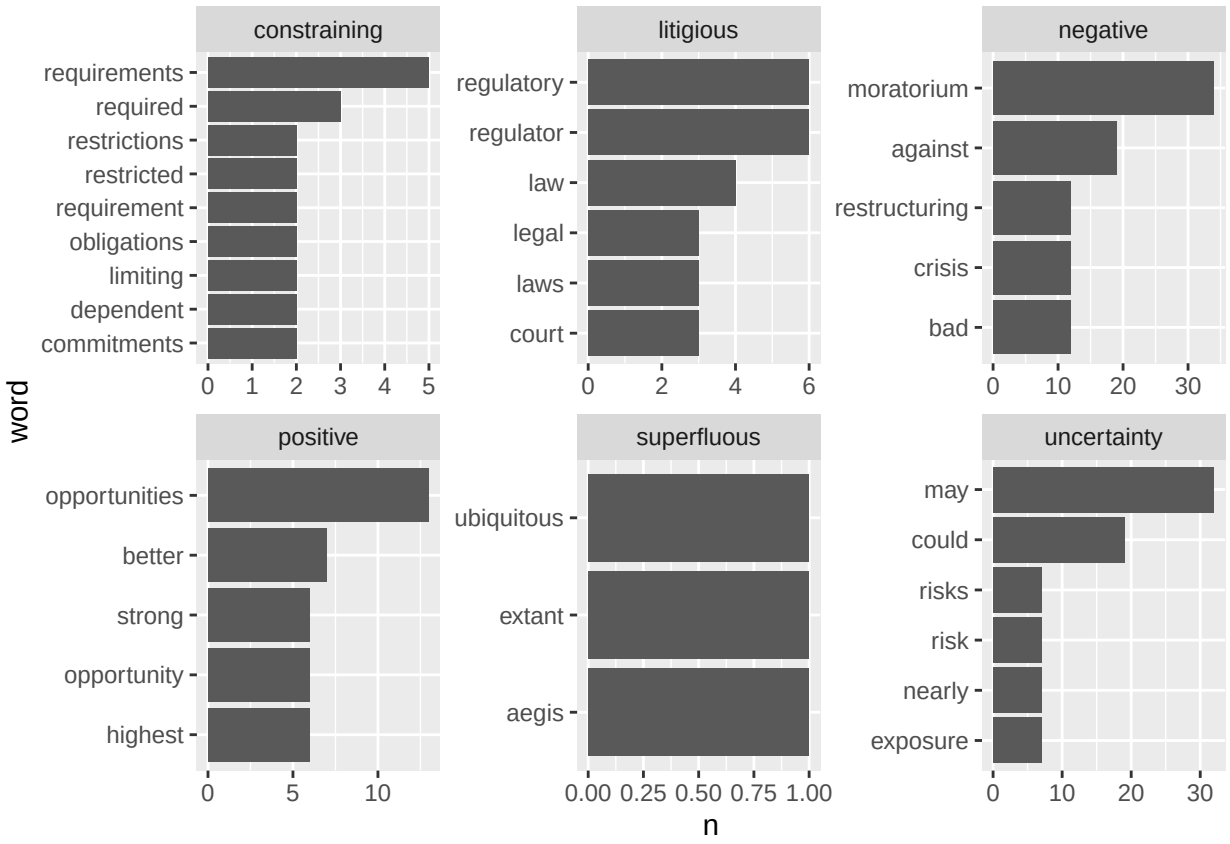
The Finance/Banking Dataset

```
head(scfin, 5)
```

```
## # A tibble: 5 x 2
##   title text
##   <chr> <chr>
## 1 fin   Mr. Rakesh Rathi dons many hats. He is a chartered account, entrepreneu~
## 2 fin   MUMBAI: Financial system loans under moratorium fell to 25% for the Jun~
## 3 fin   MUMBAI: Rural play fintech companies are on a revival path, aided by a ~
## 4 fin   New Delhi: Non-banking financial company Manappuram Finance on Tuesday ~
## 5 fin   The Covid-19 pandemic has been the black swan event that no one saw. Th~
```

