(https://www.analyticsvidhya.com/blog/)

INTERMEDIATE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/INTERMEDIATE/)

NLP (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/NLP/) PROJECT (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PROJECT/)

PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/)

TEXT (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/TEXT/)

Hands-on NLP Project: A Comprehensive Guide to Information Extraction using Python

ANIRUDDHA BHANDARI (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/ANIRUDDHA/), JUNE 29, 2020 LOGIN TO BOOKMARK THIS ARTIC...

Overview

- Information extraction is a powerful NLP concept that will enable you to parse through any piece of text
- Learn how to perform information extraction using NLP techniques in Python

Introduction

I'm a bibliophile – I love pouring through books in my free time and extracting as much knowledge as I can. But in today's information overload age, the way we read stuff has changed. Most of us tend to skip the entire text, whether that's an article, a book, or a tutorial – and just read the relevant bits of text.

Let me share a personal example around this. I was recently reading an article about India's upcoming tour of Australia when I realized how quickly I reached the end of the text. I skimmed over most of it, reading just the headlines and a few bits about Virat Kohli. Here's a passage from the article:

> On Friday Cricket Australia had announced that the schedule for upcoming India's tour Down Under. The four Tests of the much-anticipated series will be played at Gabba, Adelaide Oval, MCG, and the SCG respectively starting December 3, the CA had said on Thursday.

Can you guess why some of the words are highlighted in red? Yep, I focused on them when reading the article. And this got me thinking – as someone in the Natural Language Processing (NLP) space, could I build a model that could extract these relevant pieces from any given text?

Turns out the answer is yes – thanks to a concept called Information Extraction. We'll learn more about what information extraction is later, but suffice to say it really helped me fine-tune my NLP skills and build a powerful extraction model that I can use for parsing through most articles.

In this article, I present to you my learnings from when I worked on an Information Extraction project. This is a comprehensive guide so strap in for the ride and enjoy the NLP-fuelled journey!

If you haven't read about any NLP concepts before, I suggest starting for the below free course:

- *Introduction to Natural Language Processing (NLP) (https://courses.analyticsvidhya.com/courses/Intro-to-NLP?utm_source=blog&utm_medium=nlp-project-information-extraction)*

Table of Contents

What is Information Extraction?

Text data contains a lot of information but not all of it will be important to you. We might be looking for names of entities, others would want to extract specific relationships between those entities. Our intentions differ according to our requirements.

Imagine having to go through all the legal documents to find legal precedence to validate your current case. Or having to go through all the research papers to find relevant information to cure a disease. There are many more examples like resume harvesting, media analysis, email scanning, etc.

But just imagine having to manually go through all of the textual data and extracting the most relevant information. Clearly, it is an uphill battle and you might even end up skipping some important information.

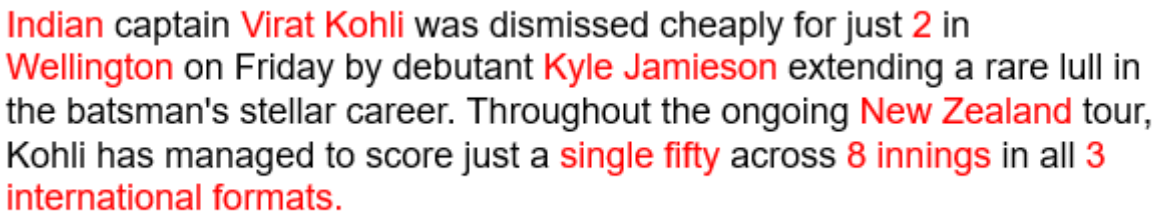
For anyone trying to analyze textual data, the difficult task is not of finding the right documents, but of finding the right information from these documents. Understanding the relationship between entities, understanding how the events have unfolded, or just simply finding hidden gems of information, is clearly what anyone is looking for when they go through a piece of text.

Therefore, coming up with an automated way of extracting the information from textual data and presenting it in a structured manner will help us reap a lot of benefits and tremendously reduce the amount of time we have to spend time skimming through text documents. This is precisely what information extraction strives to achieve.

> *The task of **Information Extraction (IE)** involves extracting meaningful information from unstructured text data and presenting it in a structured format.*

Using information extraction, we can retrieve pre-defined information such as the name of a person, location of an organization, or identify a relation between entities, and save this information in a structured format such as a database.

Let me show you another example I've taken from a cricket news article:



We can extract the following information from the text:

- Country – India, Captain – Virat Kohli
- Batsman – Virat Kohli, Runs – 2
- Bowler – Kyle Jamieson
- Match venue – Wellington
- Match series – New Zealand
- Series highlight – single fifty, 8 innings, 3 formats

This enables us to reap the benefits of powerful query tools like SQL for further analysis. Creating such structured data using information extraction will not only help us in analyzing the documents better but also help us in understanding the hidden relationships in the text.

How does Information Extraction work?

Given the capricious nature of text data that changes depending on the author or the context, Information Extraction seems like a daunting task. But it doesn't have to be that way!

We all know that sentences are made up of words belonging to different **Parts of Speech (POS)** (https://medium.com/analytics-vidhya/part-of-speech-tagging-what-when-why-and-how-9d250e634df6). There are eight different **POS** in the English language: **noun, pronoun, verb, adjective, adverb, preposition, conjunction,** and **intersection**.

The POS determines how a specific word functions in meaning in a given sentence. For example, take the word "right". In the sentence, "The boy was awarded chocolate for giving the right answer", "right" is used as an adjective. Whereas, in the sentence, "You have the right to say whatever you want", "right" is treated as a noun.

This goes to show that the POS tag of a word carries a lot of significance when it comes to understanding the meaning of a sentence. And we can leverage it to extract meaningful information from our text.

Let's take an example to understand this. We'll be using the popular spaCy library (https://www.analyticsvidhya.com/blog/2020/03/spacy-tutorial-learn-natural-language-processing/?utm_source=blog&utm_medium=nlp-project-information-extraction) here.

```
# import spacy
import spacy

# load english language model
nlp = spacy.load('en_core_web_sm',disable=['ner','textcat'])

text = "This is a sample sentence."

# create spacy
doc = nlp(text)

for token in doc:
    print(token.text,'->',token.pos_)
```

st.github.com/aniruddha27/c928abfc6c3934de496d898659572d5b/raw/63615b17ae7e679a8384e65a47ec81ed6939d25b/nlp_ie_1.py)
nlp_ie_1.py (https://gist.github.com/aniruddha27/c928abfc6c3934de496d898659572d5b#file-nlp_ie_1-py) hosted with ❤ by GitHub (https://github.com)

We were easily able to determine the POS tags of all the words in the sentence. But how does it help in Information Extraction?

Well, if we wanted to extract nouns from the sentences, we could take a look at POS tags of the words/tokens in the sentence, using the attribute **.pos_**, and extract them accordingly.

```
for token in doc:
    # check token pos
    if token.pos_=='NOUN':
        # print token
        print(token.text)
```

aw
//gist.github.com/aniruddha27/514ff7736a7fd6bf196662334cafd79a/raw/d25f70a158ffba4efb41c05d6c305de6d9b178e9/nlp_ie_2.py)
nlp_ie_2.py (https://gist.github.com/aniruddha27/514ff7736a7fd6bf196662334cafd79a#file-nlp_ie_2-py) hosted with ❤ by GitHub (https://github.com)

It was that easy to extract words based on their POS tags. But sometimes extracting information purely based on the POS tags is not enough. Have a look at the sentence below:

```
text = "The children love cream biscuits"

# create spacy
doc = nlp(text)

for token in doc:
    print(token.text,'->',token.pos_)
```

ist.github.com/aniruddha27/e01d2115128f5c22aeabc59ebc22d357/raw/e25f1317697c5423df862ad547d05b1871099c16/nlp_ie_3.py) nlp_ie_3.py (https://gist.github.com/aniruddha27/e01d2115128f5c22aeabc59ebc22d357#file-nlp_ie_3-py) hosted with ❤ by **GitHub (https://github.com)**

If I wanted to extract the subject and the object from a sentence, I can't do that based on their POS tags. For that, I need to look at how these words are related to each other. These are called **Dependencies**.

We can make use of spaCy's displacy visualizer (https://explosion.ai/demos/displacy) that displays the word dependencies in a graphical manner:

```
from spacy import displacy
displacy.render(doc, style='dep',jupyter=True)
```

ist.github.com/aniruddha27/77430236550b3a2d7ce2aabd6ab97432/raw/aadf0ab56c6301f0d29742ef04a6570ec5d63d42/nlp_ie_4.py) nlp_ie_4.py (https://gist.github.com/aniruddha27/77430236550b3a2d7ce2aabd6ab97432#file-nlp_ie_4-py) hosted with ❤ by **GitHub (https://github.com)**

Pretty cool! This directed graph is known as a **dependency graph (https://www.analyticsvidhya.com/blog/2017/12/introduction-computational-linguistics-dependency-trees/)**. It represents the relations between different words of a sentence.

