# Text Mining: Extracting and Analyzing all my Blogs on Machine Learning

Niranjan Kumar Follow
Mar 16 · 13 min read


Photo by Thought Catalog on Unsplash

Recently I have started working on Natural Language Processing at work and at home. So I decided to apply some of the techniques I had come across, to create a word cloud of the words extracted from all of my blogs.

I have published my first article on Feb 15, 2019. 2 months into writing, I was selected as the top writer on Medium in Artificial Intelligence. The recognition motivated me to continue sharing my knowledge with the data science community. Since then I have written more than 25+ articles on the topics related to machine learning, deep learning on Medium and Marktechpost.

In this article, we will scrap the contents of all the articles and process the data using the NLTK Package for further analysis. To scrap the data, we will be using `Selenium` and `BeautifulSoup`.

> *Before you start with scrapping the data from any website, please check the Terms and Conditions of that website.*

## Setting Up Environment

Download and Install the Anaconda package manager. Anaconda distribution contains a collection of many open source packages. Apart from the packages present in the Anaconda, we will be using new packages to extract the content from the webpages.

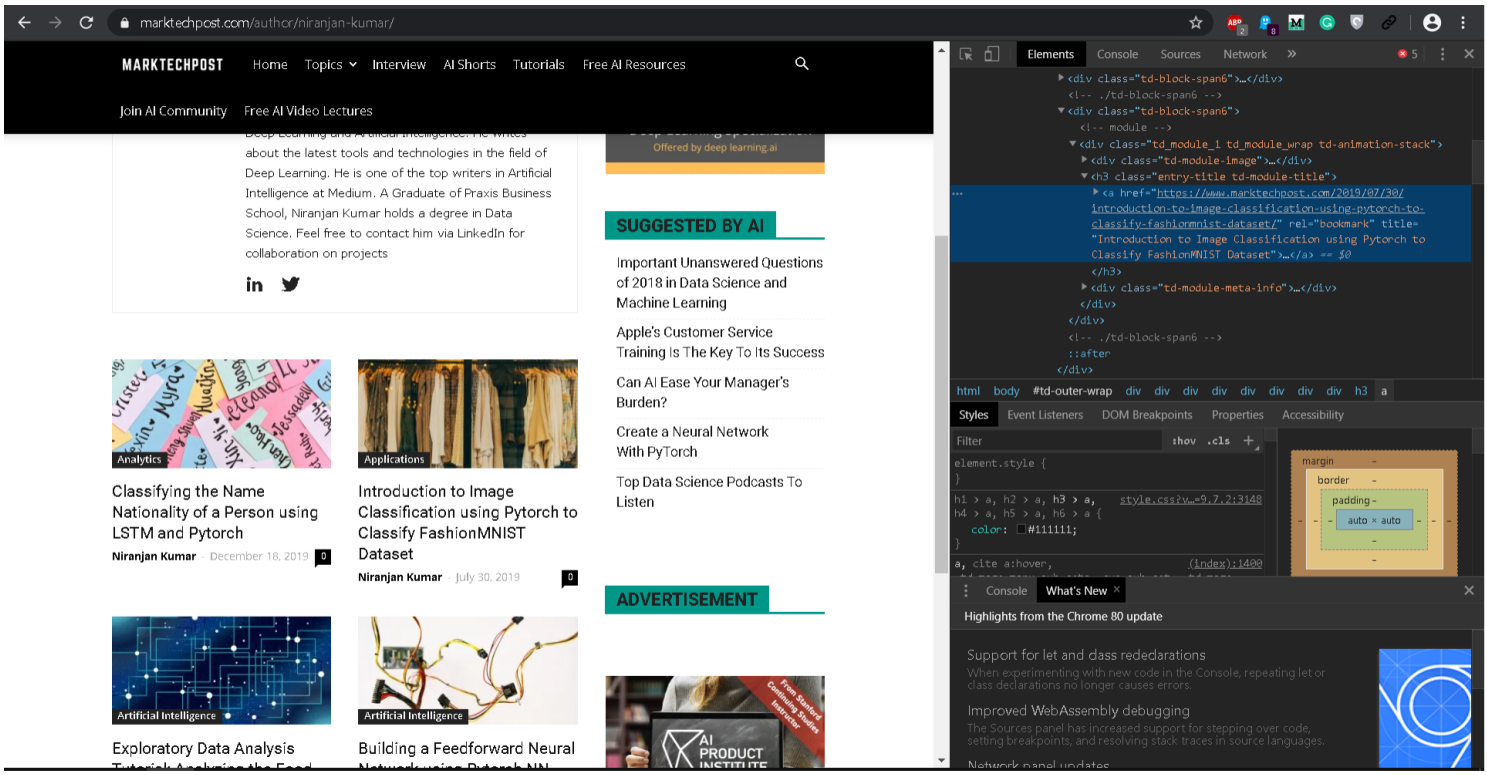
To install the new packages, you can use Anaconda's package manager, conda or pip to install those packages.

```
pip install selenium
pip install beautifulsoup4
```

## Inspect the Webpage

To find out which elements of the webpage that we need to extract using python, we need to first inspect the webpage. To inspect any webpage you need to right-click on the webpage and select inspect or Fn + F12. This brings up the HTML code inspect tool where we can see the element that each field is contained within.