1. Some of the characteristic words in each sentiment category is presented below. We can immediately notice some of the characteristic words pertaining to each category:

```
scfin %>%
  unnest_tokens(word, text) %>%
  count(word) %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  group_by(sentiment) %>%
  top_n(5,n) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~ sentiment, scales = "free")
```



2. Word-count for various sentiment categories is presented below. Some of the key remarks that can be made are:

- The degree of ***uncertainty*** is **19.3%**. Notice that it is lower than that of the ***Auto*** sector, albeit marginally.
- The degree of ***negativity*** is **47%**. This sector has the highest degree of ***negativity*** among all other sectors.
- The degree of ***positivity*** is **21.5%**. This sector has the lowest degree of ***positivity*** among all other sectors.

```
scfin %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments('loughran')) %>%
  count(sentiment)
```

```
## Joining, by = "word"
```

```
## # A tibble: 6 x 2
##   sentiment        n
##   <chr>        <int>
## 1 constraining    30
## 2 litigious       54
## 3 negative       335
```

```
## 4 positive       154
## 5 superfluous      3
## 6 uncertainty    138
```

Word similarities

Here we essentially **vectorize** our words using the previously described method and then attempt to discover their degree of similarity using the **Cosine-similarity** measure, which quantifies the angular separation of two vectors in a vector space, typically, the higher this value, the more correlated or similar are two words. Cosine similarity is given by:

$$Similarity = \frac{\vec{v_1} \cdot \vec{v_2}}{||\vec{v_1}|| \times ||\vec{v_2}||}$$

```
def cosine_simi(v1,v2):
    if norm(v1)>0 and norm(v2)>0:
        return(dot(v1,v2)/(norm(v1)*norm(v2)))
    else:
        return 0.0
```

- Here is a list of the top **60** words that seem to be quite similar to the word **TVS**. We can examine a few top meaningful correlations as:
- ***Salary Drop***
- ***Threatening***
- ***Fall***
- ***Cuts***

```
head(sort(cos_w_tweet[,1], decreasing = T), 60)
```

```
##           tvs         motor        option      mahindra           nil
##     1.0000000     0.6343084     0.4522907     0.4336492     0.4264266
##     townships unavailability      wheelers     fantastic            mg
##     0.4186089     0.4060296     0.3903382     0.3895612     0.3799053
##     kirloskar          lost        suzuki      produced         waluj
##     0.3750384     0.3749597     0.3738089     0.3727410     0.3722442
##          corp   responsible          shot           gst      question
##     0.3713505     0.3661443     0.3638162     0.3637873     0.3525057
##         super           smc          alto       hyundai concentration
##     0.3491450     0.3440414     0.3426737     0.3415848     0.3404233
##       manesar       offered     quantified     dependence         india
##     0.3390953     0.3350910     0.3325444     0.3317749     0.3312779
##         metro      solution       formats      tractors           map
##     0.3286048     0.3283325     0.3273242     0.3261730     0.3235997
##      finishes           tkm          uttar         total      reported
```

```
##      0.3226214      0.3212750      0.3198554      0.3182181      0.3169845
##           ebit    appointment          skoda  manufacturing       charging
##      0.3097221      0.3080405      0.3078407      0.3051608      0.3035005
##     compounded          plant      preferred           debt   unparalleled
##      0.3034324      0.3019378      0.2994243      0.2984708      0.2980109
##         medium     mitsubishi  bhattacharyya      weakening             rr
##      0.2966248      0.2959471      0.2939834      0.2937654      0.2933690
##         maruti        supreme      operators         adding    experiences
##      0.2921137      0.2910440      0.2905487      0.2902910      0.2873926
```

Insights on China

The final section of this report demonstrates the **NLP** concept of breaking down a document into **Ngrams** - a collection of ***n*** word phrases - and extracting relevant sentences on their basis, so as to gather Key insights from a large volume of documents, essentially without having to actually read through the documents. This process allows us to obtain relevant information regarding Key-subjects quite fast. The dataset for this purpose comprises Articles pertaining to **China** and have been mined from the international Publication - **The Economist**. Here is a view of the dataset:

```
head(data_xi, 5)
```

```
## # A tibble: 5 x 2
##      X1 sent_xi
##   <dbl> <chr>
## 1     0 Police in their home province of Henan, in central China, concluded tha~
## 2     1 The overturning of these verdicts is proof, officials say, that China's~
## 3     2 Many wrongful convictions have come to light after re-examination of ca~
## 4     3 Anti-crime campaigns on such a scale are rarer these days, but much abo~
## 5     4 A more common cause there is mistaken or deceitful testimony by witness~
```

