Exploration of NYT with NLP

Agenda

- About NYT Dataset
- Introducing the Problem
- Data Cleaning and Preparation
- Exploratory Data Analysis on comments and articles
- Sentiment Analysis
- Text Summarization
- Auto-Text Generation

Our Dataset

The New York Times

Articles and comments in New York Times between Jan-May 2017 and Jan-April 2018

- Over 2 million comments with 34 features
  - Features: approveDate, commentBody, commentID, commentSequence, commentTitle, commentType, createDate, depth, editorsSelection, inReplyTo, replyCount, recommendations, etc.

- Over 9,000 articles with 16 features
  - Features: articleID, abstract, byline, documentType, headline, keywords, multimedia, newDesk, printPage, pubDate, sectionName, snippet, source, typeOfMaterial, webURL, articleWordCount

Introducing the Problem

1. Too much textual data → Text Summarization
2. Texts have a huge influence on society → Sentiment Analysis
3. Guessing the intended future statements → Auto-Text Generation

Our Solution: Natural Language Processing (NLP)

Explaining the Process

Data Preparation and Cleaning

- Web Scraping
- Text Cleaning
- Corpus + Document Term Matrix

Exploratory Data Analysis

- Visualization
- Insights
- Trends + Patterns

Build Models

- Sentiment Analysis
- Predicting NYT's Pick
- Text Summarization
- Auto-Text Generation

Data Preparation and Cleaning

Web Scraping

- Libraries: requests, BeautifulSoup
  - pip install beautifulsoup4


```
import requests
from bs4 import BeautifulSoup

def url_to_text(url):
    html = requests.get(url).text
    soup = BeautifulSoup(html)

    for script in soup(["script", "style"]):
        script.extract()

    text = soup.get_text()

    lines = (line.strip() for line in text.splitlines())
    chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
    text = '\n'.join(chunk for chunk in chunks if chunk)

    return text
```

Text Cleaning

- Module: re (regular expressions)

```
import re

def clean_text_data(text):
    text = text.lower()
    text = re.sub("[^a-z ]+", "", text)
    return text
```

Before:

'With New Congress Poised to Convene, Obama's Policies Are in Peril - The New York Times\nSectionsSEARCHSkip to contentSkip to site indexPoliticsToday's PaperPolitics|With New Congress Poised to Convene, Obama's Policies Are in Perilhttps://nyti.ms/2iT3jXLAdvertisementContinue reading the main storySupported byContinue reading the main storyWith New Congress Poised to Convene, Obama's Policies Are in PerilSenator Mitch McConnell of Kentucky, the majority leader, last month. The new Congress will convene on Tuesday.Credit...Mark Wilson/Getty ImagesBy Jennifer SteinhauerJan. 1, 2017WASHINGTON — The most powerful and ambitious Republican-led Congress in 20 years will convene Tuesday, with plans to leave its mark on virtually every facet of American life — refashioning the country's social safety net, wiping out scores of labor and environmental regulations and unraveling some of the most significant policy prescriptions put forward by the Obama administration.Even before Preside

After:

with new congress poised to convene obamas policies are in peril the new york timessectionssearchskip to contentskip to site indexpoliticstodays paperpoliticswith new congress poised to convene obamas policies are in perilhttpsnytimsitjxladvertisementcontinue reading the main storysupported bycontinue reading the main storywith new congress poised to convene obamas policies are in perilsenator mitch mcconnell of kentucky the majority leader last month the new congress will convene on tuesdaycreditmark wilsongetty imagesby jennifer steinhauerjan washington the most powerful and ambitious republicanled congress in years will convene tuesday with plans to leave its mark on virtually every facet of american life refashioning the countrys social safety net wiping out scores of labor and environmental regulations and unraveling some of the most significant policy prescriptions put forward by the obama administrationeven before presidentelect donald j trump is sworn in on jan givin

Corpus + Document-Term Matrix

Corpus:

newDesk	text
Arts&Leisure	6 Ways 'Girls' Changed Television. Or Didn't. ...
Automobiles	Opinion | When Flamethrowers Like Ann Coulter ...
BookReview	Is It Possible for a Writer to Be Objective? -...
Business	Megyn Kelly's Jump to NBC From Fox News Will T...
Climate	Americans Are Staying Home More. That's Saving...
Culture	'The Affair' Season 3, Episode 6: Noah Goes Ho...
Dining	Swedish Meatballs, From the Comfort of Home - ...
EdLife	Wanted: Factory Workers, Degree Required - The...
Editorial	Opinion | Why Corporations Are Helping Donald ...
Express	Michigan State President Lou Anna Simon Resign...
Foreign	The Afghan War and the Evolution of Obama - Th...
Games	Little Troublemakers - The New York Times\nSec...

Document-Term Matrix:

newDesk	aa	aaa	aaalike	aaarated	aabar	aabc	aabowl	aacm
Arts&Leisure	0	0	1	0	0	0	0	0
Automobiles	0	0	0	0	0	0	0	0
BookReview	3	0	0	0	0	0	0	0
Business	1	0	0	0	0	0	0	0
Climate	0	0	0	0	0	0	0	0
Culture	1	3	0	0	0	0	0	3
Dining	0	0	0	0	0	0	0	0
EdLife	0	0	0	0	0	0	0	0
Editorial	0	0	0	0	0	0	0	0
Express	0	0	0	0	0	0	0	0
Foreign	0	0	0	0	0	0	0	0
Games	4	1	0	0	0	0	0	0
Insider	1	0	0	0	0	0	0	0

