RAMNARAIN RUIA AUTONOMOUS COLLEGE

DEPARMENT OF COMPUTER SCIENCE & IT (2020-21)

|  |  |  |
| --- | --- | --- |
| **SEAT NO**  **533** | **STUDENT NAME**  **Manideep Ramesh Choutapelly** |  |
| **CLASS: M.Sc CS** | **SEMESTER: I** | **COURSE CODE:**RP**SCSP103** |
|  |  |  |

COURSE NAME – Natural Language Processing

FACULTY IN-CHARGE - Ms. Edith Juni

|  |  |  |  |
| --- | --- | --- | --- |
| **INDEX** | | |  |
| **SR.**  **NO.** | **DATE** | **PPACTICAL NAME** | **SIGNATURE** |
| 1 | 14/02/21 | Preprocessing of text: Word Analysis (Tokenization, Filtration, word frequency, Stop Word Removal) |  |
| 2 | 17/02/21 | Preprocessing of text: Word Generation (Stemming , Lemmatiziation, Edit distance) |  |
| 3 | 25/02/21 | Morphological Analysis |  |
| 4 | 27/02/21 | N-gram model |  |
| 5 | 28/02/21 | Regular expression |  |
| 6 | 16/03/21 | POS tagging: HMM |  |
| 7 | 17/03/21 | POS tagging: Viterbi Decoding & Building POS Tagger |  |
| 8 | 18/03/21 | Name Entity Extraction & Chunking |  |

# Practical #1 - Preprocessing of text: Word Analysis (Tokenization, Filtration, word frequency, Stop Word Removal)

> Install NLTK

|  |
| --- |
| !pip install nltk |

> Download ‘punkt’ corpus

|  |
| --- |
| import nltk nltk.download('punkt') |

>Tokenize

|  |
| --- |
| from nltk.tokenize import sent\_tokenize text="""Hello Mr. Smith, how are you doing today? The weather is great, and city is awesome. The sky is pinkish-blue. You shouldn't eat cardboard""" tokenized\_text=sent\_tokenize(text) tokenized\_text |

Output:

|  |
| --- |
| ['Hello Mr. Smith, how are you doing today?',  'The weather is great, and city is awesome.',  'The sky is pinkish-blue.',...] |

> Word Tokenize

|  |
| --- |
| from nltk.tokenize import word\_tokenize tokenized\_word=word\_tokenize(text) tokenized\_word |

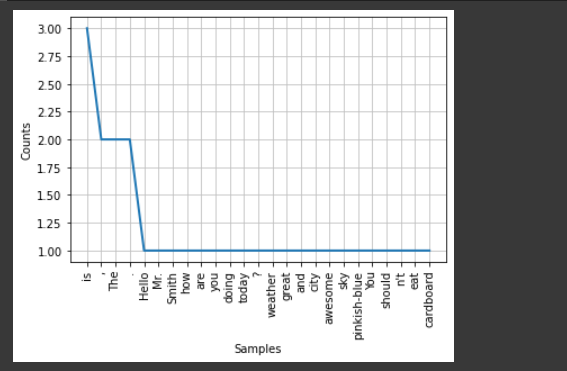
|  |
| --- |
|  |

Output

|  |
| --- |
| ['Hello',  'Mr.',  'Smith',  ',',...]  'how',  'are',  'you',  'doing',] |

> Frequency Distribution

|  |
| --- |
| from nltk.probability import FreqDist fdist = FreqDist(tokenized\_word) print(fdist) fdist.most\_common(2) [('is', 3), (',', 2)]  import matplotlib.pyplot as plt fdist.plot(30,cumulative=False) plt.show() |



> Download Stop words and find.

nltk.download('stopwords')

from nltk.corpus import stopwords

stop\_words=set(stopwords.words("english"))

stop\_words

Output

|  |
| --- |
| {'a',  'about',  'above',  'after',  'again',  'against',  'ain',  'all',  'am',....} |

# Practical #2 - Preprocessing of text: Word Generation (Stemming , Lemmatiziation, Edit distance)

> Download ‘punkt’ corpus and tokenize words/sentence

|  |
| --- |
| import nltk nltk.download('punkt') from nltk.tokenize import word\_tokenize tokenized\_word=word\_tokenize(text) print(tokenized\_word)  from nltk.tokenize import sent\_tokenize text="""Hello Mr. Smith, how are you doing today? The weather is great, and city is awesome. The sky is pinkish-blue. You shouldn't eat cardboard""" tokenized\_text=sent\_tokenize(text) print(tokenized\_text) |

> Output

|  |
| --- |
| ['Hello Mr. Smith, how are you doing today?', 'The weather is great, and city is awesome.', 'The sky is pinkish-blue.', "You shouldn't eat cardboard"]  ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.', 'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat', 'cardboard'] |

> Stemming

|  |
| --- |
| from nltk.stem import PorterStemmer from nltk.tokenize import sent\_tokenize, word\_tokenize  ps = PorterStemmer()  stemmed\_words=[] for w in tokenized\_word:  stemmed\_words.append(ps.stem(w))  print("Filtered Sentence:",tokenized\_word) print("Stemmed Sentence:",stemmed\_words) |

Output

|  |
| --- |
| Filtered Sentence: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.', 'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat', 'cardboard']  Stemmed Sentence: ['hello', 'mr.', 'smith', ',', 'how', 'are', 'you', 'do', 'today', '?', 'the', 'weather', 'is', 'great', ',', 'and', 'citi', 'is', 'awesom', '.', 'the', 'sky', 'is', 'pinkish-bl...] |

> Lemmatization

nltk.download('wordnet')

#Lexicon Normalization

#performing stemming and Lemmatization

from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer

stem = PorterStemmer()

word = "flagged"

print("Lemmatized Word:",lem.lemmatize(word,"v"))

print("Stemmed Word:",stem.stem(word))

Output

|  |
| --- |
| [nltk\_data] Downloading package wordnet to /root/nltk\_data... [nltk\_data] Unzipping corpora/wordnet.zip. Lemmatized Word: flag Stemmed Word: flag |