Each word is a **node** in the Dependency graph. The relationship between words is denoted by the edges. For example, "The" is a determiner here, "children" is the subject of the sentence, "biscuits" is the object of the sentence, and "cream" is a compound word that gives us more information about the object.

The arrows carry a lot of significance here:

- The **arrowhead** points to the words that are **dependent** on the word pointed by the **origin of the arrow**
- The former is referred to as the **child node** of the latter. For example, "children" is the child node of "love"

- The word which has no incoming arrow is called the **root node** of the sentence

Let's see how we can extract the subject and the object from the sentence. Like we have an attribute for POS in SpaCy tokens, we similarly have an attribute for extracting the dependency of a token denoted by **dep_**:

```
for token in doc:
    # extract subject
    if (token.dep_=='nsubj'):
        print(token.text)
    # extract object
    elif (token.dep_=='dobj'):
        print(token.text)
```

st.github.com/aniruddha27/d5b3a1715e73c06395ed0354f43144ec/raw/6fb04e06768beb299b84b4b9776842838b350e9d/nlp_ie_5.py) nlp_ie_5.py (https://gist.github.com/aniruddha27/d5b3a1715e73c06395ed0354f43144ec#file-nlp_ie_5-py) hosted with ❤ by **GitHub (https://github.com)**

Voila! We have the subject and object of our sentence.

Using POS tags and Dependency tags, we can look for relationships between different entities in a sentence. For example, in the sentence "The **cat** perches **on** the **window sill**", we have the subject, "cat", the object "window sill", related by the preposition "on". We can look for such relationships and much more to extract meaningful information from our text data.

I suggest going through this amazing tutorial (https://www.analyticsvidhya.com/blog/2019/09/introduction-information-extraction-python-spacy/?utm_source=blog&utm_medium=nlp-project-information-extraction) which explains Information Extraction in detail with tons of examples.

Where Do We Go from Here?

We have briefly spoken about the theory regarding Information Extraction which I believe is important to understand before jumping into the crux of this article.

> *“An ounce of practice is generally worth more than a ton of theory.” –E.F. Schumacher*

In the following sections, I am going to explore a text dataset and apply the information extraction technique to retrieve some important information, understand the structure of the sentences, and the relationship between entities.

So, without further ado, let’s get cracking on the code!

Getting Familiar with the Text Dataset

The dataset we are going to be working with is the United Nations General Debate Corpus (https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/0TJX8Y). It contains speeches made by representatives of all the member countries from the year 1970 to 2018 at the General Debate of the annual session of the United Nations General Assembly.

But we will take a subset of this dataset and work with speeches made by India at these debates. This will allow us to stay on track and better understand the task at hand of understanding Information Extraction. This leaves us with 49 speeches made by India over the years, each speech ranging from anywhere between 2000 to 6000+ words.

Having said that, let’s have a look at our dataset:

```
# Folder path
folders = glob.glob('./UNGD/UNGDC 1970-2018/Converted sessions/Session*')

# Dataframe
df = pd.DataFrame(columns={'Country','Speech','Session','Year'})

# Read speeches by India
i = 0
for file in folders:

    speech = glob.glob(file+'/IND*.txt')

    with open(speech[0],encoding='utf8') as f:
```

```
        # Speech
        df.loc[i,'Speech'] = f.read()
        # Year
        df.loc[i,'Year'] = speech[0].split('_')[-1].split('.')[0]
        # Session
        df.loc[i,'Session'] = speech[0].split('_')[-2]
        # Country
        df.loc[i,'Country'] = speech[0].split('_')[0].split("\\")[-1]
        # Increment counter
        i += 1

df.head()
```

ist.github.com/aniruddha27/b656a4b9201f1b4792a510451878cf50/raw/ab4b8c3ef67120b3c4aea47a02e89144c2d5af08/nlp_ie_df.py) nlp_ie_df.py (https://gist.github.com/aniruddha27/b656a4b9201f1b4792a510451878cf50#file-nlp_ie_df-py) hosted with ❤ by **GitHub (https://github.com)**

I will print a snapshot of one of the speeches to give you a feel of what the data looks like:

Now let’s start working with our dataset!

Speech Text Pre-Processing

First, we need to clean our text data. When I went over a few speeches, I found each paragraph in the speech was numbered to distinctly identify it. There were obviously unwanted characters like newline character, a hyphen, salutations, and apostrophes, like in any other text dataset.

But another unique and unwanted information present were the references made in each speech to other documents. We obviously don't want that either.

I have written a simple function to clean the speeches. **An important point here is that I haven't used lemmatization or changed the words to lowercase as it has the potential to change the POS tag of the word.** We certainly don't want to do that as you will see in the upcoming subsections.

```
# function to preprocess speech
def clean(text):

    # removing paragraph numbers
    text = re.sub('[0-9]+.\t','',str(text))
    # removing new line characters
    text = re.sub('\n ','',str(text))
    text = re.sub('\n',' ',str(text))
    # removing apostrophes
    text = re.sub("'s",'',str(text))
    # removing hyphens
    text = re.sub("-",' ',str(text))
    text = re.sub("— ",'',str(text))
    # removing quotation marks
    text = re.sub('\"','',str(text))
    # removing salutations
    text = re.sub("Mr\.",'Mr',str(text))
    text = re.sub("Mrs\.",'Mrs',str(text))
    # removing any reference to outside text
    text = re.sub("[\(\[].*?[\)\]]", "", str(text))

    return text

# preprocessing speeches
df['Speech_clean'] = df['Speech'].apply(clean)
```

Right, now that we have our minimally cleaned speeches, we can split it up into separate sentences.

Split the Speech into Different Sentences

Splitting our speeches into separate sentences will allow us to extract information from each sentence. Later, we can combine it to get cumulative information for any specific year.

```
# split sentences
def sentences(text):
    # split sentences and questions
    text = re.split('[.?]', text)
    clean_sent = []
    for sent in text:
        clean_sent.append(sent)
    return clean_sent

# sentences
df['sent'] = df['Speech_clean'].apply(sentences)
```


Finally, we can create a dataframe containing the sentences from different years:

```
# create a dataframe containing sentences
df2 = pd.DataFrame(columns=['Sent','Year','Len'])

row_list = []

for i in range(len(df)):
    for sent in df.loc[i,'sent']:

        wordcount = len(sent.split())
        year = df.loc[i,'Year']

        dict1 = {'Year':year,'Sent':sent,'Len':wordcount}
        row_list.append(dict1)

```

```
15   df2 = pd.DataFrame(row_list)
```

v
/gist.github.com/aniruddha27/3d4be25ddfaabddef4aa0b1752cacd9f/raw/3a00779f6f5c9e7738e4f18409efa42efb7e7cd1/nlp_ie_27.py)
nlp_ie_27.py (https://gist.github.com/aniruddha27/3d4be25ddfaabddef4aa0b1752cacd9f#file-nlp_ie_27-py) hosted with ❤ by GitHub (https://github.com)

After performing this operation, we end up with 7150 sentences. Going over them and extracting information manually will be a difficult task. That's why we are looking at Information Extraction using NLP techniques!

Information Extraction using SpaCy

Now, we can start working on the task of Information Extraction. We will be using the spaCy (https://www.analyticsvidhya.com/blog/2020/03/spacy-tutorial-learn-natural-language-processing/?utm_source=blog&utm_medium=nlp-project-information-extraction) library for working with the text data. It has all the necessary tools that we can exploit for all the tasks we need for information extraction.

Let me import the relevant SpaCy modules that we will require for the task ahead:

```
import spacy
from spacy.matcher import Matcher

from spacy import displacy
import visualise_spacy_tree
from IPython.display import Image, display

# load english language model
nlp = spacy.load('en_core_web_sm',disable=['ner','textcat'])
```

gist.github.com/aniruddha27/5c83ed1a1e2f891512e6d3f07bd8b36f/raw/6674ca4c2bdc8be782f830021db949c162cb9058/nlp_ie_8.py) nlp_ie_8.py (https://gist.github.com/aniruddha27/5c83ed1a1e2f891512e6d3f07bd8b36f#file-nlp_ie_8-py) hosted with ❤ by GitHub (https://github.com)

We will need the spaCy Matcher class to create a pattern to match phrases in the text. We'll also require the displaCy module for visualizing the dependency graph of sentences.