N-gram insights

Here we break down the documents into **Ngrams**. Some of the results are presented below.

```
head(gram1, 5)
```

```
## # A tibble: 5 x 5
##      X1 word1         word2          word3     word4
##   <dbl> <chr>         <chr>          <chr>     <chr>
## 1     2 occasional    nationwide     strike    hard
## 2     3 criminal      system         remains   woeful
## 3    23 trump         administration hawks     undercut
## 4    28 confrontation german         officials shun
## 5    45 leaders       dismissed      american  promises
```

Based on certain **Ngrams** we can then extract relevant sentences to get key insights. Some examples are presented:

1. ***Now America has a president, Donald Trump, who shows no interest in Chinese repression and scorns global goods, but whose administration does include true China hawks who regard Communist Party rule as inherently immoral.***
   - This shows the duplicitous nature of American attitude towards China.

2. ***It accounts for almost 43% of the European Union's exports to China, and is duly wary of confrontation (German officials shun EU language calling China a "systemic rival")***
   - This shows the extent till which the German Economy is dependent on China's and why Germany tends to often come across as having a defensive attitude towards China.

Conclusion

To run through this report in summary, we have convered the applications of **NLP** concepts in a variety of domains that include - **Politics, Journalism, Finance, Social-Media**. The techniques provided by this field enable us to gain meaningful insights and train predictive models by utilizing the vast resource of text data available on the internet. Looking further, we can actually go into a deeper form of analysis using more sophisticated **NLP** techniques like - **Mixture Models, Latent Dirichlet Algorithm, Expectation Maximizing Algorithms**.

Technical Notes - NaiveBayes and Latent Semantic Analysis

Customized toy-programs have been implemented here, primarily relating to some of the core techniques utilized in this report - **The Naive Bayes Algorithm** and **Latent Semantic Analysis using SVD**.

Naive Bayes in a step-wise approach

This program illustrates the algorithm steps and functions associated with **training** and **applying** a NaiveBayes Classifier. The theoretical underpinning is as follows :

1. We want to find out the probability of a document **d** belonging to class **c**. Therefore by the Bayes Theorem formulation we get the following proportionality relation:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

2. Here the right most term in the above equation refers to the probability of term **t** occurring in a document, given class label **c**.
3. Our aim is to find the class label **c** that maximizes the following expression:

$$c_{map} = argmax_{c \in C} \left(\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c) \right)$$

4. Below is an illustraion in **Python** that will walk us through the mechanincs of implementing a Naive Bayes algorithm, step-by-step.

Document list - Data

Here is a list of the example documents on which we will train our model. These documents are pre-labeled as being **pro** or **anti**.

```
docs = [("the GST is good","pro"),
        ("bad law is IBC","anti"),
        ("the government is working hard to implement GST","pro"),
        ("the government did a great job by bringing GST","pro"),
        ("IBC is is harmful for us","anti"),
        ("opposition condemns IBC","anti")]
```

Vocabulary

Here we extract the vocabulary from the documents, basically collecting all the unique words into a list. Note that **D** signifies the set of all **documents**.

$$V \leftarrow ExtractVocab(D)$$

```
vocab = []
for dp in docs:
    for word in nltk.word_tokenize(dp[0]):
        if word not in vocab:
            vocab.append(word)
```

```
['the',
 'GST',
 'is',
 'good',
 'bad',
 'law',
 'IBC',
 'government',
 'working',
 'hard',
 'to',
 'implement',
 'did',
 'a',
 'great',
 'job',
 'by',
 'bringing',
 'harmful',
 'for',
 'us',
 'opposition',
 'condemns']
```

Further we define the following variables as well. Note that **V** signifies the set of **vocabulary** terms. Basically what we get from the below equations is that - ***B is the number of vocabulary terms and N is the number of documents in our corpus.***

$$B \leftarrow num(V)$$
$$N \leftarrow num(D)$$

```
B = len(vocab)
N = len(docs)
```

```
lab = ["pro","anti"]
labels = {}
```