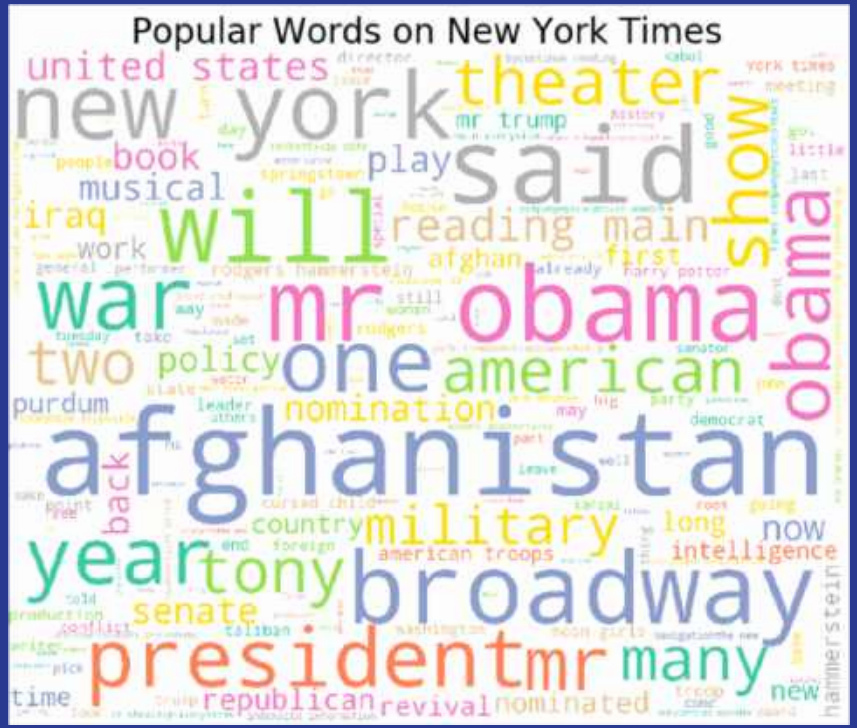
Exploratory Data Analysis

On articles

Visualization

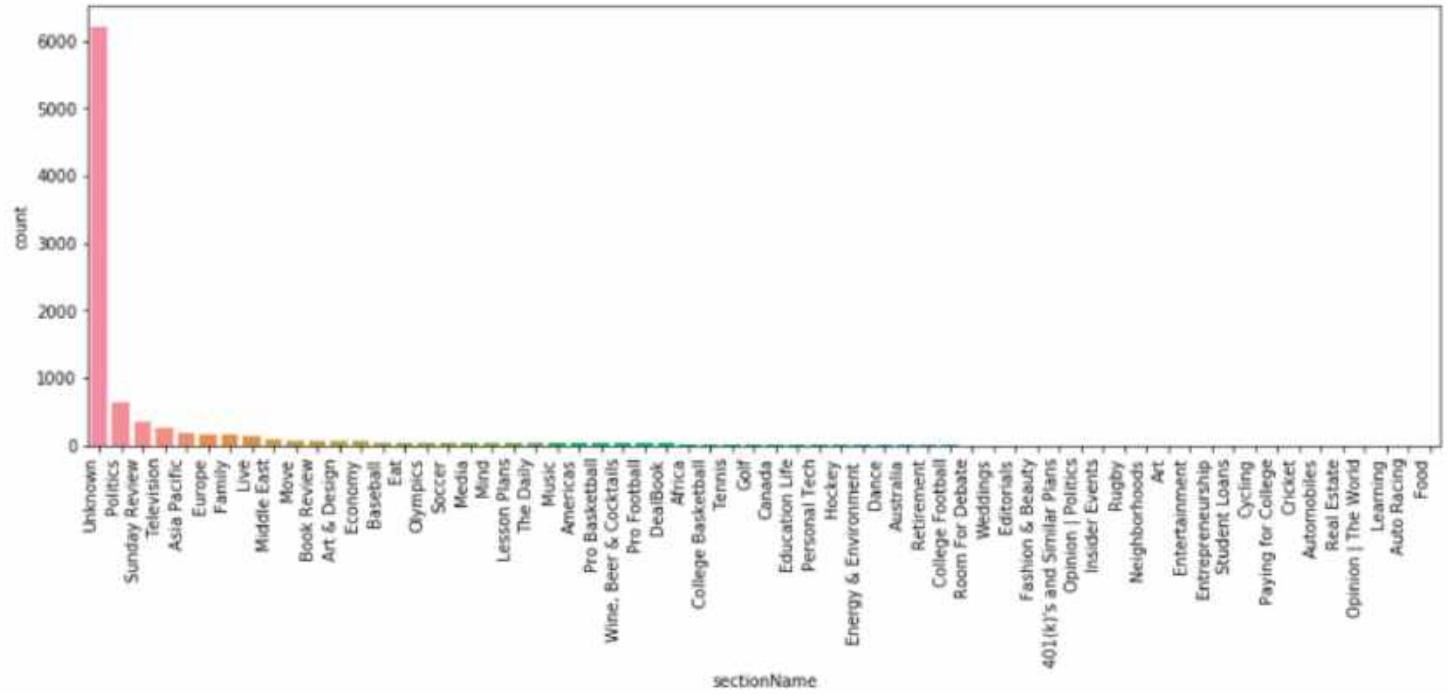
Matplotlib and WordCloud!

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

plt.figure(figsize=(15,10))
wc = WordCloud(width=1500, height=1200,
               background_color='white', colormap='Set2')
wc.generate(str(df_articles['text'].values))
plt.title('Popular Words on New York Times', fontsize=28)
plt.imshow(wc)
plt.axis('off')
plt.show()
```



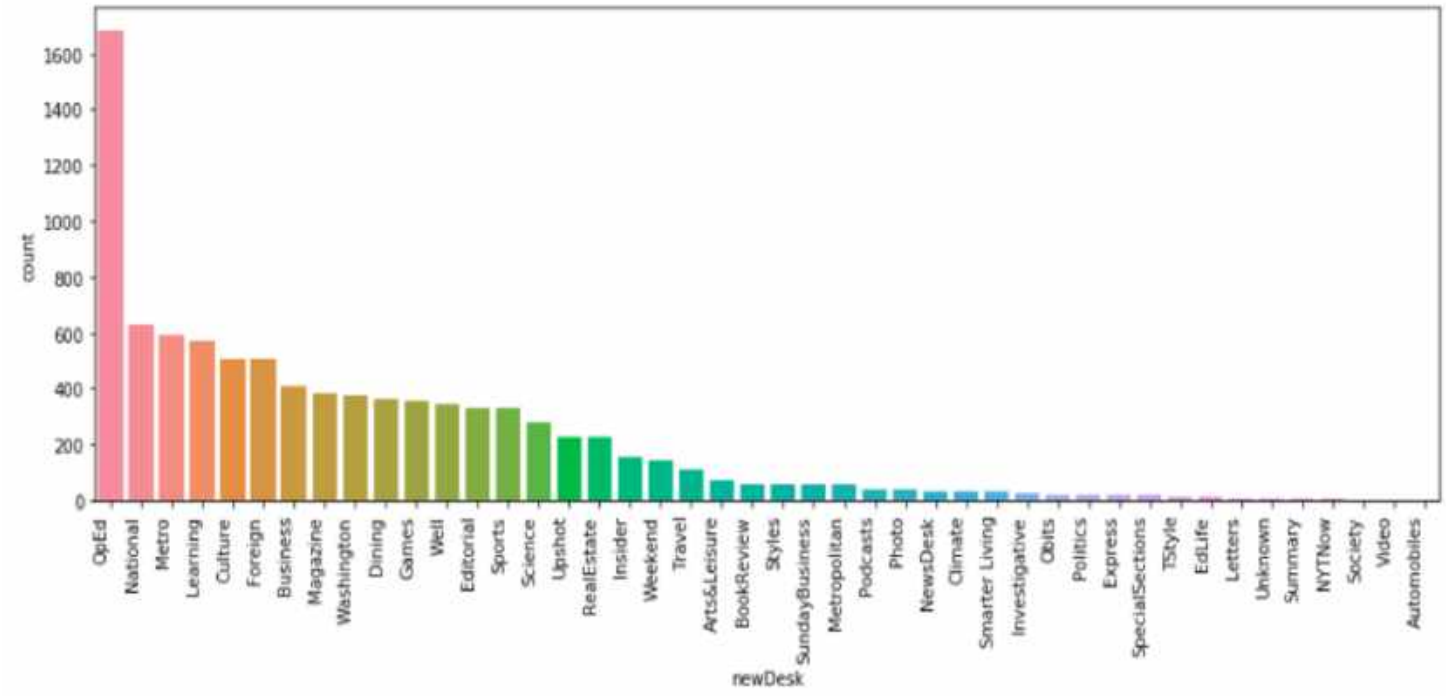
Visualizing sectionName

- Most articles are in the “Unknown” category of sectionName



Visualizing newDesk

- Better distribution of values in newDesk, most are in the “OpEd” category
  → Analysis of text grouped by newDesk



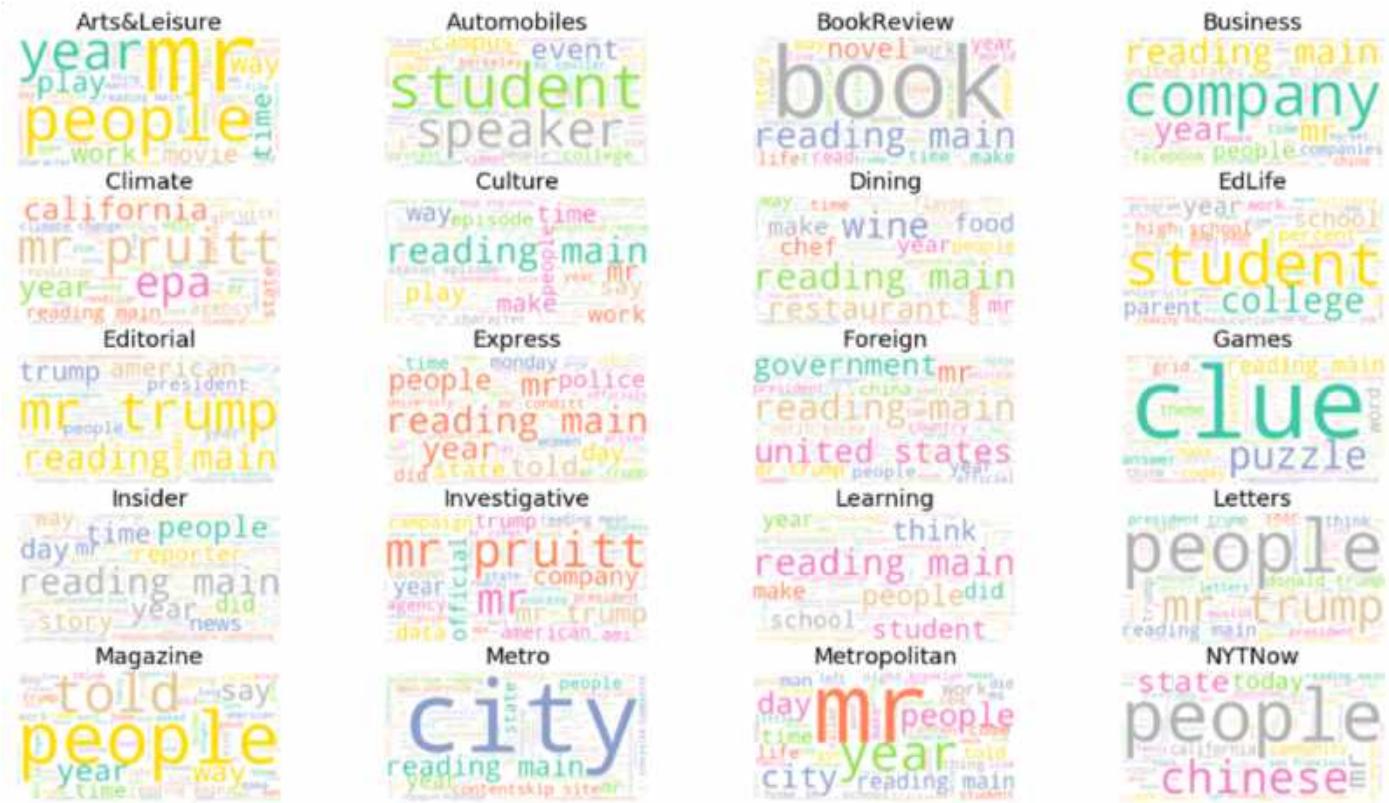
More WordClouds

- Define stop words using sklearn.feature_extraction.text.ENGLISH_STOP_WORDS
  - I.e. ‘a’, ‘i’, ‘it’, ‘me’, ‘my’, ‘the’, etc.