For example, if I inspect the marktechpost author page you can see that the article links and article title are present inside the `<a>` tag. The `<a>` tag has `href` attribute which specifies the URL of the page the link goes to and it also has `title` attribute which defines the title of the document.




On inspecting further all the articles that are under the author page follow the same pattern of `href` and `title`. The `<a>` tag for all the articles has `rel` attribute, which is used to specify the relationship between the current and the linked document. Using these attributes, we can easily scrape the relevant information from the webpage HTML code using BeautifulSoup Package. (I will discuss how to extract using BeautifulSoup down below)

Similarly, we can inspect the medium author page and find out under which tag the relevant information like article links and article title is present.

All the code discussed in the article is present on my GitHub

**Niranjankumar-c/DataScienceBlogAnalysis_NLP**

This repository contains the code to scrap the contents all of my articles and process the data using the NLTK Package...

github.com

## Scrapping Medium WebPage using Selenium

First, we will scrape the links of all the published articles under my account from Medium. Since Medium website uses javascript to load the page we can't use BeautifulSoup directly to scrape the data. Instead of BeautifulSoup, we will use the Selenium web driver to open the webpage, search for the links of the articles and return the results.

```
from selenium import webdriver
import time

medium_profile_link = "https://medium.com/@niranjankumarc"
```

In this tutorial, I will be using Chrome as my browser, so we need to download the Chrome WebDriver. You can download the WebDriver from here. If you are using any other browser then you need to download the specific WebDriver. Once you downloaded the WebDriver, run the Chrome WebDriver by pointing the executable path as the downloaded file path.

```
#create the driver for chrome browser with executable.

driver = webdriver.Chrome(executable_path =
'C:\\Users\\NiranjanKumar\\Downloads\\chromedriver.exe')
```

Now, we can connect to the webpage and search for the links of the articles. When we load the webpage in a browser often it will take a while to load the entire page and also may not even load until we scroll down towards the end of the page. To handle this problem, we will execute a javascript code that helps to load the entire webpage.

```
#load the webpage and scroll till the bottom of the page
driver.get(medium_profile_link)

# Get scroll height
last_height = driver.execute_script("return
document.body.scrollHeight")
```

```
while True:
    # Scroll down to bottom
    driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")

# Wait to load page
    time.sleep(30)

# Calculate new scroll height and compare with last scroll height
    new_height = driver.execute_script("return
document.body.scrollHeight")
    if new_height == last_height:
        break
    last_height = new_height
```

Once you execute the above script, Chrome will open the URL specified and scroll down to the bottom of the page. Next, we need to get the content of the webpage. To find all the elements of interest, we will create a BeautifulSoup Object and extract all the relevant `div` tags.

Use inspect webpage option to find out where the relevant links of the articles and article titles are present. Using the `findAll` command in the BeautifulSoup object, we will extract the relevant tags.

```
#get the html content of the page
html = driver.page_source

#create a soup object
medium_soup = BeautifulSoup(html)

#finding the divs with class = 'r s y'
soup_divs = medium_soup.findAll("div", {'class' : 'r s y'})
```

Once we got the `div` tags, we iterate through each tag and get the article link and article title. The article link is present inside `a` tag and the article link is inside `h1` the tag.

```
title_lst = []
links_lst = []

for each_div in soup_divs:
```

```
    article_link = each_div.find("a").get("href").split("?")[0]
    article_title = each_div.find("a").find("h1").text.strip()

    if article_link.startswith("https") == False:
        article_link = "https://medium.com" + article_link #appending
the address for links (eg. hackernoon: moved out of medium)

    #append the values to list
    title_lst.append(article_title)
    links_lst.append(article_link)

driver.quit() #stop the driver
```

Now that we have all the links and titles of the articles, we will execute the `driver.quit()` to ensure the browser closes.

## Scrapping Marktechpost WebPage using BeautifulSoup

To scrape the data of the marktechpost author page we can use selenium as shown in the previous step. In this step, we will use `requests` and `BeautifulSoup` libraries to scrape the data.