The **visualise_spacy_tree** library will be needed for creating a tree-like structure out of the Dependency graph. This helps in visualizing the graph in a better way. Finally, IPython Image and display classes are required to output the tree.

But you don't need to worry about these too much. It will become clear as you look at the code.

Information Extraction #1 – Finding Mentions of Prime Minister in the Speech

When working on information extraction tasks, it is important to manually go over a subset of the dataset to understand what the text is like and determine if anything catches your attention at first glance. When I first went over the speeches, I found many of them referred to what the Prime Minister had said, thought, or achieved in the past.

We know that a country is nothing without its leader. The destination a country ends up in is by and large the result of the able guidance of its leader. Therefore, I believe it is important to extract those sentences from the speeches that referred to Prime Ministers of India, and try and understand what their thinking and perspective were, and also try to unravel any common or differing beliefs over the years.

To achieve this task, I used SpaCy's Matcher class (https://spacy.io/api/matcher). It allows us to match a sequence of words based on certain patterns. For the current task, we know that whenever a Prime Minister is referred to in the speech, it will be in one of the following ways:

- Prime Minister of [Country] …

- Prime Minister [Name] …

Using this general understanding, we can come up with a pattern:

Let me walk you through this pattern:

- Here, each dictionary in the list matches a unique word
- The first and second dictionaries match the keyword “Prime Minister” irrespective of whether it is in uppercase or not, which is why I have included the key “LOWER”
- The third dictionary matches a word that is a preposition. What I am looking for here is the word “of”. Now, as discussed before, it may or may not be present in the pattern, therefore, an additional key, “OP” or optional, is mentioned to point out just that
- Finally, the last dictionary in the pattern should be a proper noun. This can either be the name of the country or the name of the prime minister
- The matched keywords have to be in continuation otherwise the pattern will not match the phrase

```
# function to find sentences containing PMs of India
def find_names(text):

    names = []

    # spacy doc
    doc = nlp(text)

    # pattern
```

```
    pattern = [{'LOWER':'prime'},
               {'LOWER':'minister'},
               {'POS':'ADP','OP':'?'},
               {'POS':'PROPN'}]

    # Matcher class object
    matcher = Matcher(nlp.vocab)
    matcher.add("names", None, pattern)

    matches = matcher(doc)

    # finding patterns in the text
    for i in range(0,len(matches)):

        # match: id, start, end
        token = doc[matches[i][1]:matches[i][2]]
        # append token to list
        names.append(str(token))

    # Only keep sentences containing Indian PMs
    for name in names:
        if (name.split()[2] == 'of') and (name.split()[3] != "India"):
                names.remove(name)

    return names

# apply function
df2['PM_Names'] = df2['Speech_clean'].apply(find_names)
```

gist.github.com/aniruddha27/1447f1881ac01cb9d7407eea5d050a95/raw/0ec37a53cf8ad367cb631880e8f0533b70ac83f6/nlp_ie_9.py) nlp_ie_9.py (https://gist.github.com/aniruddha27/1447f1881ac01cb9d7407eea5d050a95#file-nlp_ie_9-py) hosted with ❤ by **GitHub (https://github.com)**

Here are some sample sentences from the year 1989 that matched our pattern:

Now, since only 58 sentences out of 7150 total sentences gave an output that matched our pattern, I have summarised the relevant information from these outputs here:

- PM Indira Gandhi and PM Jawaharlal Nehru believed in working together in unity and with the principles of the UN

- PM Indira Gandhi believed in striking a balance between global production and consumption. She set out policies dedicated to national reconstruction and the consolidation of a secular and pluralistic political system
- PM Indira Gandhi emphasized that India does not intervene in the internal affairs of other countries. However, this stand on foreign policy took a U-turn under PM Rajiv Gandhi when he signed an agreement with the Sri Lankan Prime Minister which brought peace to Sri Lanka
- Both PM Indira Gandhi and PM Rajiv Gandhi believed in the link between economic development and protection of the environment
- PM Rajiv Gandhi advocated for the disarmament of nuclear weapons, a belief that was upheld by India over the years
- Indian, under different PMs, has always extended a hand of peace towards Pakistan over the years
- PM Narendra Modi believes that economic empowerment and upliftment of any nation involves the empowerment of its women
- PM Narendra Modi has launched several schemes that will help India achieve its SGD goals

Using information extraction, we were able to isolate only a few sentences that we required that gave us maximum results.

Information Extraction #2 – Finding Initiatives

The second interesting thing I noticed while going through the speeches is that there were a lot of initiatives, schemes, agreements, conferences, programs, etc. that were mentioned in the speeches. For example, 'Paris agreement', 'Simla Agreement', 'Conference on Security Council', 'Conference of Non Aligned Countries', 'International Solar Alliance', 'Skill India initiative', etc.

Extracting these would give us an idea about what are the priorities for India and whether there is a pattern as to why they are mentioned quite often in the speeches.

I am going to refer to all the schemes, initiatives, conferences, programmes, etc. keywords as initiatives.

To extract initiatives from the text, the first thing I am going to do is identify those sentences that talk about the initiatives. For that, I will use simple regex (https://www.analyticsvidhya.com/blog/2015/06/regular-expression-python/) to select only those sentences that contain the keyword 'initiative', 'scheme', 'agreement', etc. This will reduce our search for the initiative pattern that we are looking for:

```
# to check if keyswords like 'programs','schemes', etc. present in sentences

def prog_sent(text):

```

```
    patterns = [r'\b(?i)'+'plan'+r'\b',
                r'\b(?i)'+'programme'+r'\b',
                r'\b(?i)'+'scheme'+r'\b',
                r'\b(?i)'+'campaign'+r'\b',
                r'\b(?i)'+'initiative'+r'\b',
                r'\b(?i)'+'conference'+r'\b',
                r'\b(?i)'+'agreement'+r'\b',
                r'\b(?i)'+'alliance'+r'\b']

    output = []
    flag = 0
    for pat in patterns:
        if re.search(pat, text) != None:
            flag = 1
            break
    return flag

# apply function
df2['Check_Schemes'] = df2['Sent'].apply(prog_sent)
```


Now, you might be thinking that our task is done here as we have already identified the sentences. We can easily look these up and determine what is being talked about in these sentences. But, think about it, not all of these will contain the initiative name. Some of these might be generally talking about initiatives but no initiative name might be present in them.

Therefore, we need to come up with a better solution that extracts only those sentences that contain the initiative names. For that, I am going to use the spaCy Matcher, once again, to come up with a pattern that matches these initiatives.

Have a look at the following example sentences and see if you can come up with a pattern to extract these initiatives:

As you might have noticed, the initiative name is a proper noun that starts with a determiner and ends with either ‘initiative’/’programme’/’agreement’ etc. words in the end. It also includes an occasional preposition in the middle. I also noticed that most of the initiative names were between two to five words long. Keeping this in mind, I came up with the following pattern to match the initiative names:

```
# to extract initiatives using pattern matching
```

```
def all_schemes(text,check):

    schemes = []

    doc = nlp(text)

    # initiatives
    prog_list = ['programme','scheme',
                 'initiative','campaign',
                 'agreement','conference',
                 'alliance','plan']

    # pattern to match initiatives names
    pattern = [{'POS':'DET'},
               {'POS':'PROPN','DEP':'compound'},
               {'POS':'PROPN','DEP':'compound'},
               {'POS':'PROPN','OP':'?'},
               {'POS':'PROPN','OP':'?'},
               {'POS':'PROPN','OP':'?'},
               {'LOWER':{'IN':prog_list},'OP':'+'}
              ]

    if check == 0:
        # return blank list
        return schemes

    # Matcher class object
    matcher = Matcher(nlp.vocab)
    matcher.add("matching", None, pattern)
    matches = matcher(doc)

    for i in range(0,len(matches)):

        # match: id, start, end
        start, end = matches[i][1], matches[i][2]

        if doc[start].pos_=='DET':
            start = start+1

        # matched string
        span = str(doc[start:end])
```

```
43
44          if (len(schemes)!=0) and (schemes[-1] in span):
45              schemes[-1] = span
46          else:
47              schemes.append(span)
48
49      return schemes
50
51  # apply function
52  df2['Schemes1'] = df2.apply(lambda x:all_schemes(x.Sent,x.Check_Schemes),axis=1)
```

st.github.com/aniruddha27/83d917d9061fe525d8b90f35b9addb43/raw/e9984b8a903cee5ce6a8971d5f944806fb9266d2/nlp_ie_11.py) nlp_ie_11.py (https://gist.github.com/aniruddha27/83d917d9061fe525d8b90f35b9addb43#file-nlp_ie_11-py) hosted with ❤ by **GitHub (https://github.com)**

We got 62 sentences that match our pattern – not bad. Have a look at the output from the year 2018:

But one thing I must point out here is that there were a lot more initiatives in the speeches that did not match our pattern. For example, in the year 2018, there were other initiatives too like “MUDRA”, ”Ujjwala”, ”Paris Agreement”, etc. So is there a better way to extract them?