Obtaining the Prior

Now we will count the number of documents in each class.

$$\forall c \in C, N_c \leftarrow CountDocsInClass(D, c)$$

```
for l in lab:
    counter = 0
    for d in docs:
        if l == d[1]:
            counter += 1
    labels[l] = counter
```

Calculate **class priors** by dividing number in each class by the total number of documents.

$$Prior(c) \leftarrow \frac{N_c}{N}$$

```
for k,v in labels.items():
    priors[k] = v/N
```

```
{'pro': 0.5, 'anti': 0.5}
```

Concatenate class text

Now we will concatenate all the text documents of each class essentially under one list.

$$Text_c \leftarrow ConcatTextOfAllDocsInClass(D, c)$$

```
concat_text = {}
for i in lab:
    i_list = []
    for doc in docs:
        if i == doc[1]:
            i_list.append(doc[0])

    concat_text[i] = ' '.join(i_list)
```

```
{'pro': 'the GST is good the government is working hard to implement GST the   governmen
 'anti': 'bad law is IBC IBC is is harmful for us opposition condemns IBC'}
```

Obtaining the Likelihood

In order to get the likelihood of a term being generated, given a particular class label, we find the conditional probability:

$$P(t_k|C = c)$$

The estimate of this probability is found by counting the number of occurrences of term **t** in all the documents belonging to class **c**. This is the formula: We add **1** in the numerator and **B** in the denominator to not get **0** probability since later on we will the **log** of this value. This method of adding these constants is known as **Laplace smoothing**.

$$CondProb(t|c) \leftarrow \frac{T_{ct} + 1}{(\sum_{t'} T_{ct'}) + B}$$

1. First we get the counts of each term in the ***pro*** class label documents:

```
for t in vocab:
    counter = 0
    for word in nltk.word_tokenize(concat_text["pro"]):
        if t == word:
            counter += 1
    count_pro[t]= counter
```

```
{'the': 3,
 'GST': 3,
 'is': 2,
 'good': 1,
 'bad': 0,
 'law': 0,
 'IBC': 0,
 'government': 2,
 'working': 1,
 'hard': 1,
 'to': 1,
 'implement': 1,
 'did': 1,
 'a': 1,
 'great': 1,
 'job': 1,
 'by': 1,
 'bringing': 1,
 'harmful': 0,
 'for': 0,
 'us': 0,
 'opposition': 0,
```

```
 'condemns': 0}
```

2. Then we get the counts of each term in the ***anti*** class label documents:

```
for t in vocab:
    counter = 0
    for word in nltk.word_tokenize(concat_text["anti"]):
        if t == word:
            counter += 1
    count_anti[t]= counter
```

```
{'the': 0,
 'GST': 0,
 'is': 3,
 'good': 0,
 'bad': 1,
 'law': 1,
 'IBC': 3,
 'government': 0,
 'working': 0,
 'hard': 0,
 'to': 0,
 'implement': 0,
 'did': 0,
 'a': 0,
 'great': 0,
 'job': 0,
 'by': 0,
 'bringing': 0,
 'harmful': 1,
 'for': 1,
 'us': 1,
 'opposition': 1,
 'condemns': 1}
```

3. We will then count the number of terms in each class. This is basically our expression:

$$(\sum_{t'} T_{ct'})$$

```
total_pro = 0
total_anti = 0
for v in count_pro.values():
    total_pro += v
for v in count_anti.values():
    total_anti += v
```

4. Finally we apply the above formula for conditional probabilty and get the conditional probabilities of each term with respect to each class.

```
for term, value in count_pro.items():
    prob_pro[term] = (value + 1)/((total_pro) + B)
for term, value in count_anti.items():
    prob_anti[term] = (value + 1)/((total_anti) + B)
```