Once we have the URL of the webpage that we want to scrape the data. We then make the connection to the webpage using `requests` and parse the webpage using `BeautifulSoup`, storing the object in the variable 'mark_soup'.

```
import requests

#creating a variable
mark_url =
requests.get('https://www.marktechpost.com/author/niranjan-kumar/')

#create a beautifulsoup object
mark_soup = BeautifulSoup(mark_url.content)

#print soup
print(mark_soup.prettify())
```

As discussed in the previous section — inspect the webpage, we need to search the soup object to find the relevant tags. Once we got the `div` tags, we iterate through each tag and get the article link and article title.

```
#iterate the articles and get the links and titles
for eachitem in mark_soup.findAll("a", {'rel': 'bookmark'}):
    title_lst.append(eachitem.get("title"))
    links_lst.append(eachitem.get("href"))
```

Add the results of the search operation to the lists of article links and article titles. we can make use of that by saving the data we have scraped to a data frame. We can print the data frame to view the content.

```
#create a dataframe
titles_df = pd.DataFrame(list(zip(title_lst, links_lst)), columns =
["Title", "URL"])

titles_df.head()
```

## Extracting Content

Now that we have links to all the articles published, let's get the content of these articles. Since we are extracting the whole content from the webpage including the HTML content, we will be using `html2text` package to convert a page of HTML into clean, easy-to-read plain ASCII text.

To install `html2text` package

```
!pip install html2text
```

`html2text` gives multiple options like ignoring links, ignoring images, ignoring tables in the extracted data to get a clean and easily readable text.

```
import html2text

h = html2text.HTML2Text()

#ignoring all the links, tables and images in the blog
h.ignore_links = True
```

```
h.ignore_images = True
h.ignore_tables = True
```

Iterate over each link and extract the HTML content append it to the new column in the `titles_df` data frame.

```
content_lst = []

#iterating through all the links
for ind in titles_df.index:

    #content request object
    request_content = requests.get(titles_df["URL"][ind])

    main_content = h.handle(str(request_content.content)) #get the
text from the html content

    content_lst.append(str(main_content))
```

The resultant data frame should look like this,

```
#add the new column

titles_df["content"] = content_lst
titles_df.head()
```

| | Title | URL | content |
|---|---|---|---|
| 0 | Visualizing Convolution Neural Networks using ... | https://towardsdatascience.com/visualizing-con... | b'\n\nSign in\n\n * Data Science\n * Machine... |
| 1 | Introduction to Encoder-Decoder Models — ELI5 Way | https://towardsdatascience.com/introduction-to... | b'\n\nSign in\n\n * Data Science\n * Machine... |
| 2 | Long Short Term Memory and Gated Recurrent Uni... | https://towardsdatascience.com/long-short-term... | b'\n\nSign in\n\n * Data Science\n * Machine... |
| 3 | My Guest Blogs on Data Science Published in 2019 | https://medium.com/@niranjankumarc/my-guest-bl... | b'\n\nBecome a member\n\nSign in\n\n# My Guest... |
| 4 | Recurrent Neural Networks (RNN) Explained — th... | https://towardsdatascience.com/recurrent-neura... | b'\n\nSign in\n\n * Data Science\n * Machine... |

Final DataFrame

. . .

## Text PreProcessing— Cleaning Data

In this section, we will discuss how to pre-process (clean) the content of the webpage, so that we can extract useful information from the cleaned data.

## Import Packages

Before we start mining the data, first we need to import the required libraries. We will `nltk` tokenize the text and also remove the stopwords from the corpus and `wordcloud` package to generate the word cloud.

```
#import required packages
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns
from nltk.corpus import stopwords
import re
import string
```

import_blogmining.py hosted with ♡ by GitHub view raw

Let's clean the data.

Photo by Sophie Elvis on Unsplash

## 1. Removing Links

Even though we have set the option to ignore the links using `html2text` package, it is possible some of the links might be present in the content of the webpage. The first preprocessing step is to remove the links present in the content. We will do this by using regex.

```
def remove_links(text):
    return re.sub(r'https?://[^\s]+', '', text)
```

The `remove_links` function takes text as an input and replaces the URL with empty value in the text. We will apply this function across the data frame for all the contents using `lambda` function.

```
# 1. Remove all the links if present in the content

titles_df["content"] = titles_df.content.apply(lambda content:
remove_links(content))
titles_df.head()
```

## 2. Removing Extra Spaces and Tab

Before removing extra spaces, we will join all the contents in the data frame using "\n\n" and make it a single string. Using regex we will replace the extra spaces and tab with a single space.

```
# 2. Join all the contents in the dataframe using "\n\n" and make it
a single string

main_content_string =  "\n\n".join(titles_df.content)

# Remove all whitespaces (\n and \t) with space

main_content_string = re.sub('\s+',' ',main_content_string).strip()
main_content_string[:300]
```

## 3. Expanding Contractions

A **contraction** is a word or phrase that has been shortened by dropping one or more letters. Examples of contractions are words like “ain’t”, “aren’t”, don’t. We leverage an existing set of functions written by Rahul Agarwal to expand contractions. Check out the original post here.

```
# 3. expand contractions

def _get_contractions(contraction_dict):
    contraction_re = re.compile('(%s)' %
'|'.join(contraction_dict.keys()))
    return contraction_dict, contraction_re

contractions, contractions_re = _get_contractions(contraction_dict)

def replace_contractions(text):
    def replace(match):
        return contractions[match.group(0)]
    return contractions_re.sub(replace, text)

# Usage
main_content_string = replace_contractions(main_content_string)
```

## 4. Removing Punctuation and Special Characters

Punctuation and special characters are non-alphanumeric characters which don’t give any information while mining data. Therefore removing these characters will give us the cleaned data for analysis.