Remember how we looked at dependencies at the beginning of the article? Well, we are going to use those to make some rules to match the initiative name. But before making a rule, you need to understand how a sentence is structured, only then can you come up with a general rule to extract relevant information.

To understand the structure of the sentence I am going to print the dependency graph of a sample example but in a tree fashion which gives a better intuition of the structure. Have a look below:

```
doc = nlp(' Last year, I spoke about the Ujjwala programme , through which, I am happy to report, 5
png = visualise_spacy_tree.create_png(doc)
display(Image(png))
```

st.github.com/aniruddha27/4474e7b9c77524a8a664039322fb0a3f/raw/a3289b52aee110649873e281c8a67f4cc86159b6/nlp_ie_12.py)
nlp_ie_12.py (https://gist.github.com/aniruddha27/4474e7b9c77524a8a664039322fb0a3f#file-nlp_ie_12-py) hosted with ❤ by **GitHub (https://github.com)**

See how ‘Ujjwala’ is a child node of ‘programme’. Have a look at another example:

Notice how the ‘International Solar Alliance’ is structured.

You must have got the idea by now that the initiative names are usually children of nodes that contain words like ‘initiative’, ‘programme’, etc. Based on this knowledge we can develop our own rule.

The rule I am suggesting is pretty simple. Let me walk you through it:

- I am going to look for tokens in sentences that contain my initiative keywords
- Then I am going to look at its subtree (or words dependent on it) using *token.subtree* and extract only those nodes/words that are proper nouns, since they are most likely going to contain the name of the initiative

```
# rule to extract initiative name
def sent_subtree(text):

    # pattern match for schemes or initiatives
    patterns = [r'\b(?i)'+'plan'+r'\b',
           r'\b(?i)'+'programme'+r'\b',
           r'\b(?i)'+'scheme'+r'\b',
           r'\b(?i)'+'campaign'+r'\b',
           r'\b(?i)'+'initiative'+r'\b',
           r'\b(?i)'+'conference'+r'\b',
           r'\b(?i)'+'agreement'+r'\b',
           r'\b(?i)'+'alliance'+r'\b']

    schemes = []
    doc = nlp(text)
    flag = 0
    # if no initiative present in sentence
    for pat in patterns:

        if re.search(pat, text) != None:
            flag = 1
            break

    if flag == 0:
        return schemes

    # iterating over sentence tokens
    for token in doc:

        for pat in patterns:

            # if we get a pattern match
            if re.search(pat, token.text) != None:

                word = ''
                # iterating over token subtree
```

```
                for node in token.subtree:
                    # only extract the proper nouns
                    if (node.pos_ == 'PROPN'):
                        word += node.text+' '

                if len(word)!=0:
                    schemes.append(word)

    return schemes

# derive initiatives
df2['Schemes2'] = df2['Sent'].apply(sent_subtree)
```

st.github.com/aniruddha27/a39e114707cdb0ba194da3125a0cc903/raw/3f9fbddd2d16a30de3d695aa52a28923c4222af0/nlp_ie_13.py)
nlp_ie_13.py (https://gist.github.com/aniruddha27/a39e114707cdb0ba194da3125a0cc903#file-nlp_ie_13-py) hosted with ❤ by **GitHub (https://github.com)**

This time we matched 282 entries. That is a significant improvement over the previous result. Let's go over the 2018 output and see if we did any better this time:

Out of 7000+ sentences, we were able to zero down to just 282 sentences that talked about initiatives. I looped over these outputs and below is how I would summarise the output:

- There are a lot of different international initiatives or schemes that India has mentioned in its speeches. This goes to show that India has been an active member of the international community working towards building a better future by solving problems through these initiatives

- Another point to highlight here is that the initiatives mentioned in the initial years have been more focused on those that concern the international community. However, during recent times, especially after 2014, a lot of domestic initiatives have been mentioned in the speeches like ‘Ayushman Bharat’, ‘Pradhan Mantri Jan Dhan Yojana’, etc. This shows a shift in how the country perceives its role in the community. By mentioning a lot of domestic initiatives, India has started to put more of the domestic work in front of the international community to witness and, probably, even follow in their footsteps

Having said that, the results were definitely not perfect. There were instances when unwanted words were also getting extracted with the initiative names. But the output derived by making our own rules was definitely better than the ones derived by using SpaCy’s pattern matcher. This goes to show the flexibility we can achieve by making our own rules.

Finding Patterns in the Speeches

So far, we extracted only that information that met our analytical eye when we skimmed over the data. But is there any other information hidden in our dataset? Surely there is and we are going to explore that by making our own rules using the dependency of the words, as we did in the previous section.

But before that, I want to point out two things.

First, when we are trying to understand the structure of the speech, we cannot look at the entire speech, that would take an eternity, and time is of the essence here. What we are going to do instead is look at random sentences from the dataset and then, based on their structure, try to come up with general rules to extract information.

But how do we test the validity of these rules? That’s where my second point comes in! Not all of the rules that we come up with will yield satisfactory results. So, to sift out the irrelevant rules, we can look at the percentage of sentences that matched our rule out of all sentences. This will give us a fair idea about how well the rule is performing, and whether, in fact, there is any such general structure in the corpus!

Another very important point that needs to be highlighted here is that any corpus is bound to contain long complex sentences. Working with these sentences to try and understand their structure will be a very difficult task. Therefore, we are going to look at smaller sentences. This will give us the opportunity to better understand their structure. So what’s the magic number? Let’s first look at how the sentence length varies in our corpus.

Looking at the histogram, we can see that most of the sentences range from 15-20 words. So I am going to work with sentences that contain no more than 15 words:

```
row_list = []
# df2 contains all sentences from all speeches
for i in range(len(df2)):
    sent = df2.loc[i,'Sent']

    if (',' not in sent) and (len(sent.split()) <= 15):

        year = df2.loc[i,'Year']
        length = len(sent.split())

        dict1 = {'Year':year,'Sent':sent,'Len':length}
        row_list.append(dict1)

# df with shorter sentences
df3 = pd.DataFrame(columns=['Year','Sent',"Len"])
df3 = pd.DataFrame(row_list)
```

Now, let's write a simple function that will generate random sentences from this dataframe:

```
from random import randint
def rand_sent(df):

    index = randint(0, len(df))
    print('Index = ',index)
    doc = nlp(df.loc[index,'Sent'][1:])
    displacy.render(doc, style='dep',jupyter=True)

    return index
```

t.github.com/aniruddha27/37ed808e86919d0362c5b1e077ced095/raw/793d8fb5ae9b3c68573a487a1b0be79440b81fce/nlp_ie_15.py)
nlp_ie_15.py (https://gist.github.com/aniruddha27/37ed808e86919d0362c5b1e077ced095#file-nlp_ie_15-py) hosted with ❤ by **GitHub (https://github.com)**

Finally, let's make a function to evaluate the result of our rule:

```
# function to check output percentage for a rule
def output_per(df,out_col):

    result = 0

    for out in df[out_col]:
        if len(out)!=0:
            result+=1

    per = result/len(df)
    per *= 100

    return per
```

ist.github.com/aniruddha27/eb7f892bc3b1d955c5d5f94be379dc53/raw/2aaa5774639ea2e8ef5b4684f82922d40b516bcf/nlp_ie_16.py)
nlp_ie_16.py (https://gist.github.com/aniruddha27/eb7f892bc3b1d955c5d5f94be379dc53#file-nlp_ie_16-py) hosted with ❤ by **GitHub (https://github.com)**

Right, let's get down to the business of making some rules!

Information Extraction #3 – Rule on Noun-Verb-Noun Phrases

When you look at a sentence, it generally contains a **subject (noun), action (verb), and an object (noun)**. The rest of the words are just there to give us additional information about the entities. Therefore, we can leverage this basic structure to extract the main bits of information from the sentence. Take for example the following sentence:

What will be extracted from this sample sentence based on the rule is – “countries face threats”. This should give us a fair idea about what the sentence is trying to say.