```
from sklearn.feature_extraction import text

stop_words = text.ENGLISH_STOP_WORDS

plt.rcParams['figure.figsize'] = [20, 25]

wc = WordCloud(stopwords=stop_words, background_color="white",
               colormap="Set2", max_font_size=150, random_state=42)

for i in range(len(df_articles_newDesk)):
    wc.generate(df_articles_newDesk.text[i])
    plt.subplot(11, 4, i+1)
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(df_articles_newDesk.index[i], fontsize=18)

plt.show()
```

Text still needs more cleaning! → Words such as “like” or “said” that are high in frequency but provide little insight should be removed for further analysis

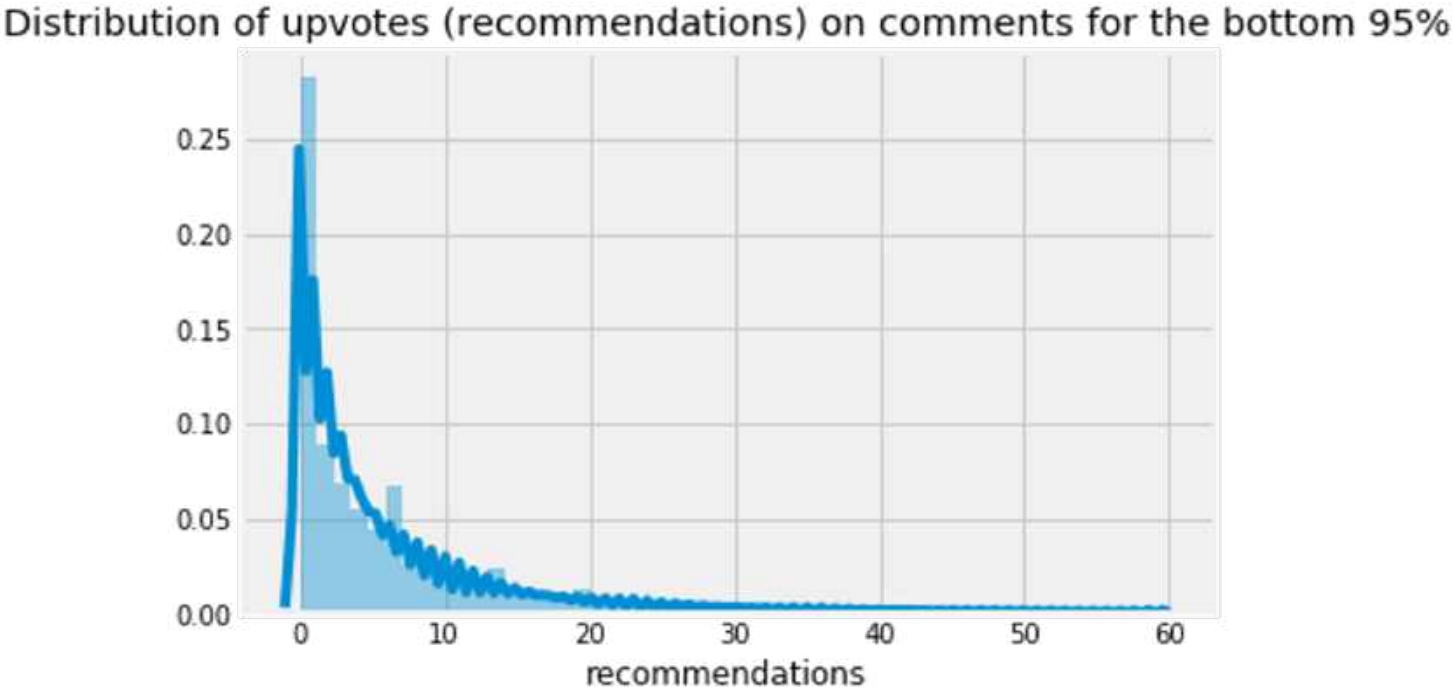
Do the resulting WordClouds make sense? What insight can we gain from them?



Exploratory Data Analysis

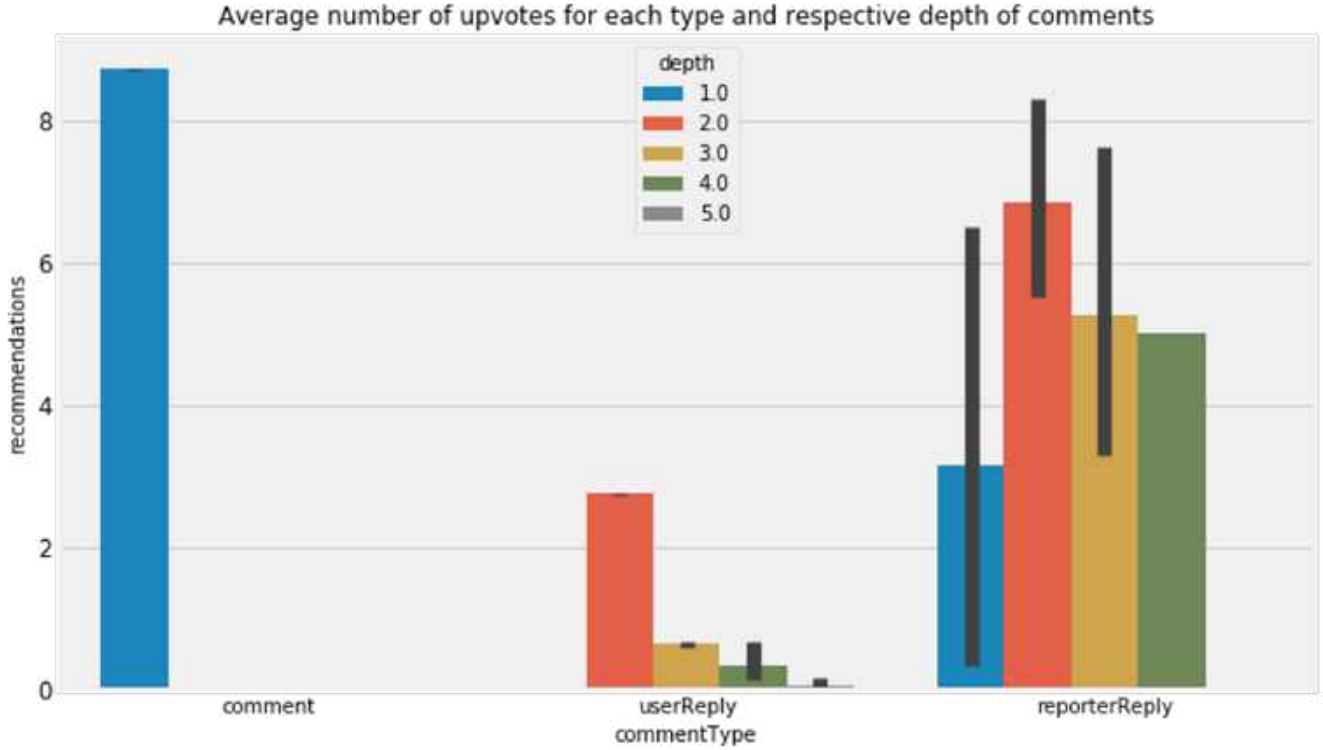
On comments

Visualizing Distribution of upvotes on comments



Upvotes for each comment type and depth of comments

Even though ‘comment’ has more upvotes, ‘reporterReply’ has more depth in the comments



WordCloud for comments

Most common words in the comments selected as Editor`s pick

Sentiment Analysis

Sentiment Analysis

What is it?

Seeks to quantify the emotional intensity of words and phrases within a text.

Where is it used?

To help businesses make sense of unorganized data.

positive

neutral

negative

Natural Language Toolkit (NLTK)

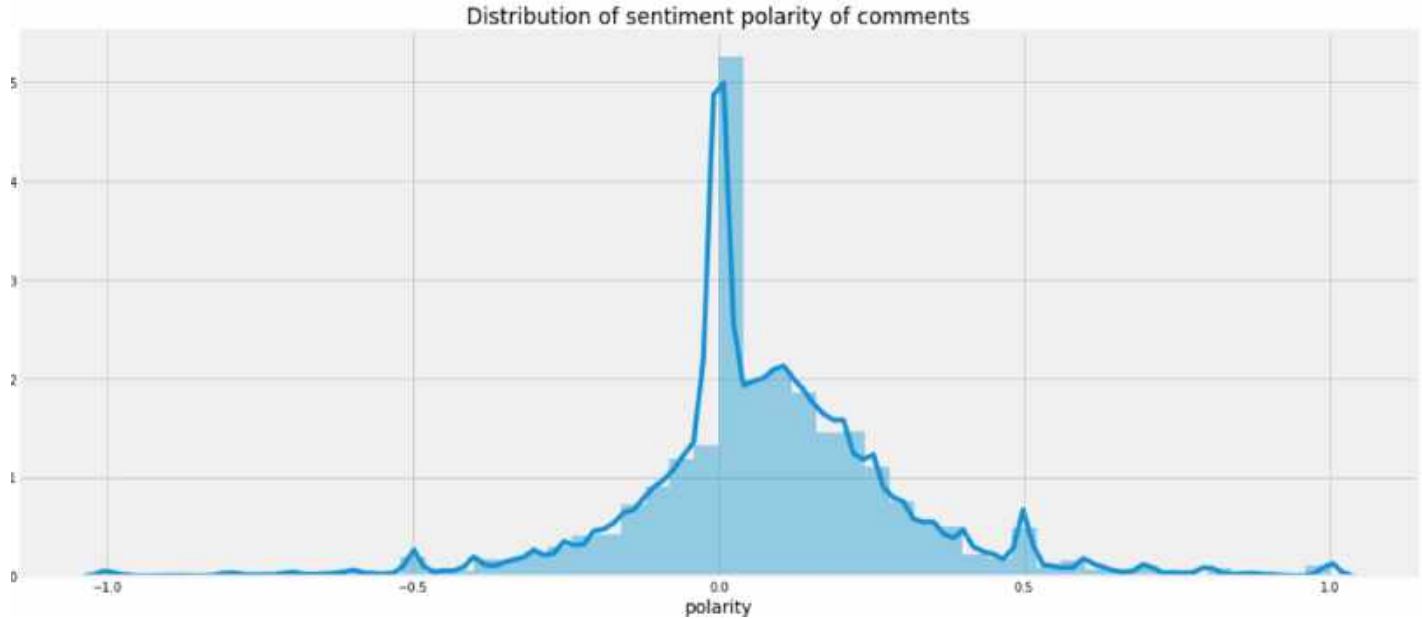
TextBlob:

- Input: corpus
- Scores:
  polarity (scale of -1 to 1)
  subjectivity (scale of 0 to 1)

VADER (Valence Aware Dictionary and sentiment Reasoner)

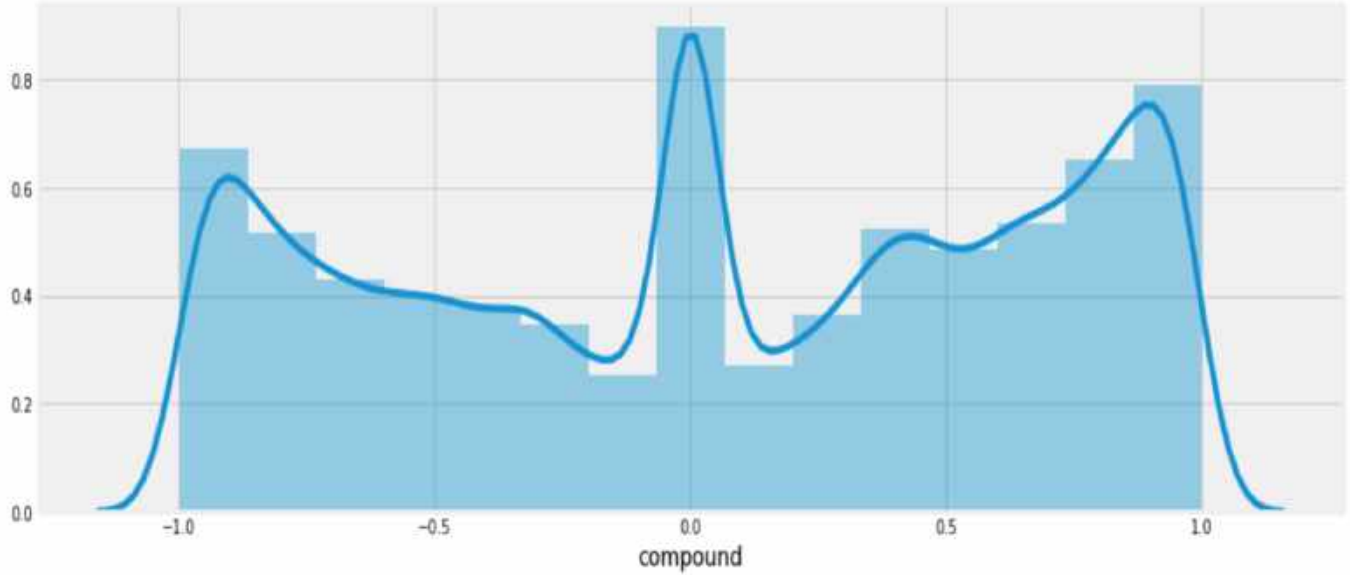