5. We can now take a look at the conditional probabilities of all the words, given the class ***pro***:

   {'the': 0.0909, 'NRC': 0.0909, 'is': 0.0681, 'good': 0.0454, 'bad': 0.0227, 'law': 0.0227, 'CAA': 0.0227, 'government': 0.0681, 'working': 0.0454, 'hard': 0.0454, 'to': 0.0454, 'implement': 0.0454, 'did': 0.0454, 'a': 0.0454, 'great': 0.0454, 'job': 0.0454, 'by': 0.0454, 'bringing': 0.0454, 'harmful': 0.0227, 'for': 0.0227, 'us': 0.0227, 'opposition': 0.0227, 'condemns': 0.0227}

Apply Naive Bayes

We start by getting a new document and tokenizing it.

```
newdoc = "the GST is a decent measure by the government"
```

```
newtokens = nltk.word_tokenize(newdoc)
```

Maximum A-Posteriori prediction

The prediction as per **MAP** is given by the following equation. This equation essentially tells us to get that value of class label **c** that maximizes the probability of document **d** being in class **c**.

$$c_{map} = argmax_{c \in C} \left(\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c) \right)$$

Implement

1. To implement the above formulation, we will first initialize our **class score** for the new document with the **log priors** of each class.

$$Score(c) \leftarrow \log(Prior(c))$$

```
scores = {}

for l in lab:
    scores[l] = math.log(priors[l])
```

2. Then we will loop through all the new document tokens and keep appending the **log likelihood** value to the overall **score** for each class:

$$Score(c) += \log(CondProb(t|c))$$

```
for t in newtokens:
    if t in prob_pro.keys():
        scores["pro"] += math.log(prob_pro[t])
    if t in prob_anti.keys():
        scores["anti"] += math.log(prob_anti[t])
```

3. Finally we compare the **scores** thus obtained for each class and assign the document that class label for which the **score** is higher.

$$argmax_{c \in C} Score(c)$$

```
if scores["pro"] > scores["anti"]:
    print("the class label for the new document is: PRO")
else:
    print("the class label for the new document is: ANTI")
```

```
the class label for the new document is: PRO
```

4. As we can see, the document has been **correctly labeled** as **PRO** since we can see from the training corpus that the word **GST** tends to be highly associated with the **PRO** category.

Demonstration for Latent Semantic Indexing

```
import pandas as pd
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import numpy as np
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot
from sklearn.metrics.pairwise import cosine_similarity
import math
import spacy
import re
```

Below is a list of documents. We will attempt to vectorize these documents as per TfIdf weighting and then perform Latent Semantic Indexing in order to display the documents in dimensionally reduced form.

```
doc1 = "xi jinping is an authorotative leader of china"
doc2 = "it is said that xi jinping is ruthless and harmful for chinas future"
doc3 = "the indian border with china is quite a dangerous area with many threats"
doc4 = "the indian prime minister is having talks with his chinese counterpart on certai
doc5 = "indian economy is doing well now"
doc6 = "sound financial policy is necessary for growth"
doc7 = "india and china have strained relations"
```

```
doc_dict = {}
for i in range(1,8):
    doc_dict["doc"+str(i)] = eval("doc"+str(i))
```

```
doc_array = []
for doc in doc_dict.values():
    doc_array.append(doc.split())
```

This code block creates the Vocabulary by counting the number of unique words across all the documents.

```
unique_words = []
for sent in doc_array:
```

```
    for word in sent:
        if word not in unique_words:
            unique_words.append(word)
```

```
len(unique_words)
```

```
50
```

```
len(doc_dict.keys())
```

```
7
```

This function creates a dictionary that keeps the term count for each word in each document.

```
tdm_array = {}

for s in range(len(doc_array)):
    name = "d"+str(s)
    tmp_array = {}
    for v in unique_words:
        tmp_array[v] = 0
        for word in doc_array[s]:
            if v == word:
                tmp_array[v] += 1
        tmp_array[v] = tmp_array[v] * doc_freq[v]
    tdm_array[name] = list(tmp_array.values())
```

This function calculates the IDF value for each word. This is then used in the previously given function to compute the TfIdf weight of a word in each document.

```
doc_freq = {}
for voc in unique_words:
    doc_freq[voc] = 0
    for sent in doc_array:
        for word in sent:
            if word == voc:
                doc_freq[voc] += 1
    doc_freq[voc] = np.log(7/doc_freq[voc])
```

```
serie = pd.Series(tdm_array, index = tdm_array.keys())
num = serie.to_numpy()
```