> Stemmer (porter stemmer)

|  |
| --- |
| import nltk from nltk.stem.porter import PorterStemmer porter\_stemmer = PorterStemmer() text = "walk walked walking cries cry" tokenization = nltk.word\_tokenize(text) for w in tokenization:  print("Stemming for {} is {}".format(w,porter\_stemmer.stem(w))) |

Output

|  |
| --- |
| Stemming for walk is walk Stemming for walked is walk Stemming for walking is walk Stemming for cries is cri Stemming for cry is cri |

> Wordnet lemmatizer

|  |
| --- |
| import nltk from nltk.stem import WordNetLemmatizer wordnet\_lemmatizer = WordNetLemmatizer() text = "walk walked walking cries cry" tokenization = nltk.word\_tokenize(text) for w in tokenization:  print("Lemma for {} is {}".format(w, wordnet\_lemmatizer.lemmatize(w))) |

Output

|  |
| --- |
| Lemma for walk is walk Lemma for walked is walked Lemma for walking is walking Lemma for cries is cry Lemma for cry is cry |

> Edit Distance

|  |
| --- |
| def editDistDP(str1, str2, m, n):  # Create a table to store results of subproblems  dp = [[0 for x in range(n + 1)] for x in range(m + 1)]   # Fill d[][] in bottom up manner  for i in range(m + 1):  for j in range(n + 1):   # If first string is empty, only option is to  # insert all characters of second string  if i == 0:  dp[i][j] = j # Min. operations = j   # If second string is empty, only option is to  # remove all characters of second string  elif j == 0:  dp[i][j] = i # Min. operations = i   # If last characters are same, ignore last char  # and recur for remaining string  elif str1[i-1] == str2[j-1]:  dp[i][j] = dp[i-1][j-1]   # If last character are different, consider all  # possibilities and find minimum  else:  dp[i][j] = 1 + min(dp[i][j-1], # Insert  dp[i-1][j], # Remove  dp[i-1][j-1]) # Replace   return dp[m][n] |

|  |
| --- |
| str2 = "PERMANENCY" str1 = "CURRENCY"  print(editDistDP(str1, str2, len(str1), len(str2))) |

Output

|  |
| --- |
| 5 |

# Practical #3 - Morphological Analysis

> Install dependencies of polyglot i.e. morfessor, pycld2, pyicu

!pip install morfessor

!pip install pycld2

!pip install pyicu

!pip install polyglot

> Download english corpus

|  |
| --- |
| %%shell polyglot download morph2.en morph2.ar |

> Find morphemes of words and tokenizing using Morphological Analysis

|  |
| --- |
| from polyglot.text import Text, Word words = ["preprocessing", "processor", "invaluable", "thankful", "crossed"] for w in words:  w = Word(w, language="en")  print("{:<20}{}".format(w, w.morphemes)) |

Output

|  |
| --- |
| preprocessing ['pre', 'process', 'ing'] processor ['process', 'or'] invaluable ['in', 'valuable'] thankful ['thank', 'ful'] crossed ['cross', 'ed'] |

Splitting string by morphological analysis'

|  |
| --- |
| blob = "Wewillmeettoday." text = Text(blob) text.language = "en" text.morphemes |

Output

|  |
| --- |
| WordList(['We', 'will', 'meet', 'to', 'day', '.']) |

> Using POS Tagging with PolyGlot

|  |
| --- |
| from polyglot.downloader import downloader print(downloader.supported\_languages\_table("pos2")) |

Output

|  |
| --- |
| 1. Italian 2. French 3. Spanish; Castilian   4. Bulgarian 5. Slovene 6. Irish   7. Finnish 8. Dutch 9. Swedish   10. Danish 11. Portuguese 12. English   13. German 14. Indonesian 15. Czech   16. Hungarian |

> Download English Pos corpus

|  |
| --- |
| %%shell polyglot download embeddings2.en pos2.en |

> POS TAGGING with Polygot (optional)

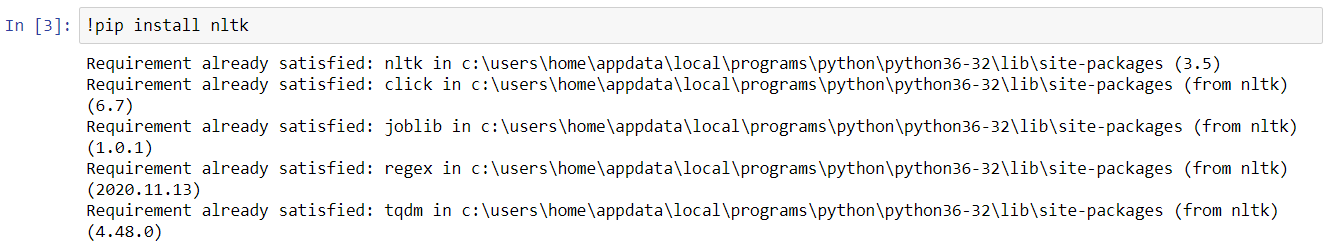
|  |
| --- |
| from polyglot.text import Text  blob = """We will meet at eight o'clock on Thursday morning.""" text = Text(blob) text.pos\_tags |