So let’s look at how this rule fairs what we run it against the short sentences that we are working with:

```
# function for rule 1: noun(subject), verb, noun(object)
def rule1(text):

    doc = nlp(text)

    sent = []

    for token in doc:

        # if the token is a verb
        if (token.pos_=='VERB'):

            phrase =''

            # only extract noun or pronoun subjects
            for sub_tok in token.lefts:

                if (sub_tok.dep_ in ['nsubj','nsubjpass']) and (sub_tok.pos_ in ['NOUN','PROPN','P

                    # add subject to the phrase
                    phrase += sub_tok.text

                    # save the root of the verb in phrase
                    phrase += ' '+token.lemma_

                    # check for noun or pronoun direct objects
                    for sub_tok in token.rights:

                        # save the object in the phrase
                        if (sub_tok.dep_ in ['dobj']) and (sub_tok.pos_ in ['NOUN','PROPN']):

                            phrase += ' '+sub_tok.text
                            sent.append(phrase)

    return sent
```

nlp_ie_18.py (https://gist.github.com/aniruddha27/a7435e199bf27478d8a18d530667e6bf#file-nlp_ie_18-py) hosted with ❤ by

gist.github.com/aniruddha27/a7435e199bf27478d8a18d530667e6bf/raw/efb24b35880febfff0d6af366347d7891423ce9a/nlp_ie_18.py) GitHub (https://github.com)

```
# create a df containing sentence and its output for rule 1
row_list = []

for i in range(len(df3)):

    sent = df3.loc[i,'Sent']
    year = df3.loc[i,'Year']
    output = rule1(sent)
    dict1 = {'Year':year,'Sent':sent,'Output':output}
    row_list.append(dict1)

df_rule1 = pd.DataFrame(row_list)

# rule 1 achieves 20% result on simple sentences
output_per(df_rule1,'Output')
```

ist.github.com/aniruddha27/5d881ce076a510f34662bdb7cd5ef9fa/raw/9a14fad748ea85cde254380abf4824b40b481310/nlp_ie_19.py) nlp_ie_19.py (https://gist.github.com/aniruddha27/5d881ce076a510f34662bdb7cd5ef9fa#file-nlp_ie_19-py) hosted with ❤ by GitHub (https://github.com)

We are getting more than 20% pattern match for our rule and we can check it for all the sentences in the corpus:

```
# create a df containing sentence and its output for rule 1
row_list = []

# df2 contains all the sentences from all the speeches
for i in range(len(df2)):

    sent = df2.loc[i,'Sent']
    year = df2.loc[i,'Year']
    output = rule1(sent)
    dict1 = {'Year':year,'Sent':sent,'Output':output}
    row_list.append(dict1)

df_rule1_all = pd.DataFrame(row_list)

# check rule1 output on complete speeches
output_per(df_rule1_all,'Output')
```

nlp_ie_20.py (https://gist.github.com/aniruddha27/fff561c3b929af6dde900840703ba9fd#file-nlp_ie_20-py) hosted with ❤ by GitHub
gist.github.com/aniruddha27/fff561c3b929af6dde900840703ba9fd/raw/ad5e26c6f1f8a9d5d3aa9d7ba2176c760b771bcb/nlp_ie_20.py)
(https://github.com)

We are getting more than a 30% match for our rules, which means 2226 out of 7150 sentences matched this pattern. Let's form a new dataframe containing only those sentences that have an output and then segregate the verb from the nouns:

```
# selecting non-empty output rows
df_show = pd.DataFrame(columns=df_rule1_all.columns)

for row in range(len(df_rule1_all)):

    if len(df_rule1_all.loc[row,'Output'])!=0:
        df_show = df_show.append(df_rule1_all.loc[row,:])

# reset the index
df_show.reset_index(inplace=True)
df_show.drop('index',axis=1,inplace=True)
```

st.github.com/aniruddha27/7d47c6de0a21225061e91d87ff007394/raw/adc18757d6137781f07377249ee664fef856e134/nlp_ie_28.py)
nlp_ie_28.py (https://gist.github.com/aniruddha27/7d47c6de0a21225061e91d87ff007394#file-nlp_ie_28-py) hosted with ❤ by
GitHub (https://github.com)

```
# separate subject, verb and object

verb_dict = dict()
dis_dict = dict()
dis_list = []

# iterating over all the sentences
for i in range(len(df_show)):

    # sentence containing the output
    sentence = df_show.loc[i,'Sent']
    # year of the sentence
    year = df_show.loc[i,'Year']
    # output of the sentence
    output = df_show.loc[i,'Output']

    # iterating over all the outputs from the sentence
    for sent in output:
```

```
19
20          # separate subject, verb and object
21          n1, v, n2 = sent.split()[:1], sent.split()[1], sent.split()[2:]
22
23          # append to list, along with the sentence
24          dis_dict = {'Sent':sentence,'Year':year,'Noun1':n1,'Verb':v,'Noun2':n2}
25          dis_list.append(dis_dict)
26
27          # counting the number of sentences containing the verb
28          verb = sent.split()[1]
29          if verb in verb_dict:
30              verb_dict[verb]+=1
31          else:
32              verb_dict[verb]=1
33
34  df_sep = pd.DataFrame(dis_list)
```

Let’s take a look at the top 10 most occurring verbs used in the sentences:

Now, we can look at specific verbs to see what kind of information is present. For example, ‘welcome’ and ‘support’ could tell us what India encourages. And verbs like ‘face’ could maybe tell us what kind of problems we face in the real world.

By looking at the output, we can try to make out what is the context of the sentence. For example, we can see that India supports ‘efforts’, ‘viewpoints’, ‘initiatives’, ‘struggles’, ‘desires, ‘aspirations’, etc. While India believes that the world faces ‘threat’, ‘conflicts’, ‘colonialism’, ‘pandemics’, etc.

We can select sentences to explore in-depth by looking at the output. This will definitely save us a lot of time than just going over the entire text.

Information Extraction #4 – Rule on Adjective Noun Structure

In the previous rule that we made, we extracted the noun subjects and objects, but the information did not feel complete. This is because many nouns have an adjective or a word with a compound dependency that augments the meaning of a noun. Extracting these along with the noun will give us better information about the subject and the object.

Have a look at the sample sentence below:

What we are looking to achieve here is – “better life”.

The code for this rule is simple, but let me walk you through how it works:

- We look for tokens that have a Noun POS tag and have subject or object dependency
- Then we look at the child nodes of these tokens and append it to the phrase only if it modifies the noun

```
# function for rule 2
def rule2(text):

    doc = nlp(text)

    pat = []

    # iterate over tokens
    for token in doc:
        phrase = ''
        # if the word is a subject noun or an object noun
        if (token.pos_ == 'NOUN')\
            and (token.dep_ in ['dobj','pobj','nsubj','nsubjpass']):

            # iterate over the children nodes
            for subtoken in token.children:
                # if word is an adjective or has a compound dependency
                if (subtoken.pos_ == 'ADJ') or (subtoken.dep_ == 'compound'):
                    phrase += subtoken.text + ' '

            if len(phrase)!=0:
                phrase += token.text

        if  len(phrase)!=0:
            pat.append(phrase)


    return pat
```

jist.github.com/aniruddha27/91ae8334f1f6c304310ddc43c2b746d3/raw/c4bb75f73189f4338aac08698a1f6d4a82f40c2c/nlp_ie_21.py)
nlp_ie_21.py (https://gist.github.com/aniruddha27/91ae8334f1f6c304310ddc43c2b746d3#file-nlp_ie_21-py) hosted with ❤ by **GitHub (https://github.com)**

```
# create a df containing sentence and its output for rule 2
row_list = []

for i in range(len(df3)):

    sent = df3.loc[i,'Sent']
    year = df3.loc[i,'Year']
    # rule
```

```
 9       output = rule2(sent)
10
11       dict1 = {'Year':year,'Sent':sent,'Output':output}
12       row_list.append(dict1)
13
14   df_rule2 = pd.DataFrame(row_list)
```

ist.github.com/aniruddha27/2d2126774f940e653e6b6437e48ee974/raw/19c489cf74f92e9145aaf6f22cb2211c3b6b417d/nlp_ie_30.py) nlp_ie_30.py (https://gist.github.com/aniruddha27/2d2126774f940e653e6b6437e48ee974#file-nlp_ie_30-py) hosted with ❤ by **GitHub (https://github.com)**

51% of the short sentences match this rule. We can now try to check it on the entire corpus:

```
# create a df containing sentence and its output for rule 2
row_list = []

# df2 contains all the sentences from all the speeches
for i in range(len(df2)):

    sent = df2.loc[i,'Sent']
    year = df2.loc[i,'Year']
    output = rule2(sent)
    dict1 = {'Year':year,'Sent':sent,'Output':output}
    row_list.append(dict1)

df_rule2_all = pd.DataFrame(row_list)

# check rule output on complete speeches
output_per(df_rule2_all,'Output')
```