To remove punctuation we will make use of `string` package which we imported earlier.

```
# 4. Remove punctuation

main_content_string = main_content_string.translate(str.maketrans('', '', string.punctuation))
main_content_string = main_content_string.replace("‘", '').replace("’", '').replace("'", '')
main_content_string[:300]
```

punct_blogmining.py hosted with ♡ by GitHub view raw

## 5. Lowercase

We need to transform all our tweets to lowercase to eliminate the duplicate occurrence of words like '**USA**', '**usa**' and '**Usa**'.

```
# 5. Lowercase

main_content_string = main_content_string.lower().strip("b").strip()
```

## 6. Tokenization

Tokenization is a process of dividing the text into words or sentences. Since we have only one big sentence, we will split this tokenize the sentence into words using `word_tokenize` from NLTK Package.

```
# 6. Tokenization

word_tokens = word_tokenize(main_content_string)
word_tokens[:10]
```

## 7. Removing StopWords

Stopwords are the words that occur frequently in the corpus and have little or no significance to the features extracted from the text. Typically, these can be articles, conjunctions, prepositions and so on. Some examples of stopwords are *a*, *an*, *the*, *and*.

We will make use of the stopwords list present in the NLTK library to remove those words from our tokens.

```
# 7. remove stop words
stop_words = stopwords.words('english')

#remove stopwords
main_content_tokens = [w for w in word_tokens if not w in stop_words]
```

## 8. Removing AlphaNumeric Words

The words which contain both alphabets and numbers don't give any extra information regarding the text. So we will remove all the words that contain numbers.

```
# 8. Remove all words containing numbers

main_content_tokens = [word for word in main_content_tokens if
word.isalpha()]
```

## 9. Removing Frequently Occurring Words in the Corpus

First, we will check the most frequently occurring words in the corpus. I felt like these words will not affect the analysis, we will be doing on the data. So I have removed these words from the corpus.

```
# 9. Remove frequently occuring words in the corpus

freq = pd.Series(main_content_tokens).value_counts()[:10]
freq

#output
data         659
network      483
function     460
learning     447
nnnn         441
xexx         394
neural       355
model        309
input        306
using        290

#removing words
main_content_tokens = [word for word in main_content_tokens if word
not in list(freq.index)]
```

## 10. Removing Rare Words Removal

We will remove the 1000 rare words from the corpus, these rare words will not make any impact on the analysis. These words are so rare that they act as the noise for the analysis.

```
# 10. removing the least frequent words in the corpus

freq = pd.Series(main_content_tokens).value_counts()[-1000:]
freq
```

```
#removing words - less frequent words
main_content_tokens = [word for word in main_content_tokens if word
not in list(freq.index)]
```

### 11. Removing Miscellaneous Words

On further inspecting the cleaned tokens that we got so far, I found out that there are miscellaneous words which make no sense to keep to them in our corpus. For example '**rnrn**', '**rntt**' and '**xcxb**'. So I have removed all the tokens that contain these words.

```
#11. Removing miscellaneous words

main_content_tokens = [word for word in main_content_tokens if not
any([phrase in word for phrase in ["rnrn", "rntt", "xcxb", "xx"]])]

#miscellaneous words
main_content_tokens = [word for word in main_content_tokens if
len(word) >= 4]
```

### Cleaned Corpus

Up to this point, we have done all our text cleaning or pre-processing steps, now we will merge our cleaned tokens into a single sentence so that we can the cleaned corpus to identify the top topics using WordCloud.

```
#merge all the tokens
cleaned_content = " ".join(main_content_tokens)
```

. . .