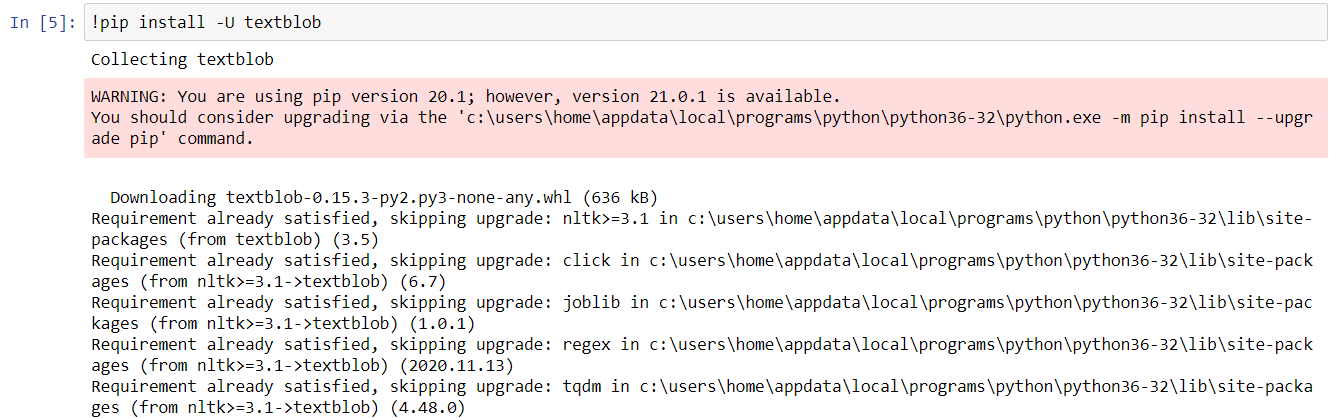
Output

|  |
| --- |
| [('We', 'PRON'),  ('will', 'AUX'),  ('meet', 'VERB'),  ('at', 'ADP'),  ('eight', 'NUM'),  ("o'clock", 'NOUN'),  ('on', 'ADP'),  ('Thursday', 'PROPN'),  ('morning', 'NOUN'),  ('.', 'PUNCT')] |

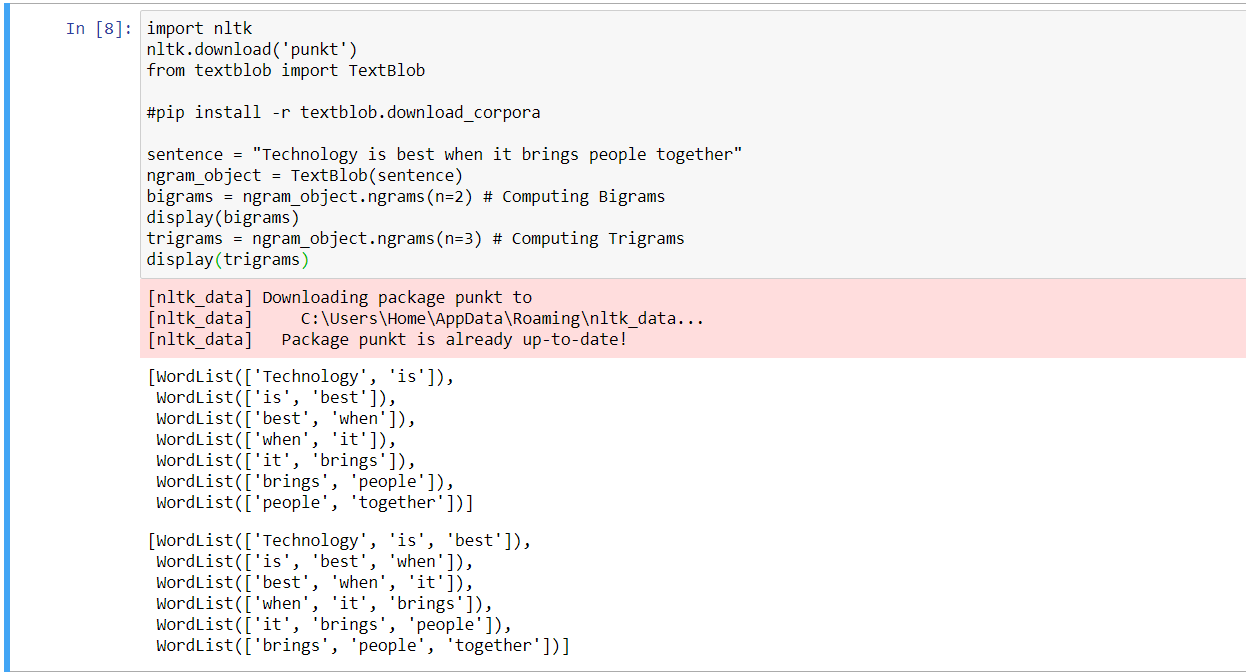
# Practical #4 - N-gram model

1. Install Dependencies (NLTK and textblob)





1. N-Grams



3)







# Practical #5 - Regular expression

> Basic REGEX

|  |
| --- |
| import re  regex = re.compile('\s+') text = """101 COM Computers 205 MAT Mathematics 189 ENG English"""  re.split('\s+', text) regex.split(text) print(text) regex\_num = re.compile('\d+') regex\_num.findall(text) |

> Output

|  |
| --- |
| 101 COM Computers 205 MAT Mathematics 189 ENG English ['101', '205', '189'] |

|  |
| --- |
| text2 = """COM Computers 205 MAT Mathematics 189""" regex\_num = re.compile('\d+')   s = regex\_num.search(text2)  print('Starting Position: ', s.start()) print('Ending Position: ', s.end()) print(text2[s.start():s.end()]) |

Output:

|  |
| --- |
| Starting Position: 17 Ending Position: 20 205 |

|  |
| --- |
| text = """101 COM \t Computers 205 MAT \t Mathematics 189 ENG \t English"""  print(text) regex = re.compile('\s+') print(regex.sub(' ', text)) |

Output

|  |
| --- |
| 101 COM Computers 205 MAT Mathematics 189 ENG English 101 COM Computers 205 MAT Mathematics 189 ENG English |

> Substitution

|  |
| --- |
| print(re.sub('\s+', ' ', text)) # get rid of all extra spaces except newline regex = re.compile('((?!\n)\s+)') print(regex.sub(' ', text)) text = """101 COM Computers 205 MAT Mathematics 189 ENG English"""  display(re.findall('[0-9]+', text)) display(re.findall('[A-Z]{3}', text)) display(re.findall('[A-Za-z]{4,}', text)) |

Output

|  |
| --- |
| 101 COM Computers 205 MAT Mathematics 189 ENG English 101 COM Computers 205 MAT Mathematics 189 ENG English  ['101', '205', '189'] ['COM', 'MAT', 'ENG'] ['Computers', 'Mathematics', 'English'] |

> define the course text pattern groups and extract

|  |
| --- |
| course\_pattern = '([0-9]+)\s\*([A-Z]{3})\s\*([A-Za-z]{4,})' display(re.findall(course\_pattern, text)) text = "< body>Regex Greedy Matching Example < /body>" display(re.findall('<.\*>', text)) display(re.findall('<.\*?>', text)) display(re.search('<.\*?>', text).group()) |