On the entire corpus of 7150, 76% or 5117 sentences matched our pattern rule, since most of them are bound to contain the noun and its modifier.

```
# selecting non-empty outputs
df_show2 = pd.DataFrame(columns=df_rule2_all.columns)

for row in range(len(df_rule2_all)):

    if len(df_rule2_all.loc[row,'Output'])!=0:
        df_show2 = df_show2.append(df_rule2_all.loc[row,:])

# reset the index
df_show2.reset_index(inplace=True)
df_show2.drop('index',axis=1,inplace=True)
```

Now we can combine this rule along with the rule we created previously. This will give us a better perspective of what information is present in a sentence:

```
def rule2_mod(text,index):

    doc = nlp(text)

    phrase = ''

    for token in doc:

        if token.i == index:
```

```
10
11              for subtoken in token.children:
12                  if (subtoken.pos_ == 'ADJ'):
13                      phrase += ' '+subtoken.text
14              break
15
16      return phrase
```

ist.github.com/aniruddha27/07e760dd5b53cef35b52652bafa5f687/raw/519201df6964b6ea70e0694e49d6872a60e5eff5/nlp_ie_22.py) nlp_ie_22.py (https://gist.github.com/aniruddha27/07e760dd5b53cef35b52652bafa5f687#file-nlp_ie_22-py) hosted with ❤ by GitHub (https://github.com)

```
1   # rule 1 modified function
2   def rule1_mod(text):
3
4       doc = nlp(text)
5
6       sent = []
7
8       for token in doc:
9           # root word
10          if (token.pos_=='VERB'):
11
12              phrase =''
13
14              # only extract noun or pronoun subjects
15              for sub_tok in token.lefts:
16
17                  if (sub_tok.dep_ in ['nsubj','nsubjpass']) and (sub_tok.pos_ in ['NOUN','PROPN','P
18
19                      # look for subject modifier
20                      adj = rule2_mod(text,sub_tok.i)
21
22                      phrase += adj + ' ' + sub_tok.text
23
24                      # save the root word of the word
25                      phrase += ' '+token.lemma_
26
27                      # check for noun or pronoun direct objects
28                      for sub_tok in token.rights:
29
```

```
                        if (sub_tok.dep_ in ['dobj']) and (sub_tok.pos_ in ['NOUN','PROPN']):

                            # look for object modifier
                            adj = rule2_mod(text,sub_tok.i)

                            phrase += adj+' '+sub_tok.text
                            sent.append(phrase)

    return sent
```

st.github.com/aniruddha27/d796631e54d8b49c906aa03bfc5640e5/raw/0bf2197f794145798b0450477728b505f227433f/nlp_ie_23.py) nlp_ie_23.py (https://gist.github.com/aniruddha27/d796631e54d8b49c906aa03bfc5640e5#file-nlp_ie_23-py) hosted with ❤ by **GitHub (https://github.com)**

```
# create a df containing sentence and its output for modified rule 1
row_list = []

# df2 contains all the sentences from all the speeches
for i in range(len(df2)):

    sent = df2.loc[i,'Sent']
    year = df2.loc[i,'Year']
    output = rule1_mod(sent)
    dict1 = {'Year':year,'Sent':sent,'Output':output}
    row_list.append(dict1)

df_rule1_mod_all = pd.DataFrame(row_list)
# check rule1 output on complete speeches
output_per(df_rule1_mod_all,'Output')
```

t.github.com/aniruddha27/892523c05814ce7e6599531d83e7b203/raw/ba29db215ad81c0941221669683c692b3f7d8efd/nlp_ie_33.py) nlp_ie_33.py (https://gist.github.com/aniruddha27/892523c05814ce7e6599531d83e7b203#file-nlp_ie_33-py) hosted with ❤ by **GitHub (https://github.com)**

We get a 31% output match some of which are displayed below:

Here, we end up with phrases like “we take a fresh pledge”, “we have a sizeable increase”, “people expecting better life”, etc. which included the nouns and their modifiers. This gives us better information about what is being extracted here.

As you can see, we not only came up with a new rule to understand the structure of the sentences but also combined two rules to get better information from the extracted text.

Information Extraction #5 – Rule on Prepositions

Thank god for prepositions! They tell us where or when something is in a relationship with something else. For example, *The people **of** India believe **in** the principles **of** the United Nations.* Clearly extracting phrases including prepositions will give us a lot of information from the sentence. This is exactly what we are going to achieve with this rule.

Let's try to understand how this rule works by going over it on a sample sentence – "India has once again shown faith in democracy."

- We iterate over all the tokens looking for prepositions. For example, ***in*** this sentence
- On encountering a preposition, we check if it has a headword that is a noun. For example, the word ***faith*** in this sentence
- Then we look at the child tokens of the preposition token falling on its right side. For example, the word ***democracy***

This should finally extract the phrase ***faith in democracy*** from the sentence. Have a look at the dependency graph of the sentence below:

Now let’s apply this rule to our short sentences:

```
# rule 3 function
def rule3(text):

    doc = nlp(text)
```

```
5
6       sent = []
7
8       for token in doc:
9
10          # look for prepositions
11          if token.pos_=='ADP':
12
13              phrase = ''
14
15              # if its head word is a noun
16              if token.head.pos_=='NOUN':
17
18                  # append noun and preposition to phrase
19                  phrase += token.head.text
20                  phrase += ' '+token.text
21
22                  # check the nodes to the right of the preposition
23                  for right_tok in token.rights:
24                      # append if it is a noun or proper noun
25                      if (right_tok.pos_ in ['NOUN','PROPN']):
26                          phrase += ' '+right_tok.text
27
28                  if len(phrase)>2:
29                      sent.append(phrase)
30
31      return sent
```

gist.github.com/aniruddha27/ed39b4de3e5bd4ba1d2af92f3dcceeed/raw/535fa6bc5c9980bf99b27cffeac02b3b0d46465c/nlp_ie_24.py)
nlp_ie_24.py (https://gist.github.com/aniruddha27/ed39b4de3e5bd4ba1d2af92f3dcceeed#file-nlp_ie_24-py) hosted with ❤ by **GitHub (https://github.com)**

```
1   # create a df containing sentence and its output for rule 3
2   row_list = []
3
4   for i in range(len(df3)):
5
6       sent = df3.loc[i,'Sent']
7       year = df3.loc[i,'Year']
8
9       # rule
```

```
        output = rule3(sent)

        dict1 = {'Year':year,'Sent':sent,'Output':output}
        row_list.append(dict1)

    df_rule3 = pd.DataFrame(row_list)
    # output percentage for rule 3
    output_per(df_rule3,'Output')
```

gist.github.com/aniruddha27/03345faa5f83b6110dc0515b17a148a2/raw/adfeb235d59fbc07fe836f6b9633ea47b27a6f05/nlp_ie_34.py) nlp_ie_34.py (https://gist.github.com/aniruddha27/03345faa5f83b6110dc0515b17a148a2#file-nlp_ie_34-py) hosted with ❤ by **GitHub (https://github.com)**

About 48% of the sentences follow this rule:

We can test this pattern on the entire corpus since we have a good amount of sentences matching the rule:

```
# create a df containing sentence and its output for rule 3
row_list = []

# df2 contains all the sentences from all the speeches
for i in range(len(df2)):

    sent = df2.loc[i,'Sent']
    year = df2.loc[i,'Year']
    output = rule3(sent)
    dict1 = {'Year':year,'Sent':sent,'Output':output}
    row_list.append(dict1)

df_rule3_all = pd.DataFrame(row_list)
# check rule3 output on complete speeches
output_per(df_rule3_all,'Output')
```

st.github.com/aniruddha27/8b02558f244afed3aa75496c26314383/raw/ca1a156c4c566671b59d25f8a29c247e24001df8/nlp_ie_35.py)
nlp_ie_35.py (https://gist.github.com/aniruddha27/8b02558f244afed3aa75496c26314383#file-nlp_ie_35-py) hosted with ❤ by
GitHub (https://github.com)