This code is to essentially convert the Dataframe object to a Matrix.

```
nn = np.zeros([7,50])
for i in range(7):
    nn[i] = np.asarray(num[i])
```

SVD is performed below.

```
u, s, vt = np.linalg.svd(nn)
```

```
s_d = np.diag(s)
```

We have selected from the Sigma matrix, the top two singular values which also depicts the effective rank of the Document-Term matrix.

```
sigma = s_d[:,:2]
sigma
```

```
array([[6.50467178, 0.        ],
       [0.        , 5.81635768],
       [0.        , 0.        ],
       [0.        , 0.        ],
       [0.        , 0.        ],
       [0.        , 0.        ],
       [0.        , 0.        ]])
```

```
vtnew = vt[:2,:]
```

```
array([[-3.44456777e-03, -3.44456777e-03, -3.31183271e-18,
        -3.60312066e-03, -3.60312066e-03, -3.60312066e-03,
        -3.60312066e-03, -4.91779128e-02, -1.74730840e-03,
        -1.74730840e-03, -1.74730840e-03, -1.74730840e-03,
        -3.23335696e-03, -1.74730840e-03, -1.20585658e-03,
        -1.74730840e-03, -1.74730840e-03, -2.48209631e-01,
        -1.72432025e-01, -1.06064167e-01, -2.14057911e-01,
        -1.06064167e-01, -1.06064167e-01, -1.06064167e-01,
```

```
        -1.06064167e-01, -1.06064167e-01, -1.06064167e-01,
        -2.79478551e-01, -2.79478551e-01, -2.79478551e-01,
        -2.79478551e-01, -2.79478551e-01, -2.79478551e-01,
        -2.79478551e-01, -2.79478551e-01, -2.79478551e-01,
        -2.79478551e-01, -1.04658685e-02, -1.04658685e-02,
        -1.04658685e-02, -1.04658685e-02, -1.25742305e-04,
        -1.25742305e-04, -1.25742305e-04, -1.25742305e-04,
        -1.25742305e-04, -3.27504802e-03, -3.27504802e-03,
        -3.27504802e-03, -3.27504802e-03],
       [-2.52879505e-01, -2.52879505e-01, -6.09177542e-19,
        -7.09102659e-02, -7.09102659e-02, -7.09102659e-02,
        -7.09102659e-02, -5.57681382e-02, -3.21886143e-01,
        -3.21886143e-01, -3.21886143e-01, -3.21886143e-01,
        -2.29690005e-01, -3.21886143e-01, -2.31630171e-01,
        -3.21886143e-01, -3.21886143e-01, -6.73058154e-03,
        -4.73408880e-03, -2.22770903e-02, -1.42521851e-02,
        -2.22770903e-02, -2.22770903e-02, -2.22770903e-02,
        -2.22770903e-02, -2.22770903e-02, -2.22770903e-02,
         1.18225133e-02,  1.18225133e-02,  1.18225133e-02,
         1.18225133e-02,  1.18225133e-02,  1.18225133e-02,
         1.18225133e-02,  1.18225133e-02,  1.18225133e-02,
         1.18225133e-02, -4.17764193e-04, -4.17764193e-04,
        -4.17764193e-04, -4.17764193e-04, -3.79037872e-02,
        -3.79037872e-02, -3.79037872e-02, -3.79037872e-02,
        -3.79037872e-02, -3.48901371e-02, -3.48901371e-02,
        -3.48901371e-02, -3.48901371e-02]])
```

```
final = u.dot(s_d[:,:2])
final
```

```
array([[-7.83441905e-02, -1.23278841e+00],
       [-3.79924722e-02, -5.59605158e+00],
       [-2.30619846e+00, -3.87291436e-01],
       [-6.07682142e+00,  2.05536633e-01],
       [-2.27563845e-01, -7.26290964e-03],
       [-2.73406860e-03, -6.58964522e-01],
       [-7.12107668e-02, -6.06571644e-01]])
```

Below are the dimensionally reduced Document vectors.