- Input: corpus
- Scores:
  compound (scale of -1 to 1)
  positive (scale of 0-1)
  neutral (scale of 0-1)
  negative (scale of 0-1)

Results: TextBlob



Conclusion: The tone of the comments here is tending towards positive side.

Results: VADER



Conclusion: The comments have balanced sentiments (negative and positive)

What affects the scores?

- **Punctuations** - The use of an exclamation mark(!), increases the magnitude of the intensity. Eg: *The food here is good!!!*
- **Capitalization-** Using upper case letters to emphasize a sentiment-relevant word in the presence of other non-capitalized words, increases the magnitude.
- **Conjunctions** - Use of conjunctions like “but” signals a shift in sentiment polarity. Eg: T*he food here is great but the service is horrible*.

Predicting NYT's Editor's pick

Steps

- Importing modules and data
- Balancing the classes to some extent by undersampling the majority class.
- Obtaining and combining the tf-idf vectors for words and character n-grams using TFIDFVectorizer and FeatureUnion.
- Training the first classifier that uses Logistic Regression coupled with Latent Semantic Analysis (LSA).
- Training the second classifier that uses NB-Logistic Regression model.
- Comparing the two classifiers by plotting the ROC and Precision-Recall curve.

Balancing the classes

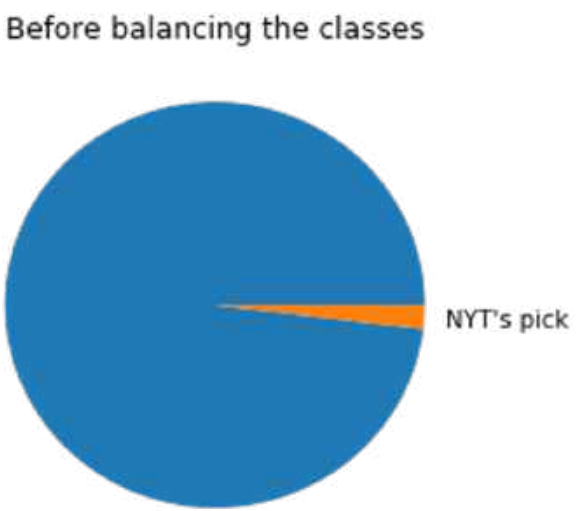
Balancing the classes by undersampling the majority class

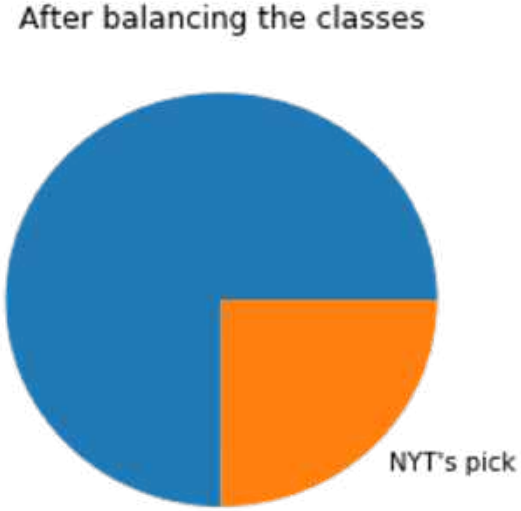
The two classes are highly imbalanced with an approxatimate ratio of 20:1. We will bring it down to less than 3:1 by undersampling the majority class

In [11]:

```
ratio = 3
def balance_classes(grp):
    picked = grp.loc[grp.editorsSelection == True]
    n = round(picked.shape[0]*ratio)
    if n:
        try:
            not_picked = grp.loc[grp.editorsSelection == False].sample(n)
        except: # In case, fewer than n comments with `editorsSelection == False`
            not_picked = grp.loc[grp.editorsSelection == False]
        balanced_grp = pd.concat([picked, not_picked])
        return balanced_grp
    else: # If no editor's pick for an article, dicard all comments from that article
        return None

comments = comments.groupby('articleID').apply(balance_classes).reset_index(drop=True)
```

Balancing the classes





Splitting the data using GroupKFold

We use GroupKFold to split the data into train and test sets such that the two sets have comments from disjoint set of articles. Here we used 5 splits for the GroupKFold

We split the data into train and test sets using GroupKFold

In [16]:
```
for train_index, test_index in GroupKFold(n_splits=5).split(commentBody, nytpicks, groups=articleID):
    train_text, test_text = commentBody[train_index], commentBody[test_index]
    train_target, test_target = nytpicks[train_index], nytpicks[test_index]
    train_groups, test_groups = articleID[train_index], articleID[test_index]

print("Number of comments for training:", train_text.shape[0])
print("Number of comments for testing:", test_text.shape[0])
```

```
Number of comments for training: 129107
Number of comments for testing: 32276
```

We get features using TFIDF for words and character n-grams and combine them using FeatureUnion

In [17]:
```
vectorizer = FeatureUnion([
    ('word_tfidf', TfidfVectorizer(
    analyzer='word',
    token_pattern=r'\w{1,}',
    ngram_range=(1, 2),
    max_features=600,
    )),

    ('char_tfidf', TfidfVectorizer(
    analyzer='char',
    ngram_range=(2, 4),
    max_features=600,
    ))
])
start_vect = time()
vectorizer.fit(commentBody)
train_text = vectorizer.transform(train_text)
test_text = vectorizer.transform(test_text)

print("Vectorization Runtime: %0.2f Minutes"%((time() - start_vect)/60))
```

```
Vectorization Runtime: 7.74 Minutes
```

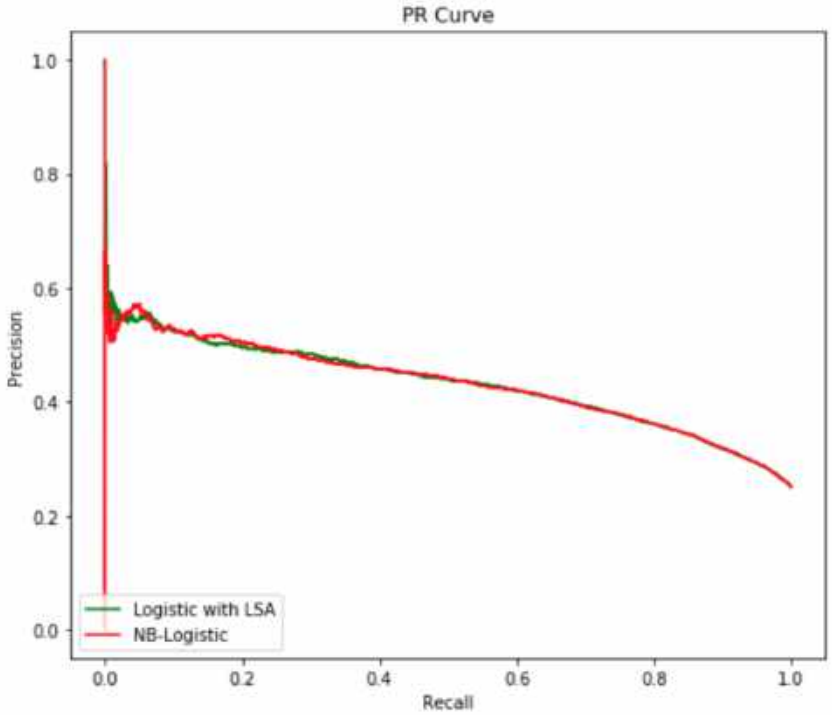
Logistic Regression coupled with Latent Semantic Analysis (LSA)

```
Classification report:
              precision    recall  f1-score   support

           0       0.77      0.95      0.85     24207
           1       0.51      0.14      0.22      8069

    accuracy                           0.75     32276
   macro avg       0.64      0.55      0.54     32276
weighted avg       0.70      0.75      0.69     32276

ROC AUC Score:  0.723
logloss:  0.5038

Runtime for training logistic regression model and predicting probabilities for the test set is 16.63 Minutes
```

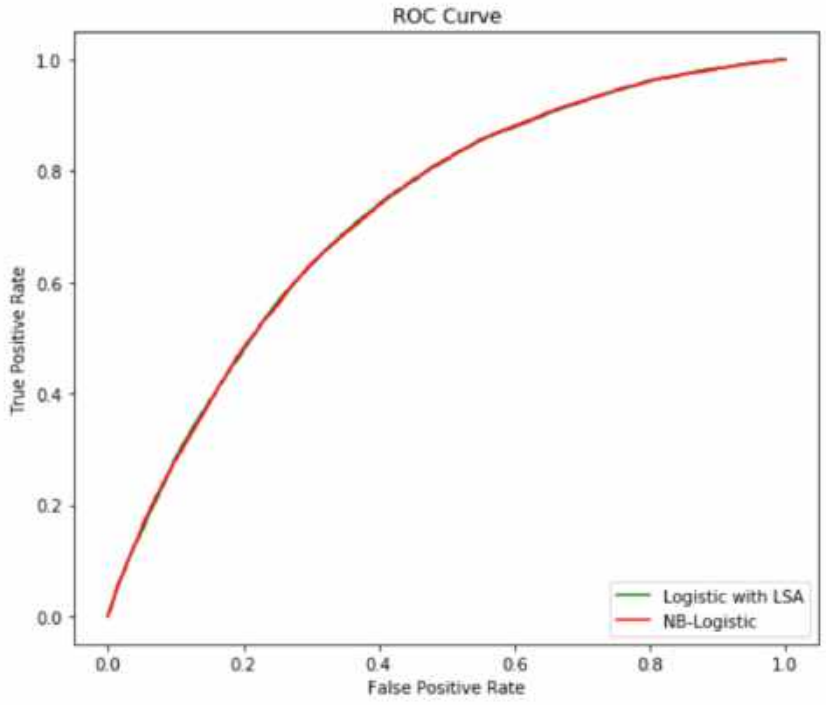
NB-Logistic Regression model

```
Classification report:
              precision    recall  f1-score   support

           0       0.76      0.97      0.86     24207
           1       0.53      0.09      0.16      8069

    accuracy                           0.75     32276
   macro avg       0.65      0.53      0.51     32276
weighted avg       0.71      0.75      0.68     32276