> Output

[]

['< body>Regex Greedy Matching Example < /body>']

['< body>', '< /body>']

'< body>

|  |
| --- |
| text = 'machinelearningplus.com' print(re.findall('.', text)) # . Any character except for a new line print(re.findall('...', text)) text = 'machinelearningplus.com' print(re.findall('\.', text)) # matches a period print(re.findall('[^\.]', text)) text = '01, Jan 2015' print(re.findall('\d+', text)) text = '01, Jan 2015' print(re.findall('\D+', text))  text = '01, Jan 2015' print(re.findall('\w+', text))  text = '01, Jan 2015' print(re.findall('\W+', text))  text = '01, Jan 2015' print(re.findall('[a-zA-Z]+', text)) |

Output

|  |
| --- |
| ['m', 'a', 'c', 'h', 'i', 'n', 'e', 'l', 'e', 'a', 'r', 'n', 'i', 'n', 'g', 'p', 'l', 'u', 's', '.', 'c', 'o', 'm'] ['mac', 'hin', 'ele', 'arn', 'ing', 'plu', 's.c'] ['.'] ['m', 'a', 'c', 'h', 'i', 'n', 'e', 'l', 'e', 'a', 'r', 'n', 'i', 'n', 'g', 'p', 'l', 'u', 's', 'c', 'o', 'm'] ['01', '2015'] [', Jan '] ['01', 'Jan', '2015'] [', ', ' '] ['Jan'] |

text = '01, Jan 2015'

print(re.findall('\d{4}', text)) # {n} Matches repeat n times.

print(re.findall('\d{2,4}', text))

print(re.findall(r'Co+l', 'So Cooool'))

print(re.findall(r'Pi\*lani', 'Pilani'))

print(re.findall(r'colou?r', 'color'))

re.findall(r'\btoy\b', 'play toy broke toys')

Output:

|  |
| --- |
| ['2015'] ['01', '2015'] ['Cooool'] ['Pilani'] ['color'] ['toy'] |

|  |
| --- |
| emails = """zuck26@facebook.com page33@google.com jeff42@amazon.com""" pattern = r'(\w+)@([A-Z0-9]+)\.([A-Z]{2,4})' re.findall(pattern, emails, flags=re.IGNORECASE) |

Output

|  |
| --- |
| [('zuck26', 'facebook', 'com'),  ('page33', 'google', 'com'),  ('jeff42', 'amazon', 'com')] |

> Retrieve all the words starting with ‘b’ or ‘B’ from the following text.

|  |
| --- |
| text = """Betty bought a bit of butter, But the butter was so bitter, So she bought some better butter, To make the bitter butter better."""  re.findall(r'\bB\w+', text, flags=re.IGNORECASE) |

Output

|  |
| --- |
| ['Betty',  'bought',  'bit',  'butter',  'But',  'butter',  'bitter',  'bought',  'better',  'butter',  'bitter',  'butter',  'Better'] |

> Split the following irregular sentence into words

|  |
| --- |
| sentence = """A, very very; irregular\_sentence""" " ".join(re.split('[;,\s\_]+', sentence)) |

Output

|  |
| --- |
| A very very irregular sentence |

|  |
| --- |
| tweet = '''Good advice! RT @TheNextWeb: What I would do differently if I was learning to code today http://t.co/lbwej0pxOd cc: @garybernhardt #rstats''' import re  def clean\_tweet(tweet):  tweet = re.sub('http\S+\s\*', '', tweet) # remove URLs  tweet = re.sub('RT|cc', '', tweet) # remove RT and cc  tweet = re.sub('#\S+', '', tweet) # remove hashtags  tweet = re.sub('@\S+', '', tweet) # remove mentions  tweet = re.sub('[%s]' % re.escape("""!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~"""), '', tweet) # remove punctuations  tweet = re.sub('\s+', ' ', tweet) # remove extra whitespace  return tweet  print(clean\_tweet(tweet)) |

Output

|  |
| --- |
| Good advice What I would do differently if I was learning to code today |

#Code to retrieve the HTML page:

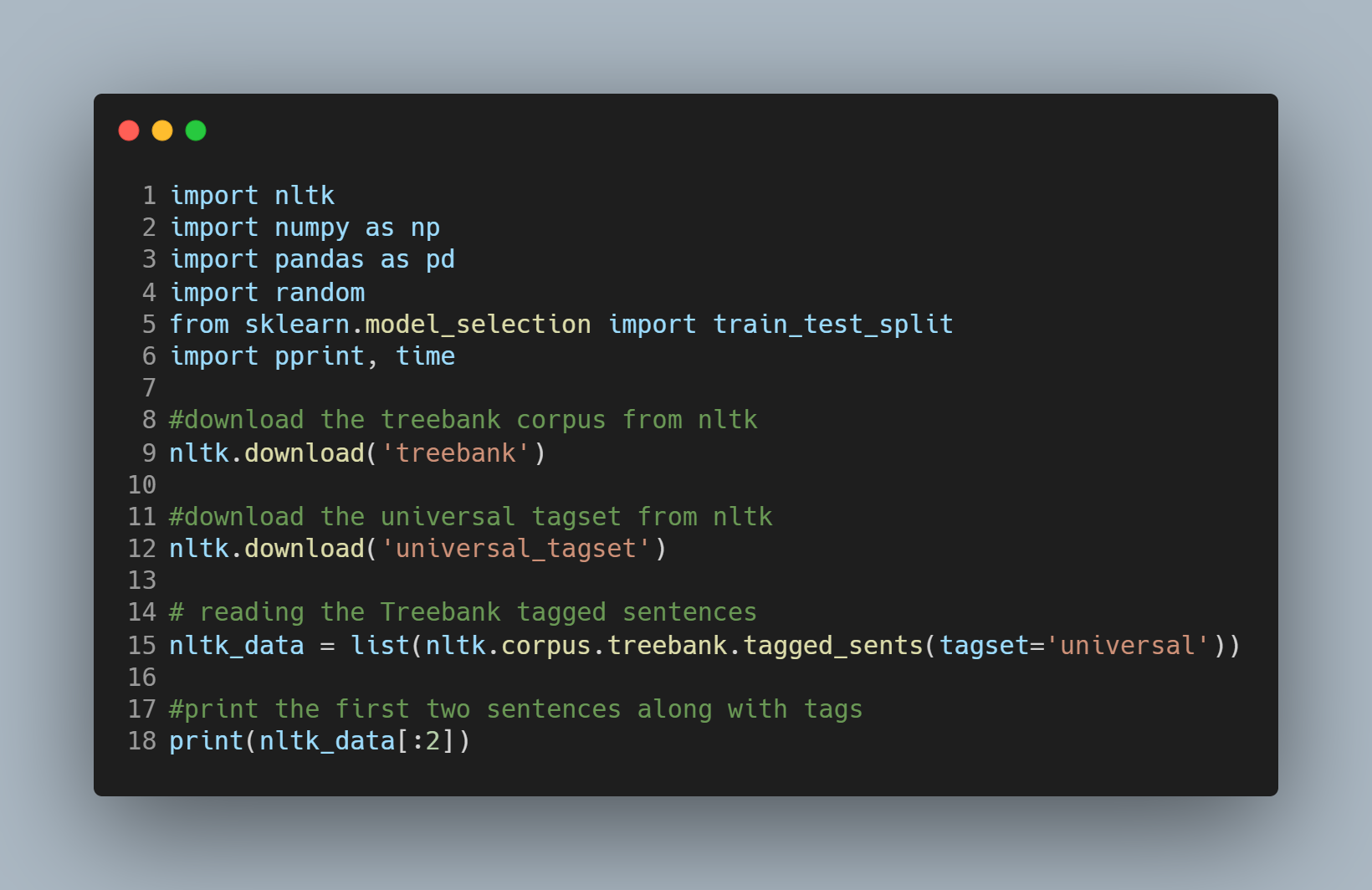
|  |
| --- |
| import requests r = requests.get("https://raw.githubusercontent.com/selva86/datasets/master/sample.html") display(r.text) # html text is contained here re.findall('<.\*?>(.\*)<\/.\*?>', r.text) |

Output

|  |
| --- |
| <HTML>\n<HEAD>\n<TITLE>Your Title Here</TITLE>\n</HEAD>\n\n<BODY>\n<HR>\n<a href="http://someurl.com">Link Name</a>\n<H1>This is a Header</H1>\n<H2>This is a Medium Header</H2>\n<P>This is a new paragraph! </P>\n<P>This is a another paragraph!</P>\n<B>This is a new sentence without a paragraph break, in bold italics.</B>\n<HR>\n</BODY>\n</HTML>\n' ['Your Title Here',  'Link Name',  'This is a Header',  'This is a Medium Header',  'This is a new paragraph! ',  'This is a another paragraph!',  'This is a new sentence without a paragraph break, in bold italics.'] |

# Practical #6 - HMM

> Global Imports



> print each word with its respective tag for first two sentences

|  |
| --- |
| for sent in nltk\_data[:2]:  for tuple in sent:  print(tuple) |



|  |
| --- |
| train\_set,test\_set =train\_test\_split(nltk\_data,train\_size=0.80,test\_size=0.20,random\_state = 101) |

> create list of train and test tagged words

|  |
| --- |
| train\_tagged\_words = [ tup for sent in train\_set for tup in sent ] test\_tagged\_words = [ tup for sent in test\_set for tup in sent ] print(len(train\_tagged\_words)) print(len(test\_tagged\_words)) |

Output:  
80310  
20366

> use set datatype to check how many unique tags are present in training data

|  |
| --- |
| tags = {tag for word,tag in train\_tagged\_words} print(len(tags)) print(tags)   # check total words in vocabulary vocab = {word for word,tag in train\_tagged\_words} |

> Compute Emission Probability

|  |
| --- |
| def word\_given\_tag(word, tag, train\_bag = train\_tagged\_words):  tag\_list = [pair for pair in train\_bag if pair[1]==tag]  count\_tag = len(tag\_list)#total number of times the passed tag occurred in train\_bag  w\_given\_tag\_list = [pair[0] for pair in tag\_list if pair[0]==word]  count\_w\_given\_tag = len(w\_given\_tag\_list)      return (count\_w\_given\_tag, count\_tag) |

> Compute Transition Probability

|  |
| --- |
| def t2\_given\_t1(t2, t1, train\_bag = train\_tagged\_words):  tags = [pair[1] for pair in train\_bag]  count\_t1 = len([t for t in tags if t==t1])  count\_t2\_t1 = 0  for index in range(len(tags)-1):  if tags[index]==t1 and tags[index+1] == t2:  count\_t2\_t1 += 1  return (count\_t2\_t1, count\_t1) |

|  |
| --- |
| tags\_matrix = np.zeros((len(tags), len(tags)), dtype='float32') for i, t1 in enumerate(list(tags)):  for j, t2 in enumerate(list(tags)):   tags\_matrix[i, j] = t2\_given\_t1(t2, t1)[0]/t2\_given\_t1(t2, t1)[1]   print(tags\_matrix) |