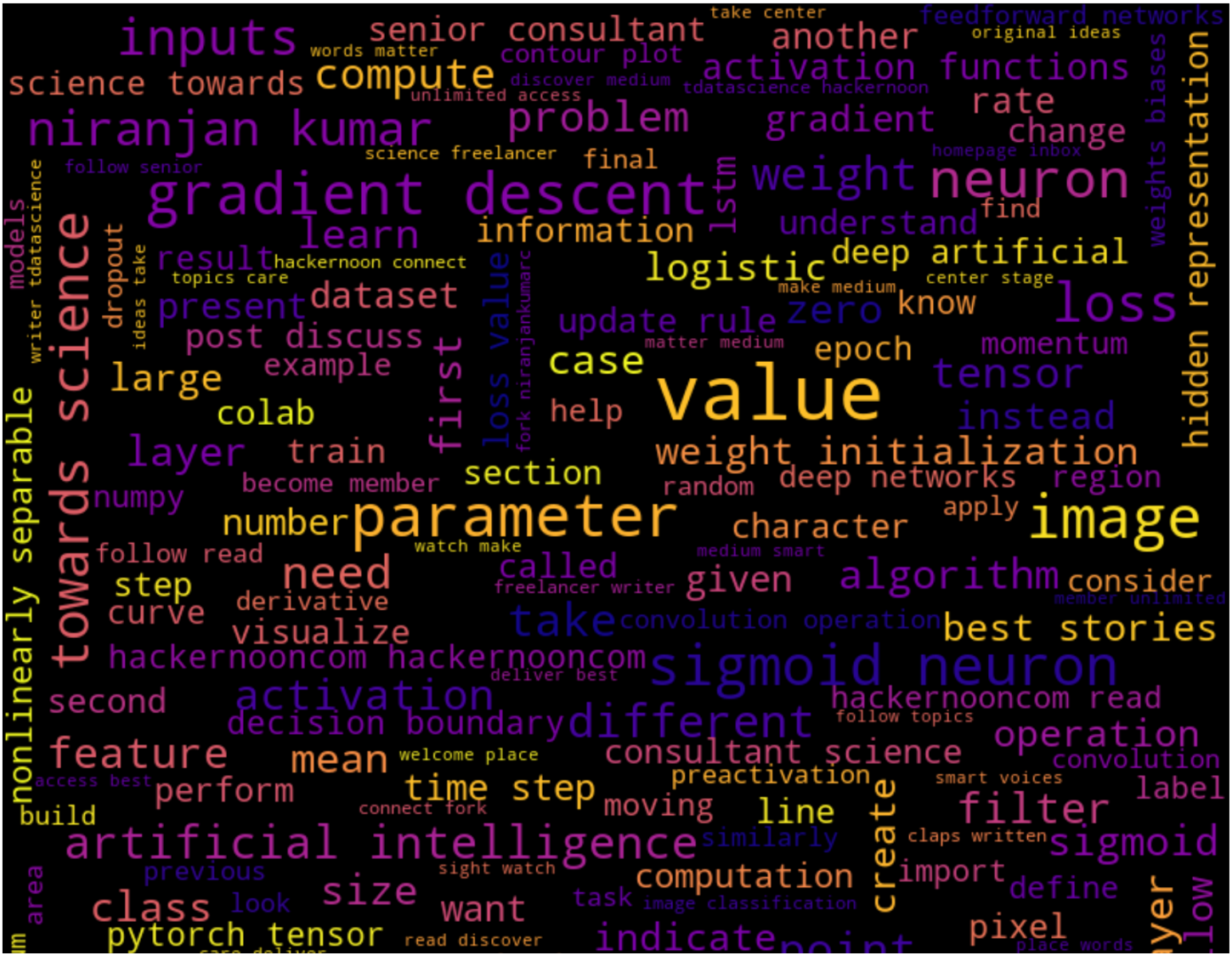
## WordCloud

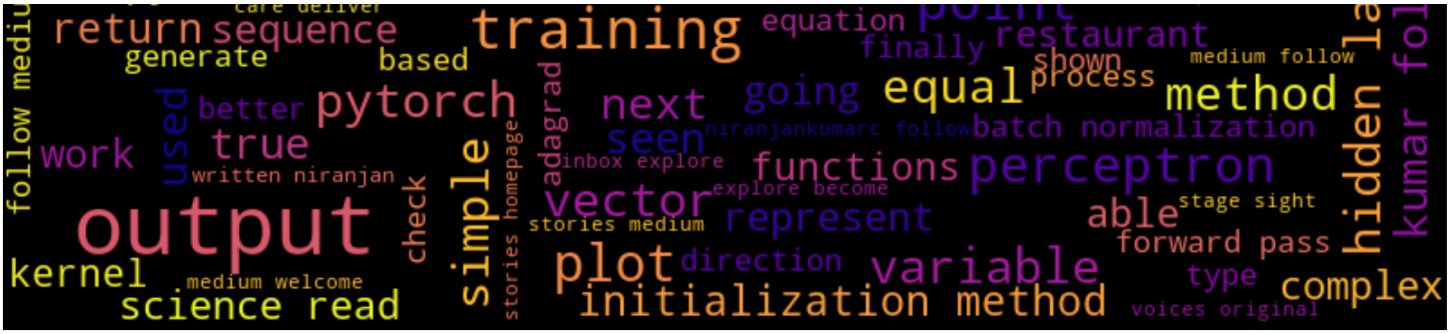
Now we are ready to create a WordCloud to identify the top topics in the corpus. To create a WordCloud we will make use of `wordcloud` library.

```
# Create stopword list:
stopwords = set(STOPWORDS)
```

```
wordcloud = WordCloud(width = 800, height = 800,
max_font_size=50,min_font_size = 10,
                stopwords = stopwords, background_color = "black",
colormap="plasma").generate(cleaned_content)

# plot the WordCloud image
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.tight_layout()

# store to file
plt.savefig("av_wordcloud.png", dpi=150)

plt.show()
```

Once we execute the above script, WordCloud will be generated and saved into the local directory as `av_wordcloud.png` .