74% of the total sentences match this pattern. Let's separate the preposition from the nouns and see what kind of information we were able to extract:

```
# select non-empty outputs
df_show3 = pd.DataFrame(columns=df_rule3_all.columns)

for row in range(len(df_rule3_all)):

    if len(df_rule3_all.loc[row,'Output'])!=0:
        df_show3 = df_show3.append(df_rule3_all.loc[row,:])

# reset the index
df_show3.reset_index(inplace=True)
df_show3.drop('index',axis=1,inplace=True)
```

st.github.com/aniruddha27/d9b5224764dd19daa6cf49dc8eea161b/raw/dc8db50743a24cdd575da5cac9a386ceab82a935/nlp_ie_36.py)
nlp_ie_36.py (https://gist.github.com/aniruddha27/d9b5224764dd19daa6cf49dc8eea161b#file-nlp_ie_36-py) hosted with ❤ by
GitHub (https://github.com)

```
# separate noun, preposition and noun

prep_dict = dict()
```

```
dis_dict = dict()
dis_list = []

# iterating over all the sentences
for i in range(len(df_show3)):

    # sentence containing the output
    sentence = df_show3.loc[i,'Sent']
    # year of the sentence
    year = df_show3.loc[i,'Year']
    # output of the sentence
    output = df_show3.loc[i,'Output']

    # iterating over all the outputs from the sentence
    for sent in output:

        # separate subject, verb and object
        n1, p, n2 = sent.split()[0], sent.split()[1], sent.split()[2:]

        # append to list, along with the sentence
        dis_dict = {'Sent':sentence,'Year':year,'Noun1':n1,'Preposition':p,'Noun2':n2}
        dis_list.append(dis_dict)

        # counting the number of sentences containing the verb
        prep = sent.split()[1]
        if prep in prep_dict:
            prep_dict[prep]+=1
        else:
            prep_dict[prep]=1

df_sep3= pd.DataFrame(dis_list)
```

st.github.com/aniruddha27/0decb9b6f8632a209b9ab054b68b0099/raw/f5016c2a6dbed0e44ea3e48672de0e606e74aeb2/nlp_ie_37.py) nlp_ie_37.py (https://gist.github.com/aniruddha27/0decb9b6f8632a209b9ab054b68b0099#file-nlp_ie_37-py) hosted with ❤ by **GitHub (https://github.com)**

The following dataframe shows the result of the rule on the entire corpus, but the preposition and nouns are separated for better analysis:

We can look at the top 10 most occurring prepositions in the entire corpus:

```
sort = sorted(prep_dict.items(), key = lambda d:(d[1],d[0]), reverse=True)
sort[:10]
```

st.github.com/aniruddha27/4f1017dc71cfa92e0c89195c33b1c896/raw/c3e36dfba01692f0de1fde06613e854fdd650f84/nlp_ie_prep.py) nlp_ie_prep.py (https://gist.github.com/aniruddha27/4f1017dc71cfa92e0c89195c33b1c896#file-nlp_ie_prep-py) hosted with ❤ by **GitHub (https://github.com)**

We look at certain prepositions to explore the sentences in detail. For example, the preposition ‘against’ can give us information about what India does not support:

Skimming over the nouns, some important phrases like:

- efforts against proliferation
- fight against terrorism, action against terrorism, the war against terrorism
- discrimination against women
- war against poverty
- struggle against colonialism

… and so on. This should give us a fair idea about which sentences we want to explore in detail. For example, *efforts against proliferation* talk about efforts towards nuclear disarmament. Or the sentence on the *struggle against colonialism* talks about the historical links between India and Africa borne out of their common struggle against colonialism.

```
df_sep3.loc[11272,'Sent']
```

```
'  This year, 25 years after Prime Minister Rajiv Gandhi  put forward a comprehensive action plan for a nuclear  weapon fre
e and non violent world order, we must  strengthen efforts against nuclear proliferation and  pursue time bound, universal,
non discriminatory,  phased and verifiable nuclear disarmament'
```

```
df_sep3.loc[11513,'Sent']
```

```
' Africa is a region with which we have historical bonds, a solidarity born out of a common struggle against colonialism an
d a belief in a future of shared prosperity'
```

```
df_sep3.loc[11618,'Sent']
```

```
' We need to forget our prejudices and join hands to draw up an effective strategy against terror'
```

```
df_sep3.loc[11859,'Sent']
```

```
' It has become something of a habit for Pakistan to throw the dust of deceit and deception up against India in order to pr
ovide some thin cover for its own guilt'
```

As you can see, prepositions give us an important relationship between two nouns. And with a little domain knowledge, we can easily sieve through the vast data and determine what India supports or does not support, among other things.

But the output seems a bit incomplete. For example, in the sentence *efforts against proliferation*, what kind of *proliferation* are we talking about? Certainly, we need to include the modifiers attached to the nouns in the phrase as we did in Information Extraction #4. This would definitely increase the comprehensibility of the extracted phrase.

This rule can be easily modified to include the new change. I have created a new function to extract the noun modifiers for nouns that we extracted from Information Extraction #4:

```
1  # rule 0
2  def rule0(text, index):
3
4      doc = nlp(text)
5
6      token = doc[index]
7
8      entity = ''
9
```

```
    for sub_tok in token.children:
        if (sub_tok.dep_ in ['compound','amod']):
            entity += sub_tok.text+' '

    entity += token.text

    return entity
```

ist.github.com/aniruddha27/f870032621ea9c0349a47969d85445cc/raw/b222027543cff2a09f5e3ced35e87df64716742b/nlp_ie_25.py) nlp_ie_25.py (https://gist.github.com/aniruddha27/f870032621ea9c0349a47969d85445cc#file-nlp_ie_25-py) hosted with ❤ by GitHub (https://github.com)

All we have to do is call this function whenever we encounter a noun in our phrase:

```
# rule 3 function
def rule3_mod(text):

    doc = nlp(text)

    sent = []

    for token in doc:

        if token.pos_=='ADP':

            phrase = ''
            if token.head.pos_=='NOUN':

                # appended rule
                append = rule0(text, token.head.i)
                if len(append)!=0:
                    phrase += append
                else:
                    phrase += token.head.text
                phrase += ' '+token.text

                for right_tok in token.rights:
                    if (right_tok.pos_ in ['NOUN','PROPN']):

                        right_phrase = ''
                        # appended rule
```

```
                        append = rule0(text, right_tok.i)
                        if len(append)!=0:
                            right_phrase += ' '+append
                        else:
                            right_phrase += ' '+right_tok.text

                        phrase += right_phrase

                if len(phrase)>2:
                    sent.append(phrase)


    return sent
```

	Sent	Year	Noun1	Preposition	Noun2
0	Mr President, I offer you our congratulations...	1970	congratulations	on	[election]
1	Mr President, I offer you our congratulations...	1970	session	of	[General Assembly]
2	Your personal qualifications and your family ...	1970	dedication	to	[effort]
3	I should also like to express our appreciatio...	1970	appreciation	of	[services]
4	I should also like to express our appreciatio...	1970	services	of	[predecessor]
5	I would also repeat our admiration for U Than...	1970	admiration	for	[U Thant]
6	There are many developments in India which gi...	1970	developments	in	[India]
7	We have had a sizable increase in agricultura...	1970	increase	in	[production]
8	Our trade is also showing signs of improvement	1970	signs	of	[improvement]
9	India has once again demonstrated its faith i...	1970	faith	in	[democracy]
10	Alongside this there is a growing desire of t...	1970	desire	of	[man]
11	Alongside this there is a growing desire of t...	1970	distribution	of	[wealth]
12	twenty two days ago a great conference ended ...	1970	signing	of	[Charter]
13	The Conference of Lusaka owes much of its suc...	1970	people	of	[Zambia]
14	The final declarations and resolution s of th...	1970	declarations	of	[Conference]
15	They represent the consensus of 53 Members of...	1970	consensus	of	[Members]
16	They represent the consensus of 53 Members of...	1970	Members	of	[United Nations]
17	These are: international peace and security, ...	1970	solution	of	[problems]

This definitely has more information than before. For example, ‘impediments in economic development’ instead of ‘impediments in development’ and ‘greater transgressor of human rights’ rather than ‘transgressor of rights’.

Once again, combining rules has given us more power and flexibility to explore only those sentences in detail that have a meaningful extracted phrase.

End Notes

Information extraction is by no means an easy NLP task to perform. You need to spend time with the data to better understand its structure and what it has to offer.

In this article, we used theoretical knowledge and put it to practical use. We worked with a text dataset and tried to extract the information using traditional information extraction techniques.

We looked for key phrases and relationships in the text data to try and extract the information from the text. This type of approach requires a combination of computer and human effort to extract relevant information.