ROC AUC Score:  0.7231
logloss:  0.5037
Runtime for running GridSearchCV on NB-logistic regression model and predicting probabilities for the test set is 0.26 Minutes
```

Precision Recall Curve



ROC Curve



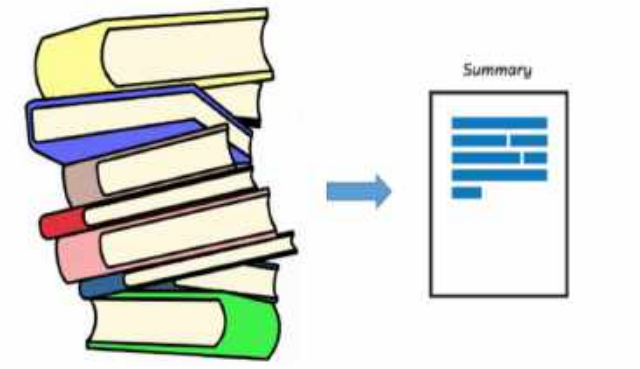
Conclusion: Both models seem to have performed equally well but the time taken by the NB Logistic model is very less compared to Logistic Regression coupled with LSA

Text Summarization

Understanding the Problem

New Articles can often be long and descriptive. Analyzing these articles manually, as you can imagine, is really time-consuming. This is where the brilliance of Natural Language Processing can be applied to generate a summary for long articles.

We will be working on the NY Times Articles. Our objective here is to generate a summary for the NY Times articles using the extraction-based approach.

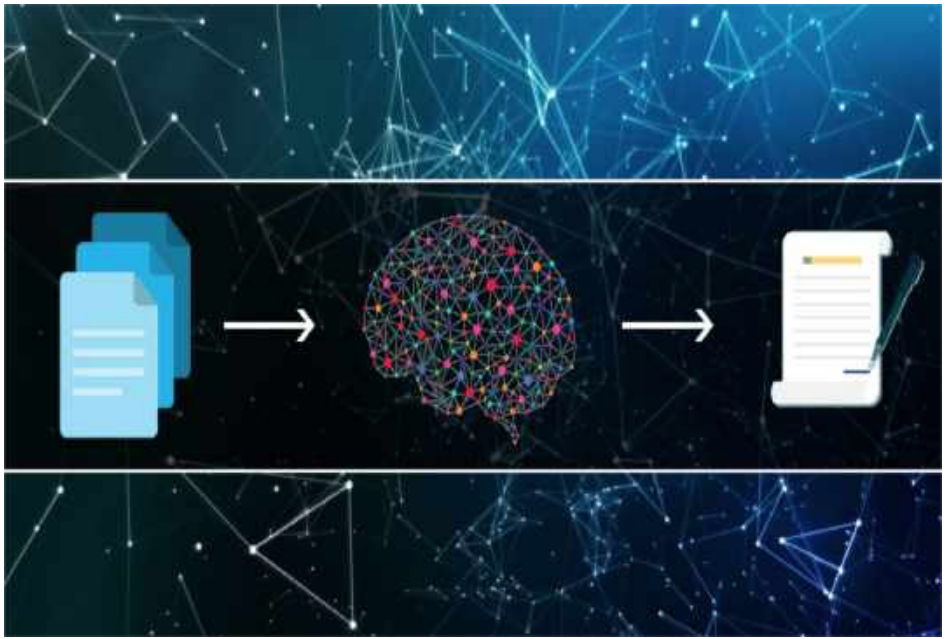


What is Text Summarization in NLP?

"Automatic text summarization is the task of producing a concise and fluent summary while preserving key information content and overall meaning"
- Text Summarization Techniques: A Brief Survey

There are broadly two different approaches that are used for text summarization:

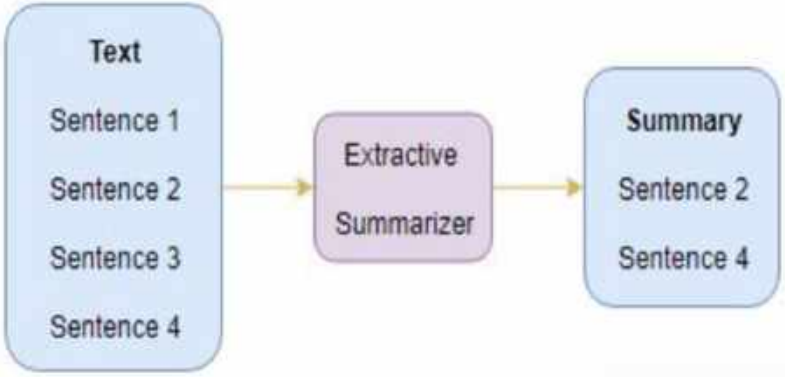
- Extractive Summarization
- Abstractive Summarization



Summarization Approaches

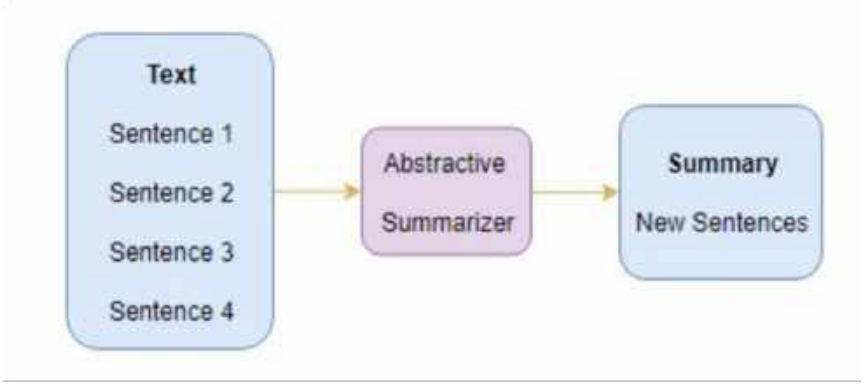
Extractive Summarization

The name gives away what this approach does. **We identify the important sentences or phrases from the original text and extract only those from the text.** Those extracted sentences would be our summary. The below diagram illustrates extractive summarization:



Abstractive Summarization

These methods uses advanced NLP techniques to generate new sentences from the original text. This is in contrast to the extractive approach, where we used only the sentences that were present. The sentences generated through abstractive summarization might not be present in the original text.



TextRank Algorithm

Graph-based ranking algorithms are essentially a way of deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph. The basic idea implemented by a graph-based ranking model is that of "voting" or "recommendation"

Example:

webpage	links
w1	[w4, w2]
w2	[w3, w1]
w3	[ ]
w4	[w1]

M =

	w1	w2	w3	w4
w1	0	0.5	0	0.5
w2	0.5	0	0.5	0
w3	0.25	0.25	0.25	0.25
w4	1	0	0	0

The initialization of the probabilities is explained in the steps below:

1. Probability of going from page i to j, i.e., M[i][j], is initialized with **1/(number of unique links in web page wi)**
2. If there is no link between the page i and j, then the probability will be initialized with **0**
3. If a user has landed on a dangling page, then it is assumed that he is equally likely to transition to any page. Hence, M[i][j] will be initialized with **1/(number of web pages)**

Similarity Matrix

A widely used measure in Natural Language Processing is the Cosine Similarity. The Cosine Similarity computes the cosine of the angle between 2 vectors. If the vectors are identical, the cosine is 1.0. If the vectors are orthogonal, the cosine is 0.0.

Example:

Doc Trump (A) : Mr. Trump became president after winning the political election. Though he lost the support of some republican friends, Trump is friends with President Putin.

Doc Trump Election (B) : President Trump says Putin had no political interference is the election outcome. He says it was a witch hunt by political parties. He claimed President Putin is a friend who had nothing to do with the election.

Doc Putin (C) : Post elections, Vladimir Putin became President of Russia. President Putin had served as the Prime Minister earlier in his political career.

Since, Doc B has more in common with Doc A than with Doc C, we would expect the Cosine between A and B to be larger than (C and B).