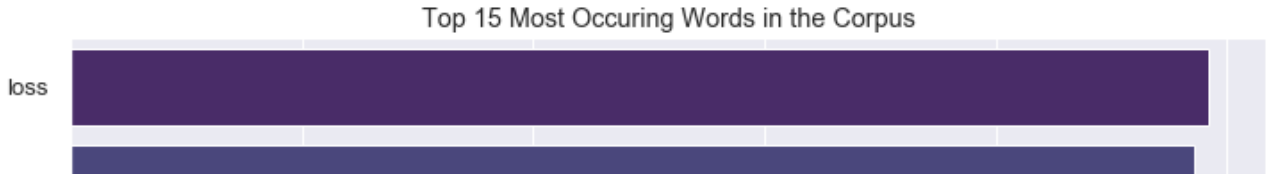

From the WordCloud, we can clearly see some of the top words such as “Gradient Descent”, “Output”, “Loss”, “Decision Boundary”, “Hidden Representation” etc.. indicates that the corpus mainly talks about Neural Networks and Deep Learning Techniques. This is expected because most of my articles are structured around Deep Learning starting from basics like Perceptron to Encoder-Decoder Models.

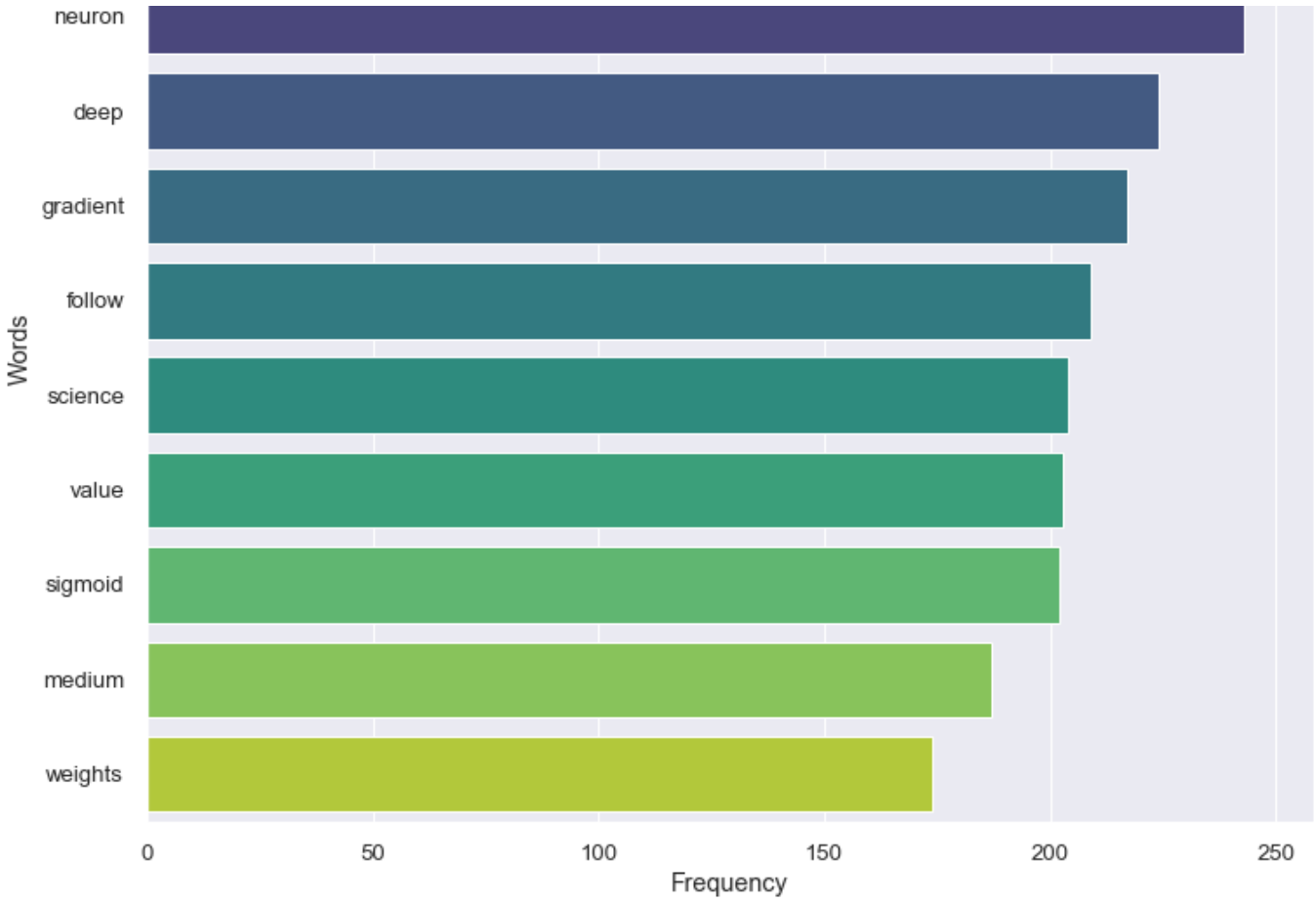
## Most Frequent Words

Another way to visualize the top words is by using bar plots. We will use `Counter` from `collections` package to count the frequency of words and plot the top 10 words and their frequency.

```
#frequent words

counted_words = Counter(main_content_tokens)
most_common_df = pd.DataFrame(counted_words.most_common(10), columns=
["words", "count"])