Going forward, you can explore the following courses to expand your knowledge in the field of NLP:

- Natural Language Processing (NLP) Using Python (https://courses.analyticsvidhya.com/courses/natural-language-processing-nlp?utm_source=blog&utm_medium=nlp-project-information-extraction)
- A Comprehensive Learning Path to Understand and Master NLP in 2020 (https://www.analyticsvidhya.com/blog/2020/01/learning-path-nlp-2020/?utm_source=blog&utm_medium=nlp-project-information-extraction)

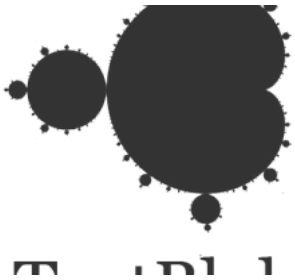
You can also read this article on our Mobile APP 

(//play.google.com/store/apps/details?id=com.analyticsvidhya.android&utm_source=blog_article&utm_campaign=blog&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1) 

(https://apps.apple.com/us/app/analytics-vidhya/id1470025572)

Related Articles

(https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/)
A Step-by-Step NLP Guide to Learn ELMo for Extracting Features from Text (https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/)



(https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/)
Natural Language Processing for Beginners: Using TextBlob (https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/)

(https://www.analyticsvidhya.com/blog/2020/08/people-to-follow-in-the-field-of-natural-language-processing-nlp/)
People to Follow in the field of Natural Language Processing (NLP) (https://www.analyticsvidhya.com/blog/2020/08/people-to-follow-in-the-field-of-natural-language-processing-nlp/)

TAGS : DEPENDENCY TREES (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/DEPENDENCY-TREES/), INFORMATION EXTRACTION (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/INFORMATION-EXTRACTION/), SPACY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/SPACY/)

NEXT ARTICLE

4 Simple Ways to Split a Decision Tree in Machine Learning

(https://www.analyticsvidhya.com/blog/2020/06/4-ways-split-decision-tree/)

•••

PREVIOUS ARTICLE

Hugging Face Releases New NLP 'Tokenizers' Library Version (v0.8.0)

(https://www.analyticsvidhya.com/blog/2020/06/hugging-face-tokenizers-nlp-library/)

(https://www.analyticsvidhya.com/blog/author/aniruddha/)

Aniruddha Bhandari (Https://Www.Analyticsvidhya.Com/Blog/Author/Aniruddha/)

I am on a journey to becoming a data scientist. I love to unravel trends in data, visualize it and predict the future with ML algorithms! But the most satisfying part of this journey is sharing my learnings, from the challenges that I face, with the community to make the world a better place!

6 COMMENTS

NAVEEN

Reply

June 29, 2020 at 7:47 pm (https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/#comment-162083)

Excellent article!

APPALANAIDU CHINTA **Reply**

June 29, 2020 at 11:21 pm (https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/#comment-162089)

Very Useful Information…!

RAYMUND DILAN (HTTP://DMMMSU.EDU.PH) **Reply**

July 6, 2020 at 7:43 am (https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/#comment-162163)

Very informative. I will share this link to my students. Thank you.

ANIRUDDHA BHANDARI **Reply**

July 6, 2020 at 6:17 pm (https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/#comment-162170)

Thanks for sharing!

JAGAN **Reply**

July 16, 2020 at 8:11 pm (https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/#comment-162324)

Nice writeup Anniruddha. Thanks for sharing.

HUNAIDKHAN PATHAN **Reply**

July 17, 2020 at 9:35 am (https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/#comment-162342)

What an amazing and detailed article Aniruddha , kudos. Haven’t seen any article with such detail on NLP Spacy. Great work

---

LEAVE A REPLY

Your email address will not be published.

Comment

Name (required)

Email (required)

Website

Notify me of new posts by email.

SUBMIT COMMENT

(https://courses.analyticsvidhya.com/courses/data-science-hacks-tips-and-tricks?utm_source=hacksandtipsbanner&utm_medium=blog)

POPULAR POSTS

45 Questions to test a data scientist on basics of Deep Learning (along with solution) (https://www.analyticsvidhya.com/blog/2017/01/must-know-questions-deep-learning/)

40 Questions to test a data scientist on Machine Learning [Solution: SkillPower – Machine Learning, DataFest 2017] (https://www.analyticsvidhya.com/blog/2017/04/40-questions-test-data-scientist-machine-learning-solution-skillpower-machine-learning-datafest-2017/)

10 Powerful YouTube Channels for Data Science Aspirants! (https://www.analyticsvidhya.com/blog/2020/08/10-powerful-youtube-channels-for-data-science-aspirants/)

Commonly used Machine Learning Algorithms (with Python and R Codes) (https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/)

6 Top Tools for Analytics and Business Intelligence in 2020 (https://www.analyticsvidhya.com/blog/2020/08/6-top-tools-for-analytics-and-business-intelligence-in-2020/)

40 Questions to test a Data Scientist on Clustering Techniques (Skill test Solution) (https://www.analyticsvidhya.com/blog/2017/02/test-data-scientist-clustering/)

30 Questions to test a data scientist on Linear Regression [Solution: Skilltest – Linear Regression] (https://www.analyticsvidhya.com/blog/2017/07/30-questions-to-test-a-data-scientist-on-linear-regression/)

30 Questions to test a Data Scientist on Deep Learning (Solution – Skill test, July 2017) (https://www.analyticsvidhya.com/blog/2017/08/skilltest-deep-learning/)

RECENT POSTS

Basic Concepts of Object-Oriented Programming in Python (https://www.analyticsvidhya.com/blog/2020/08/object-oriented-programming/)

AUGUST 31, 2020

A Simple Introduction to Sequence to Sequence Models (https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/)

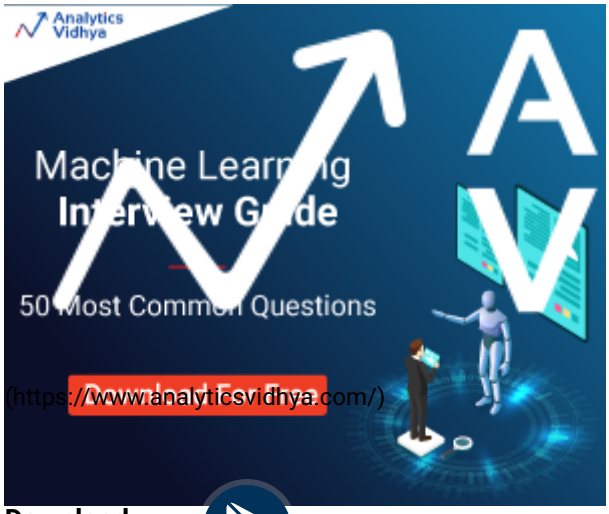
AUGUST 31, 2020

Plotting Decision Surface for Classification Machine Learning Algorithms (https://www.analyticsvidhya.com/blog/2020/08/plotting-decision-surface-for-classification-machine-learning-algorithms/)

AUGUST 31, 2020

Integrating Python in Power BI: Get the best of both worlds (https://www.analyticsvidhya.com/blog/2020/08/integrating-python-in-power-bi/)

AUGUST 31, 2020



(https://www.analytics vidhya.com/)

ıtt s:// w .an lyt sv dhya c m/back-channel/download-starter-kit.php?utm_so m(https://play.google.com/store/apps/details?id=com.analyticsvidhya.interviewguide&id=10)

Download App (https://apps.apple.com/us/app/analytics-vidhya/id1470025572)

Analytics Vidhya

About Us (https://www.analytics vidhya.com/about-me/)

Our Team (https://www.analytics vidhya.com/about-me/team/)

Careers (https://www.analytics vidhya.com/about-me/career-analytics-vidhya/)

Contact us (https://www.analytics vidhya.com/contact/)

Data Science

Blog (https://www.analytics vidhya.com/blog/)

Hackathon (https://datahack.analytics vidhya.com/)

Discussions (https://discuss.analytics vidhya.com/)

Apply Jobs (https://www.analytics vidhya.com/jobs/)

Companies

Post Jobs (https://www.analytics vidhya.com/corporate/)

Trainings (https://courses.analytics vidhya.com/)

Hiring Hackathons (https://datahack.analytics vidhya.com/)

Advertising (https://www.analytics vidhya.com/contact/)

Visit us

(https://www.linkedin.com/company/analytics-vidhya/) (https://www.facebook.com/analyticsvidhya) (https://twitter.com/analyticsvidhya) (https://www.youtube.com/channel/UCcIyqDteHtH4hg3o2343iObA)

Privacy Policy Terms of Use Refund Policy

(https://www.analytics vidhya.com/back-channel/download-pdf.php?utm_source=pdf&pid=66875&next=https://www.analytics vidhya.com/blog/2020/06/nlp-project-information-extraction/)

(https://id.analyticsvidhya.com/auth/login/?next=https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/)