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components = 2)
```

```
svd.fit(nn)
result = svd.transform(nn)
print(result)
```

```
[[ 7.83441905e-02  1.23278841e+00]
 [ 3.79924722e-02  5.59605158e+00]
 [ 2.30619846e+00  3.87291436e-01]
 [ 6.07682142e+00 -2.05536633e-01]
 [ 2.27563845e-01  7.26290964e-03]
 [ 2.73406860e-03  6.58964522e-01]
 [ 7.12107668e-02  6.06571644e-01]]
```

Below we have plotted the documents on a 2-dimensional axis.

```
import matplotlib.pyplot as plt
plt.figure(figsize = (20,10))
plt.scatter(result[:,0],result[:,1])
```

```
<matplotlib.collections.PathCollection at 0x1a1e9e3b90>
```

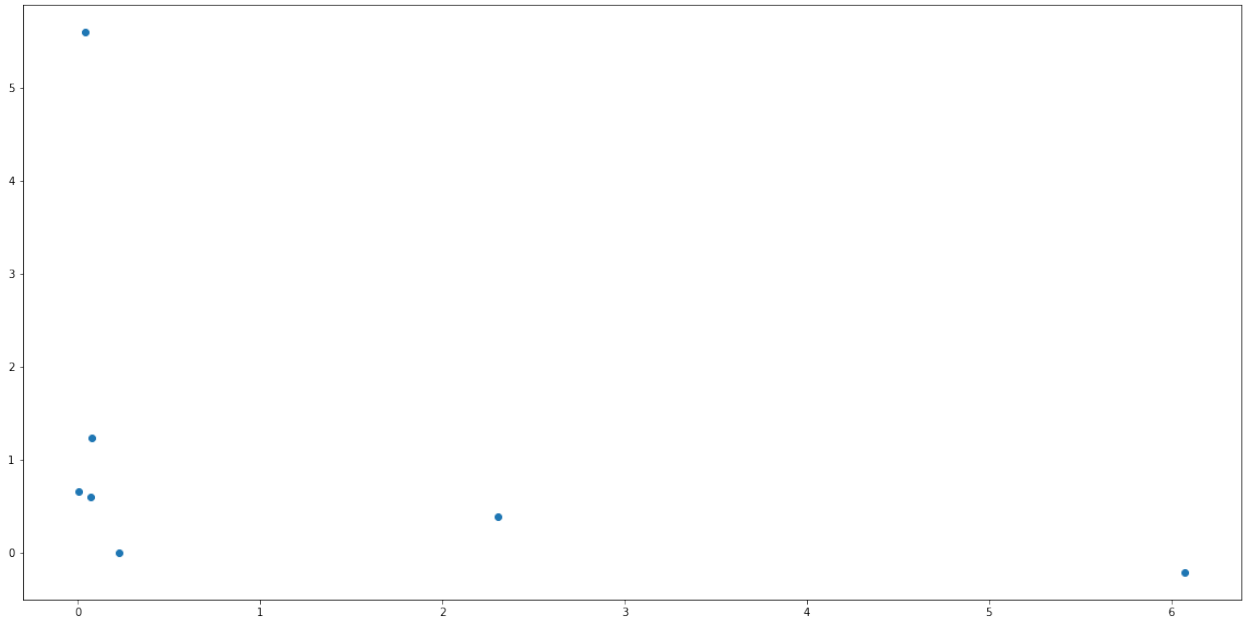


Figure 1: png

Finally, we present the cosine similarity measures for the reduced form documents.

```
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(result,result))
```

```
[[ 1.          0.99839435  0.22782906  0.02965054  0.09522563  0.99824132
   0.99857462]
 [ 0.99839435  1.          0.1723073  -0.02701782  0.03868449  0.99999652
   0.99394788]
 [ 0.22782906  0.1723073   1.          0.98002829  0.99097153  0.16970615
   0.27947408]
 [ 0.02965054 -0.02701782  0.98002829  1.          0.99784153 -0.0296568
   0.08295823]
 [ 0.09522563  0.03868449  0.99097153  0.99784153  1.          0.03604629
   0.14822078]
 [ 0.99824132  0.99999652  0.16970615 -0.0296568   0.03604629  1.
   0.99365441]
 [ 0.99857462  0.99394788  0.27947408  0.08295823  0.14822078  0.99365441
   1.        ]]
```