#plot the most common words
sns.barplot(y = "words", x = "count", data = most_common_df,
palette="viridis")
plt.xlabel("Frequency")
plt.ylabel("Words")
plt.title("Top 15 Most Occuring Words in the Corpus")
plt.show()
```

The barplot generated would look like this,






## StyleCloud

Instead of creating simple and boring WordCloud, we will use `stylecloud` package to generate stylistic WordClouds, including gradients and icon shapes!.

`stylecloud` is a Python package that leverages the popular word_cloud package, adding useful features to create truly unique word clouds!

To install the package

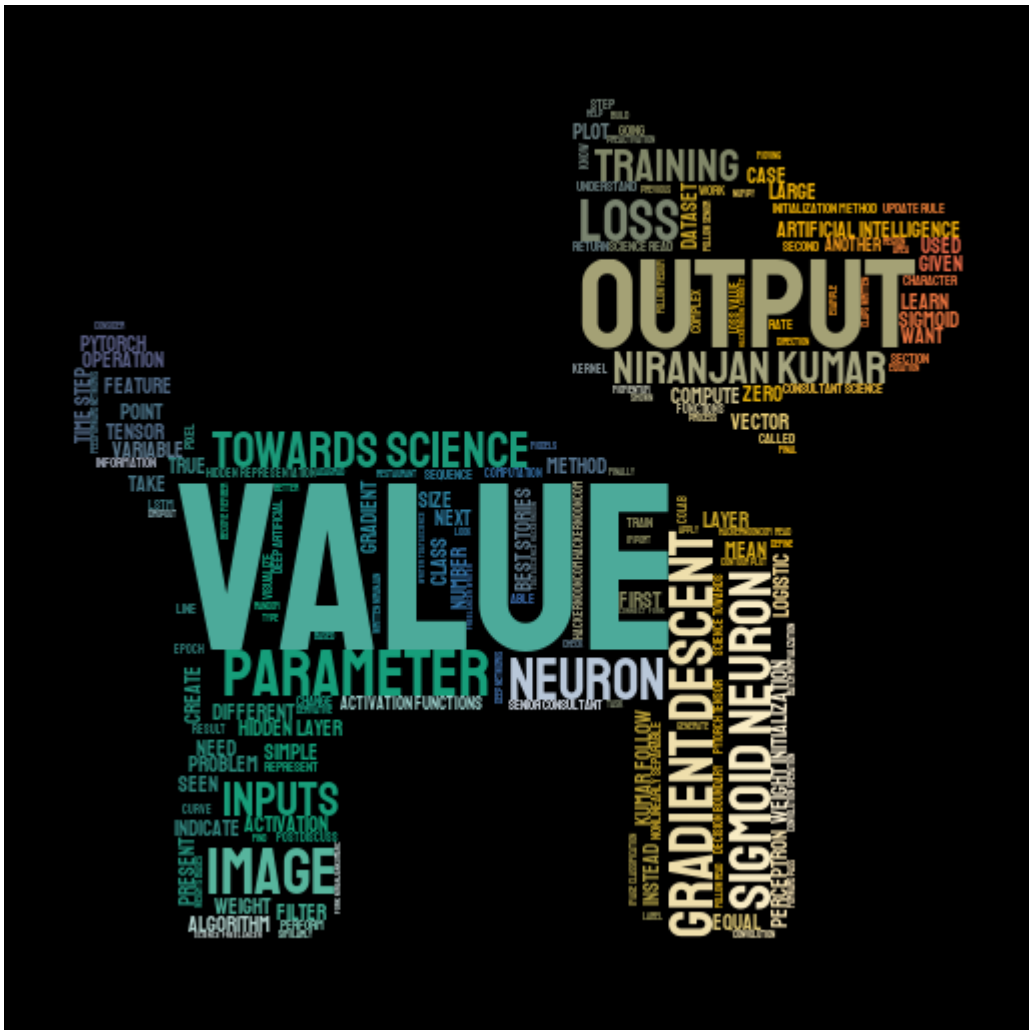
```
pip install stylecloud
```

### Usage

Using `stylecloud` we can generate WordClouds of any shapes and sizes. For example, let's try to create a WordCloud in the shape of a dog with a dark-themed background.