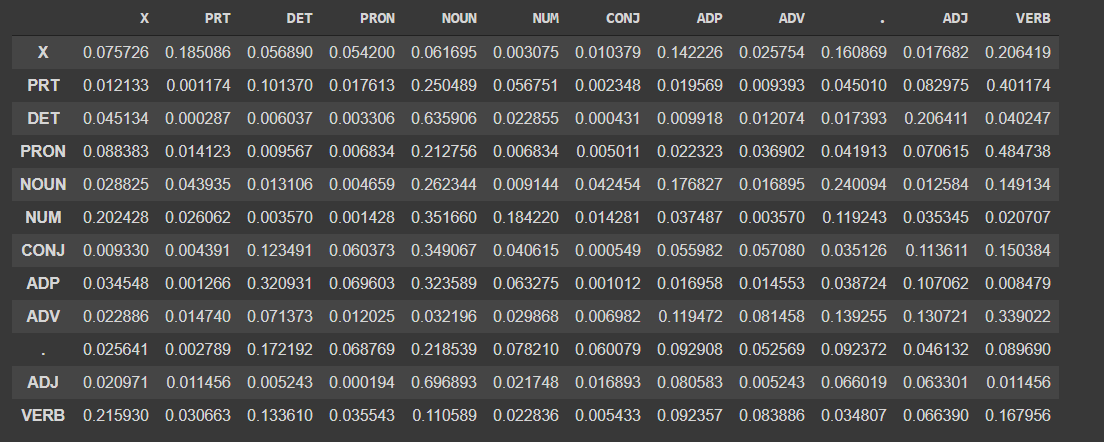
Output:

|  |
| --- |
| [[7.57255405e-02 1.85085520e-01 5.68902567e-02 5.41995019e-02  6.16951771e-02 3.07514891e-03 1.03786280e-02 1.42225638e-01  2.57543717e-02 1.60868734e-01 1.76821072e-02 2.06419379e-01]  [1.21330721e-02 1.17416831e-03 1.01369865e-01 1.76125243e-02  2.50489235e-01 5.67514673e-02 2.34833662e-03 1.95694715e-02  9.39334650e-03 4.50097844e-02 8.29745606e-02 4.01174158e-01]  [4.51343954e-02 2.87480245e-04 6.03708485e-03 3.30602261e-03  6.35906279e-01 2.28546783e-02 4.31220367e-04 9.91806854e-03  1.20741697e-02 1.73925534e-02 2.06410810e-01 4.02472317e-02]  [8.83826911e-02 1.41230067e-02 9.56719834e-03 6.83371304e-03  2.12756261e-01 6.83371304e-03 5.01138950e-03 2.23234631e-02  3.69020514e-02 4.19134386e-02 7.06150308e-02 4.84738052e-01]  [2.88252197e-02 4.39345129e-02 1.31063312e-02 4.65906132e-03  2.62344331e-01 9.14395228e-03 4.24540639e-02 1.76826611e-01  1.68945398e-02 2.40094051e-01 1.25838192e-02 1.49133503e-01]  [2.02427700e-01 2.60621198e-02 3.57015361e-03 1.42806140e-03  3.51660132e-01 1.84219927e-01 1.42806144e-02 3.74866128e-02  3.57015361e-03 1.19243130e-01 3.53445187e-02 2.07068902e-02]  [9.33040585e-03 4.39077942e-03 1.23490669e-01 6.03732169e-02  3.49066973e-01 4.06147093e-02 5.48847427e-04 5.59824370e-02  5.70801310e-02 3.51262353e-02 1.13611415e-01 1.50384188e-01]  [3.45482156e-02 1.26550242e-03 3.20931405e-01 6.96026310e-02  3.23588967e-01 6.32751212e-02 1.01240189e-03 1.69577319e-02  1.45532778e-02 3.87243740e-02 1.07061505e-01 8.47886596e-03]  [2.28859577e-02 1.47401085e-02 7.13731572e-02 1.20248254e-02  3.21955010e-02 2.98681147e-02 6.98215654e-03 1.19472459e-01  8.14584941e-02 1.39255241e-01 1.30721495e-01 3.39022487e-01]  [2.56410260e-02 2.78940029e-03 1.72191828e-01 6.87694475e-02  2.18538776e-01 7.82104954e-02 6.00793920e-02 9.29084867e-02  5.25694676e-02 9.23720598e-02 4.61323895e-02 8.96899477e-02]  [2.09708735e-02 1.14563107e-02 5.24271838e-03 1.94174761e-04  6.96893215e-01 2.17475723e-02 1.68932043e-02 8.05825219e-02  5.24271838e-03 6.60194159e-02 6.33009672e-02 1.14563107e-02]  [2.15930015e-01 3.06629837e-02 1.33609578e-01 3.55432779e-02  1.10589318e-01 2.28360966e-02 5.43278083e-03 9.23572779e-02  8.38858187e-02 3.48066315e-02 6.63904250e-02 1.67955801e-01]] |

> convert the matrix to a df for better readability

|  |
| --- |
| tags\_df = pd.DataFrame(tags\_matrix, columns = list(tags), index=list(tags)) display(tags\_df) |

OUTPUT:



# Practical #7: Viterbi Decoding

> download brown corpus

|  |
| --- |
| import nltk nltk.download('brown') |

> Implementation

|  |
| --- |
| import nltk import sys from nltk.corpus import brown  # Estimating P(wi | ti) from corpus data using Maximum Likelihood Estimation (MLE): # P(wi | ti) = count(wi, ti) / count(ti) # # We add an artificial "start" tag at the beginning of each sentence, and # We add an artificial "end" tag at the end of each sentence. # So we start out with the brown tagged sentences, # add the two artificial tags, # and then make one long list of all the tag/word pairs.  brown\_tags\_words = [ ] for sent in brown.tagged\_sents():  # sent is a list of word/tag pairs  # add START/START at the beginning  brown\_tags\_words.append( ("START", "START") )  # then all the tag/word pairs for the word/tag pairs in the sentence.  # shorten tags to 2 characters each  brown\_tags\_words.extend([ (tag[:2], word) for (word, tag) in sent ])  # then END/END  brown\_tags\_words.append( ("END", "END") )  # conditional frequency distribution cfd\_tagwords = nltk.ConditionalFreqDist(brown\_tags\_words) # conditional probability distribution cpd\_tagwords = nltk.ConditionalProbDist(cfd\_tagwords, nltk.MLEProbDist) |

|  |
| --- |
| print("The probability of an adjective (JJ) being 'new' is", cpd\_tagwords["JJ"].prob("new")) print("The probability of a verb (VB) being 'duck' is", cpd\_tagwords["VB"].prob("duck")) |