```
print(cosine_similarity(df, df))
#> [[ 1.         0.48927489 0.37139068]
#>  [ 0.48927489 1.         0.38829014]
#>  [ 0.37139068 0.38829014 1.        ]]
```

GloVe Vectors

GloVe is an unsupervised learning algorithm for obtaining vector representations for words.

Model Overview

GloVe is essentially log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding.

Example:

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

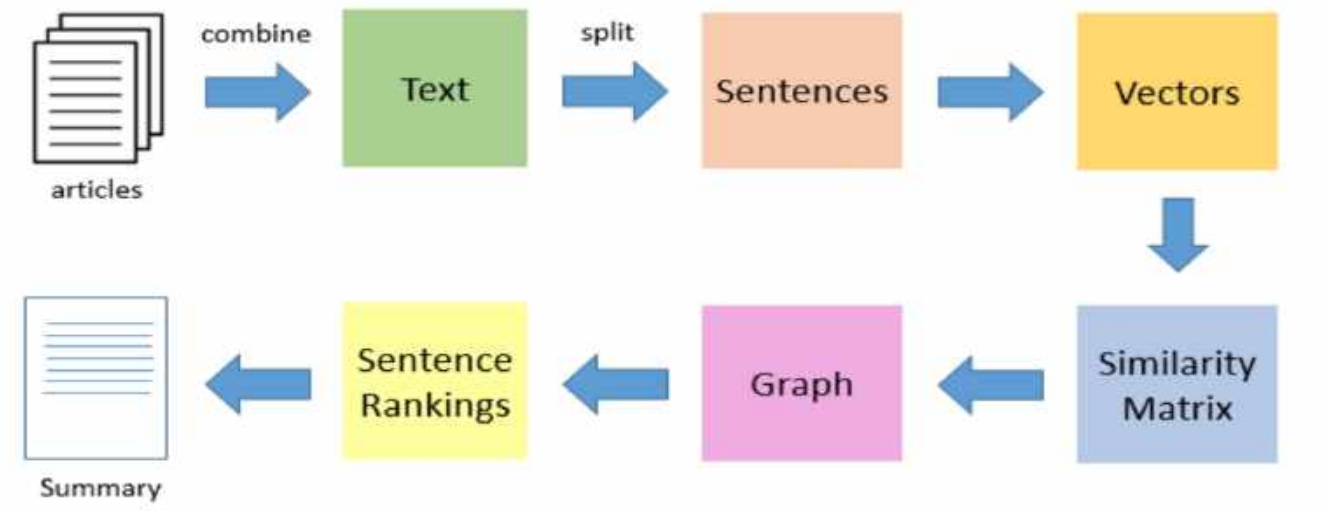
GloVe text file:

Each word will be associated to 100 co-relational values

steam -0.53452 0.81197 -0.24997 -0.50114 0.33931 -0.84524 0.92475 0.047031 -0.19397 -0.40454 0.27995 0.29819 -0.74724 0.65648 -0.27242 -1.3483 -0.3241 0.25163 0.34261 -0.92609 0.72125 -0.32556 -0.60705 -0.92492 0.19722 -0.43509 -0.99115 0.46721 -0.2004 0.33468 -0.52064 0.18069 -0.94552 -0.39733 -0.87828 0.34831 0.060954 -0.69427 0.71339 -0.10077 -0.67026 -1.0672 -0.37575 -0.49421 0.87882 0.019126 0.63018 -0.45878 1.3492 -0.50639 -0.60674 0.44845 -0.037007 0.59822 0.16497 -1.5688 -0.48435 0.41972 1.0518 -0.14557 0.0060301 -0.0043333

Steps

1. Data collection from web through web-scraping
2. Data Cleanup (remove special characters, numeric values, stopwords, punctuations)
3. Tokenization - Creation of tokens (Sentences tokens)
4. Find vector representation (word embeddings) for each and every sentence
5. Build Similarity matrix with calculated sentence vectors
6. Similarity matrix is converted to graph using Pagerank, where sentences will be vertices and similarity score as edges.
7. Calculate sentence ranks
8. Creation of summary with top N highest scored sentences



Auto-Text Generation

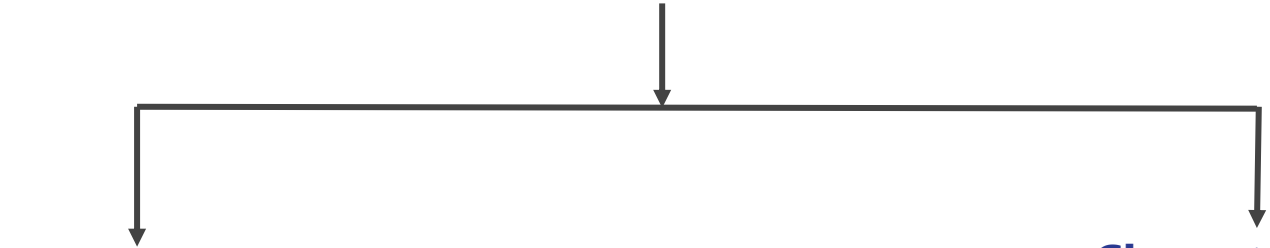
Neural Network

Neural Networks are set of algorithms which closely resemble the human brain and are designed to recognize patterns.

Natural Language Processing (NLP)

Natural language processing is necessary for tasks like the classification of word documents or the creation of a chatbot.

Two ways to tackle NLP task



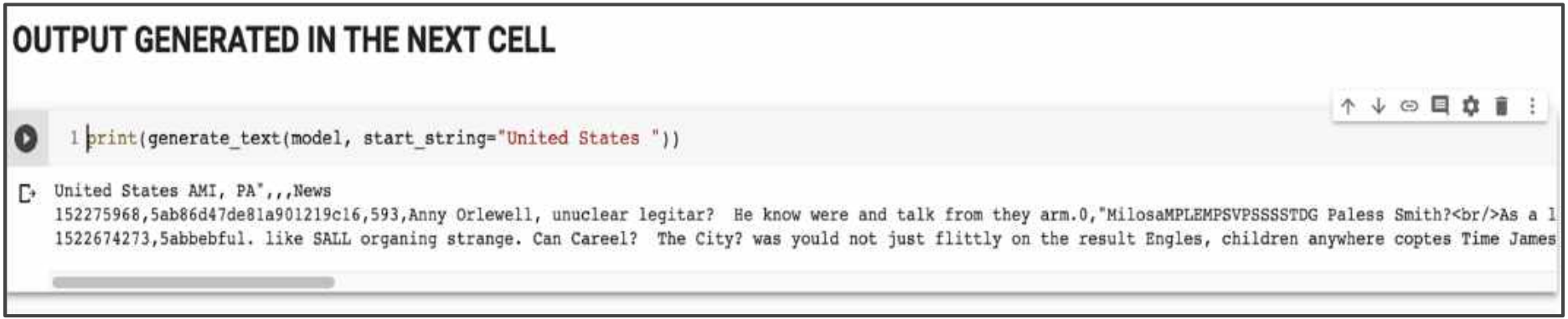
Word-level Text Generation

- Display higher accuracy
- Form shorter representations of sentences

Character-level Text Generation

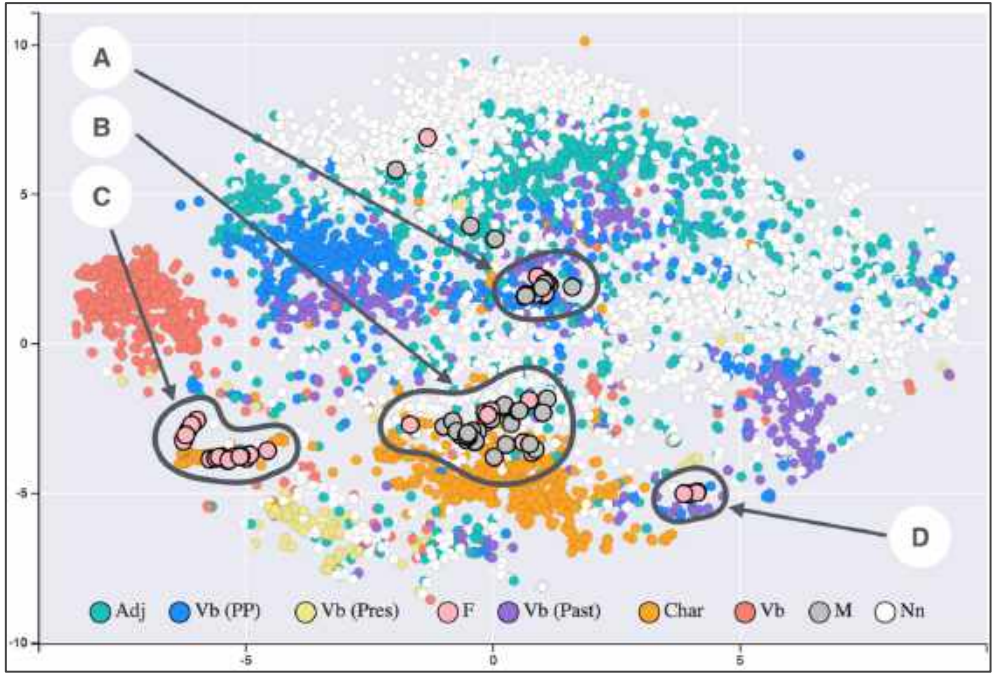
- Quicker to train the model
- Less memory required

Output for Character-level Text Generation



Word Embeddings

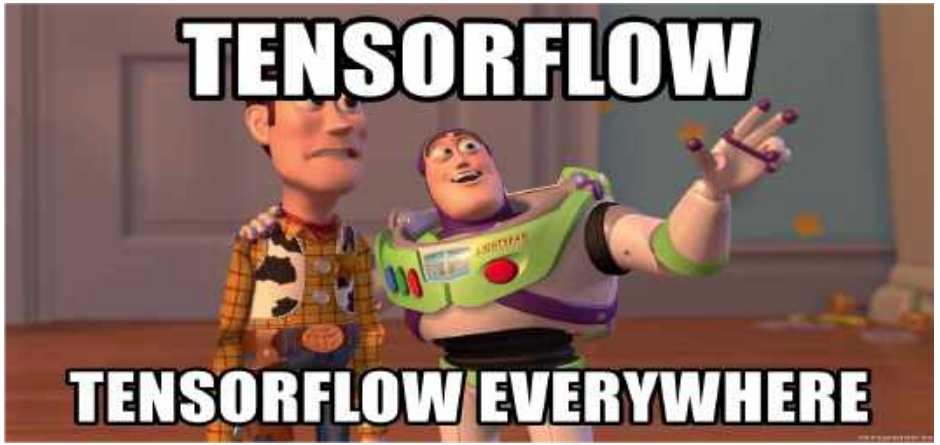
- Represents words or phrases as a vector of real numbers
- Similar to one-hot encoding
- But uses more numbers than simply ones and zeros
- Therefore it can form more complex representations

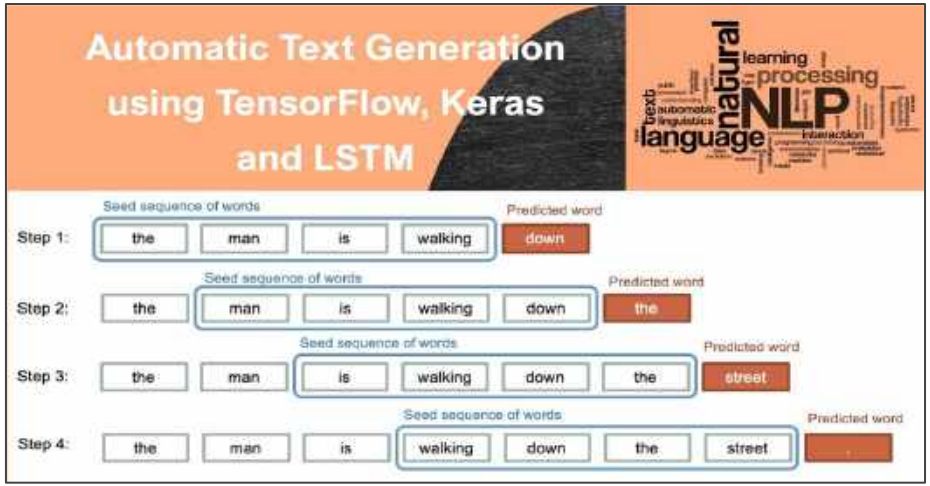


TensorFlow and Keras

- **TensorFlow**
  One of the most commonly used machine learning libraries specializing in the creation of deep neural networks