To get the shape of the dog, we just need to change the `icon_name` to '**fas fa-dog**' and by setting `background_color` to '**black**' we will get the dark theme.

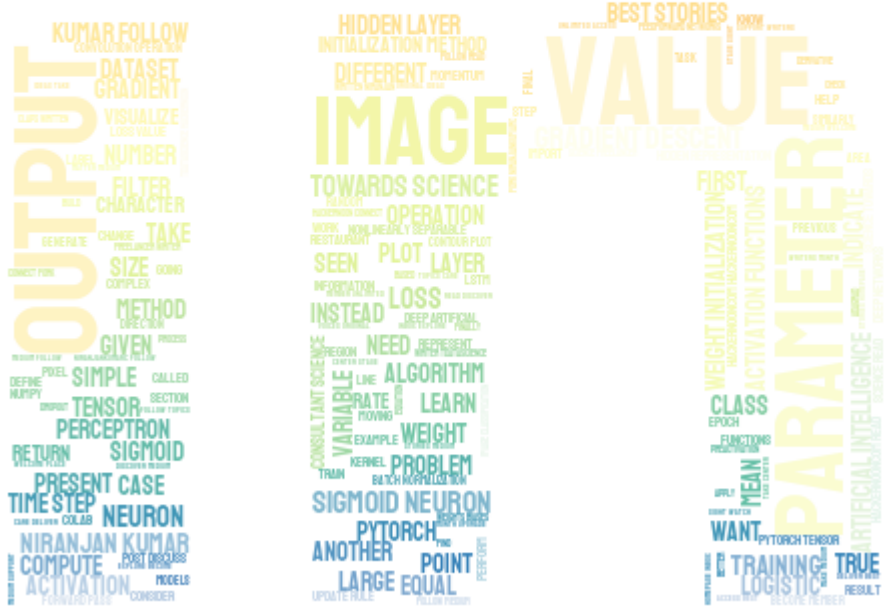
```
import stylecloud

#dog wordcloud
stylecloud.gen_stylecloud(text = cleaned_content,
                          icon_name='fas fa-dog',
                          background_color='black',
                          gradient='horizontal',
output_name='stylecloud_dog.png')
```




Cute Dog

If you want to change the WordCloud to look like a twitter icon or LinkedIn icon, we can do that too just by changing one line of code.

```
#can also generate linkedln wordclouds

stylecloud.gen_stylecloud(text = cleaned_content,
                          icon_name="fab fa-linkedin-in",

palette='colorbrewer.diverging.Spectral_11',
                          background_color='white',
                          gradient='vertical',
output_name='stylecloud_li.png')
```




```
#can also generate twitter wordclouds

stylecloud.gen_stylecloud(text = cleaned_content,
                          icon_name="fab fa-twitter",
                          palette='colorbrewer.sequential.Blues_9',
                          background_color='black',
                          gradient='vertical',
output_name='stylecloud_tw.png')
```


Twitter Icon

. . .

**Niranjankumar-c/DataScienceBlogAnalysis_NLP**

This repository contains the code to scrap the contents all of my articles and process the data using the NLTK Package...

github.com

## Conclusion

In this article, we have discussed two different ways to scrap the data from two different websites based on whether the website is using javascript to dynamically load the site or

not. We have seen how to use selenium web driver to scrap the data. After that, we have discussed various techniques in cleaning the data before analyzing it for insights. From there, we went to create a WordCloud and Barplot to identity the top topics from the corpus. Finally, we have seen how to create a stylish WordCloud using stylecloud package.

Feel free to reach out to me via LinkedIn or twitter if you face any problems while implementing the code present in my GitHub repository.

Until next time Peace :)

NK.

## Author Bio

Niranjan Kumar is Senior Consultant Data Science at Allstate India. He is passionate about Deep Learning and Artificial Intelligence. Apart from writing on Medium, he also writes for Marktechpost.com as a freelance data science writer. Check out his articles here.

You can connect with him on LinkedIn or follow him on Twitter for updates about upcoming articles on deep learning and machine learning.

### References


- Ultimate guide to deal with Text Data (using Python) — for Data Scientists and Engineers
- Data Science Skills: Web scraping using python
- Data Science Skills: Web scraping javascript using python