|  |
| --- |
| # Estimating P(ti | t{i-1}) from corpus data using Maximum Likelihood Estimation (MLE): # P(ti | t{i-1}) = count(t{i-1}, ti) / count(t{i-1}) brown\_tags = [tag for (tag, word) in brown\_tags\_words ]  # make conditional frequency distribution: # count(t{i-1} ti) cfd\_tags= nltk.ConditionalFreqDist(nltk.bigrams(brown\_tags)) # make conditional probability distribution, using # maximum likelihood estimate: # P(ti | t{i-1}) cpd\_tags = nltk.ConditionalProbDist(cfd\_tags, nltk.MLEProbDist)  print("If we have just seen 'DT', the probability of 'NN' is", cpd\_tags["DT"].prob("NN")) print( "If we have just seen 'VB', the probability of 'JJ' is", cpd\_tags["VB"].prob("DT")) print( "If we have just seen 'VB', the probability of 'NN' is", cpd\_tags["VB"].prob("NN")) |

|  |
| --- |
| prob\_tagsequence = cpd\_tags["START"].prob("PP") \* cpd\_tagwords["PP"].prob("I") \* \  cpd\_tags["PP"].prob("VB") \* cpd\_tagwords["VB"].prob("want") \* \  cpd\_tags["VB"].prob("TO") \* cpd\_tagwords["TO"].prob("to") \* \  cpd\_tags["TO"].prob("VB") \* cpd\_tagwords["VB"].prob("race") \* \  cpd\_tags["VB"].prob("END")  print( "The probability of the tag sequence 'START PP VB TO VB END' for 'I want to race' is:", prob\_tagsequence)  prob\_tagsequence = cpd\_tags["START"].prob("PP") \* cpd\_tagwords["PP"].prob("I") \* \  cpd\_tags["PP"].prob("VB") \* cpd\_tagwords["VB"].prob("saw") \* \  cpd\_tags["VB"].prob("PP") \* cpd\_tagwords["PP"].prob("her") \* \  cpd\_tags["PP"].prob("NN") \* cpd\_tagwords["NN"].prob("duck") \* \  cpd\_tags["NN"].prob("END")  print( "The probability of the tag sequence 'START PP VB PP NN END' for 'I saw her duck' is:", prob\_tagsequence) |

|  |
| --- |
| prob\_tagsequence = cpd\_tags["START"].prob("PP") \* cpd\_tagwords["PP"].prob("I") \* \  cpd\_tags["PP"].prob("VB") \* cpd\_tagwords["VB"].prob("saw") \* \  cpd\_tags["VB"].prob("PP") \* cpd\_tagwords["PP"].prob("her") \* \  cpd\_tags["PP"].prob("VB") \* cpd\_tagwords["VB"].prob("duck") \* \  cpd\_tags["VB"].prob("END")  print( "The probability of the tag sequence 'START PP VB PP VB END' for 'I saw her duck' is:", prob\_tagsequence) |

|  |
| --- |
| > Output |

|  |
| --- |
| The probability of an adjective (JJ) being 'new' is 0.01472344917632025 The probability of a verb (VB) being 'duck' is 6.042713350943527e-05 If we have just seen 'DT', the probability of 'NN' is 0.5057722522030194 If we have just seen 'VB', the probability of 'JJ' is 0.016885067592065053 If we have just seen 'VB', the probability of 'NN' is 0.10970977711020183 The probability of the tag sequence 'START PP VB TO VB END' for 'I want to race' is: 1.0817766461150474e-14 The probability of the tag sequence 'START PP VB PP NN END' for 'I saw her duck' is: 3.372745049070759e-16 The probability of the tag sequence 'START PP VB PP VB END' for 'I saw her duck' is: 7.285965712199413e-16 |

> Viterbi: If we have a word sequence, what is the best tag sequence?

|  |
| --- |
| distinct\_tags = set(brown\_tags)  sentence = ["I", "want", "to", "race" ]  sentlen = len(sentence)  viterbi = [ ] in X  backpointer = [ ]  first\_viterbi = { }  first\_backpointer = { }  for tag in distinct\_tags:  # don't record anything for the START tag  if tag == "START": continue  first\_viterbi[ tag ] = cpd\_tags["START"].prob(tag) \* cpd\_tagwords[tag].prob( sentence[0] )  first\_backpointer[ tag ] = "START"  print(first\_viterbi)  print(first\_backpointer)    viterbi.append(first\_viterbi)  backpointer.append(first\_backpointer)  currbest = max(first\_viterbi.keys(), key = lambda tag: first\_viterbi[ tag ])  print( "Word", "'" + sentence[0] + "'", "current best two-tag sequence:", first\_backpointer[ currbest], currbest)  # print( "Word", "'" + sentence[0] + "'", "current best tag:", currbest)  for wordindex in range(1, len(sentence)):  this\_viterbi = { }  this\_backpointer = { }  prev\_viterbi = viterbi[-1]    for tag in distinct\_tags:  # don't record anything for the START tag  if tag == "START": continue  # if this tag is X and the current word is w, then  # find the previous tag Y such that  # the best tag sequence that ends in X  # actually ends in Y X  # that is, the Y that maximizes  # prev\_viterbi[ Y ] \* P(X | Y) \* P( w | X)  # The following command has the same notation  # that you saw in the sorted() command.  best\_previous = max(prev\_viterbi.keys(),  key = lambda prevtag: \  prev\_viterbi[ prevtag ] \* cpd\_tags[prevtag].prob(tag) \* cpd\_tagwords[tag].prob(sentence[wordindex]))  # Instead, we can also use the following longer code:  # best\_previous = None  # best\_prob = 0.0  # for prevtag in distinct\_tags:  # prob = prev\_viterbi[ prevtag ] \* cpd\_tags[prevtag].prob(tag) \* cpd\_tagwords[tag].prob(sentence[wordindex])  # if prob > best\_prob:  # best\_previous= prevtag  # best\_prob = prob  #  this\_viterbi[ tag ] = prev\_viterbi[ best\_previous] \* \  cpd\_tags[ best\_previous ].prob(tag) \* cpd\_tagwords[ tag].prob(sentence[wordindex])  this\_backpointer[ tag ] = best\_previous  currbest = max(this\_viterbi.keys(), key = lambda tag: this\_viterbi[ tag ])  print( "Word", "'" + sentence[ wordindex] + "'", "current best two-tag sequence:", this\_backpointer[ currbest], currbest)  # print( "Word", "'" + sentence[ wordindex] + "'", "current best tag:", currbest)  # done with all tags in this iteration  # so store the current viterbi step  viterbi.append(this\_viterbi)  backpointer.append(this\_backpointer)  # done with all words in the sentence.  # now find the probability of each tag  # to have "END" as the next tag,  # and use that to find the overall best sequence  prev\_viterbi = viterbi[-1]  best\_previous = max(prev\_viterbi.keys(),  key = lambda prevtag: prev\_viterbi[ prevtag ] \* cpd\_tags[prevtag].prob("END"))  prob\_tagsequence = prev\_viterbi[ best\_previous ] \* cpd\_tags[ best\_previous].prob("END")  # best tagsequence: we store this in reverse for now, will invert later  best\_tagsequence = [ "END", best\_previous ]  # invert the list of backpointers  backpointer.reverse()  current\_best\_tag = best\_previous  for bp in backpointer:  best\_tagsequence.append(bp[current\_best\_tag])  current\_best\_tag = bp[current\_best\_tag]  best\_tagsequence.reverse()  print( "The sentence was:", end = " ")  for w in sentence: print( w, end = " ")  print("\n")  print( "The best tag sequence is:", end = " ")  for t in best\_tagsequence: print (t, end = " ")  print("\n")  print( "The probability of the best tag sequence is:", prob\_tagsequence) |