- **Keras**
  Uses TensorFlow's functions and abilities
  Makes building a neural network much simpler and easier





Recurrent Neural Network (RNN)

- A generalization of feedforward neural network with internal memory.
- Recurrent in nature

Advantages

Model size not increasing with size of input

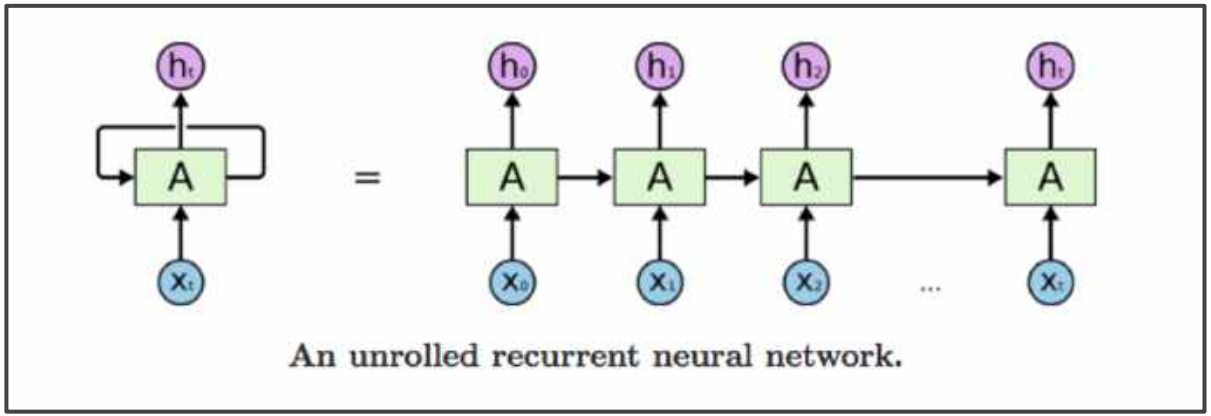
Computation takes into account historical information

Disadvantages

The longer the input series is, the more the network "forgets"

Computation is slow

How RNN works?



An unrolled recurrent neural network.

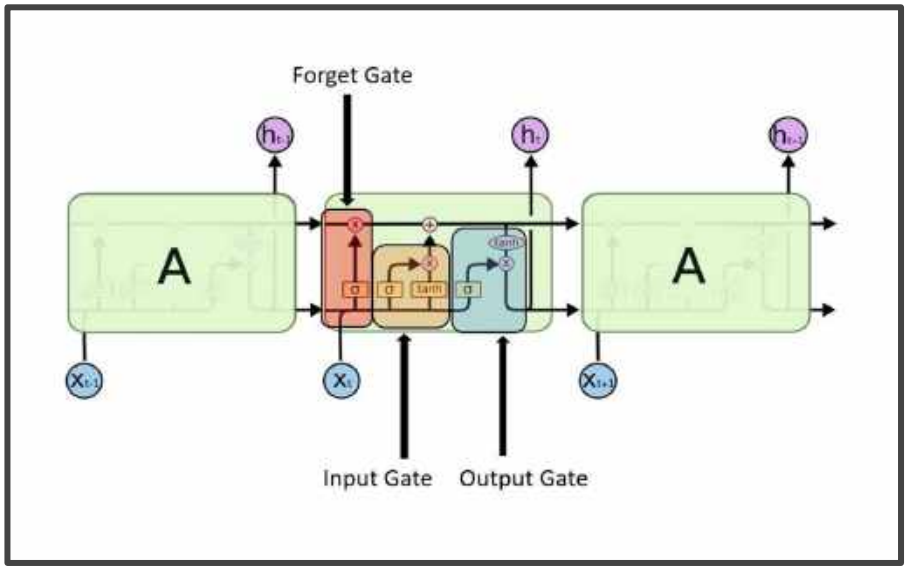
First step, Input → X(0) Output → h(0)
Second step, Input → h(0), X(1) Output → h(1)
Third step, Input → h(1), X(2) Output → h(2)

And so on.

This way, it keeps remembering the context while training.

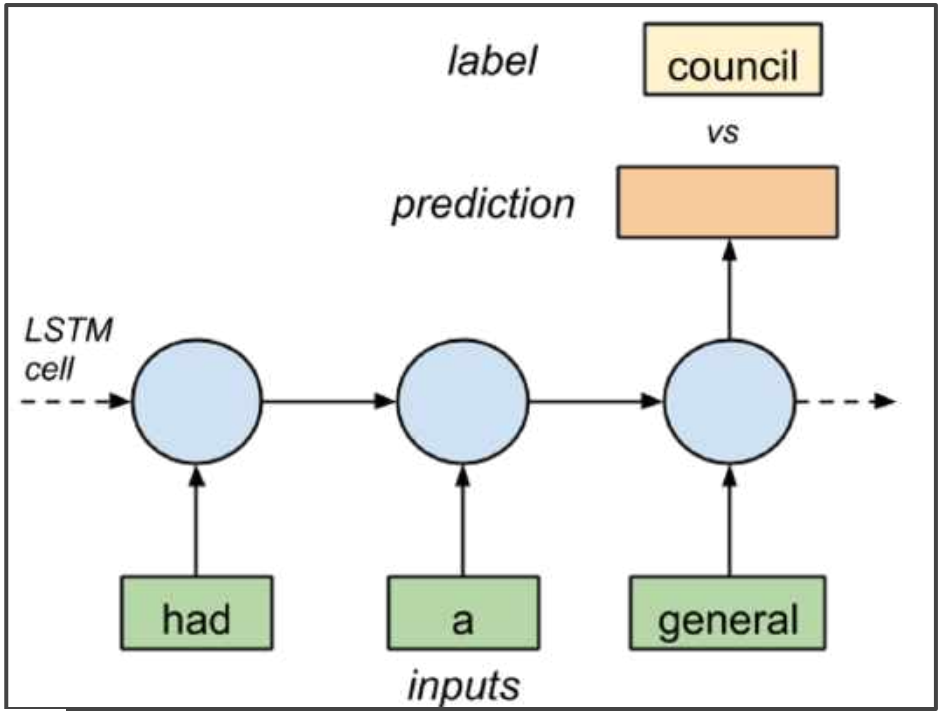
LSTM

1. A modified version of recurrent neural networks
2. Makes it easier to remember past data in memory



1. **Input Gate**
   Discover which value from input should be used to modify the memory
1. **Forget Gate**
   Discover what details to be discarded from the block
1. **Output Gate**
   The input and the memory of the block is used to decide the output

If we feed a LSTM with correct sequences from the text of 3 symbols as inputs and 1 labeled symbol, eventually the neural network will learn to predict the next symbol correctly



Technically, LSTM inputs can only understand real numbers. A way to convert symbol to number is to assign a unique integer to each symbol based on frequency of occurrence

```python
def create_model(max_length, total_words):
    length = max_length - 1
    model = Sequential()

    # Add Input Embedding Layer
    model.add(Embedding(total_words, 10, input_length=length))

    # Add Hidden Layer 1 - LSTM Layer
    model.add(LSTM(100))
    model.add(Dropout(0.1))

    # Add Output Layer
    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model

model = create_model(max_length, total_words)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 23, 10)            112650
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               44400
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 11265)             1137765
=================================================================
Total params: 1,294,815
Trainable params: 1,294,815
Non-trainable params: 0
_________________________________________________________________
```

Explanation of the algorithm

- The idea is to train the RNN with many sequences of words and the target next_word.

Example:

- If each sentence is a list of five words, then the target is a list of only one element, indicating which is the following word in the original text

Thank you!