|  |
| --- |
| > OUTPUT |

|  |
| --- |
| {':': 0.0, 'AB': 0.0, "''": 0.0, 'IN': 0.0, ',-': 0.0, '.': 0.0, 'NR': 0.0, 'JJ': 0.0, 'END': 0.0, 'DT': 0.0, '--': 0.0, 'WQ': 0.0, 'RN': 0.0, 'WR': 0.0, ':-': 0.0, 'FW': 0.0, 'BE': 0.0, 'EX': 0.0, 'RP': 0.0, '\*': 0.0, 'CC': 0.0, "'": 0.0, 'QL': 0.0, 'DO': 0.0, ',': 0.0, '(': 0.0, '``': 0.0, 'WD': 0.0, 'RB': 0.0, 'PP': 0.014930900689060006, 'OD': 0.0, 'HV': 0.0, 'AT': 0.0, 'AP': 0.0, 'PN': 0.0, 'NN': 1.0580313619573935e-06, 'NI': 3.3324520848931064e-07, 'NP': 1.7319067623793952e-06, 'CD': 0.0, '.-': 0.0, '\*-': 0.0, ')-': 0.0, ')': 0.0, '(-': 0.0, 'MD': 0.0, 'UH': 0.0, 'CS': 0.0, 'TO': 0.0, 'VB': 0.0, 'WP': 0.0} {':': 'START', 'AB': 'START', "''": 'START', 'IN': 'START', ',-': 'START', '.': 'START', 'NR': 'START', 'JJ': 'START', 'END': 'START', 'DT': 'START', '--': 'START', 'WQ': 'START', 'RN': 'START', 'WR': 'START', ':-': 'START', 'FW': 'START', 'BE': 'START', 'EX': 'START', 'RP': 'START', '\*': 'START', 'CC': 'START', "'": 'START', 'QL': 'START', 'DO': 'START', ',': 'START', '(': 'START', '``': 'START', 'WD': 'START', 'RB': 'START', 'PP': 'START', 'OD': 'START', 'HV': 'START', 'AT': 'START', 'AP': 'START', 'PN': 'START', 'NN': 'START', 'NI': 'START', 'NP': 'START', 'CD': 'START', '.-': 'START', '\*-': 'START', ')-': 'START', ')': 'START', '(-': 'START', 'MD': 'START', 'UH': 'START', 'CS': 'START', 'TO': 'START', 'VB': 'START', 'WP': 'START'} Word 'I' current best two-tag sequence: START PP Word 'want' current best two-tag sequence: PP VB Word 'to' current best two-tag sequence: VB TO Word 'race' current best two-tag sequence: IN NN The sentence was: I want to race   The best tag sequence is: START PP VB IN NN END   The probability of the best tag sequence is: 5.71772824864617e-14 |

# Practical #8: Building Chunker / Name Entity Extraction

> Imports

|  |
| --- |
| import nltk from nltk.tokenize import word\_tokenize from nltk.tag import pos\_tag |

|  |
| --- |
| ex = 'European authorities fined Google a record $5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter its practices' |

> Utility Function

|  |
| --- |
| def preprocess(sent):  sent = nltk.word\_tokenize(sent)  sent = nltk.pos\_tag(sent)  return sent |

> Download corpus and module

|  |
| --- |
| import nltk nltk.download('punkt') nltk.download('averaged\_perceptron\_tagger') |

> tokenize the sentence

|  |
| --- |
| sent = preprocess(ex) sent |

Output:

|  |
| --- |
| [('European', 'JJ'),  ('authorities', 'NNS'),  ('fined', 'VBD'),  ('Google', 'NNP'),  ('a', 'DT'),  ('record', 'NN'),  ('$', '$'),  ('5.1', 'CD'),  ('billion', 'CD'),  ('on', 'IN'),  ('Wednesday', 'NNP'),  ('for', 'IN'),  ('abusing', 'VBG'),  ('its', 'PRP$'),  ('power', 'NN'),  ('in', 'IN'),  ('the', 'DT'),  ('mobile', 'JJ'),  ('phone', 'NN'),  ('market', 'NN'),  ('and', 'CC'),  ('ordered', 'VBD'),  ('the', 'DT'),  ('company', 'NN'),  ('to', 'TO'),  ('alter', 'VB'),  ('its', 'PRP$'),  ('practices', 'NNS')] |

> Print parsed tokens

|  |
| --- |
| pattern = 'NP: {<DT>?<JJ>\*<NN>}' cp = nltk.RegexpParser(pattern) cs = cp.parse(sent) print(cs) |

Output:

|  |
| --- |
| (S  European/JJ  authorities/NNS  fined/VBD  Google/NNP  (NP a/DT record/NN)  $/$  5.1/CD  billion/CD  on/IN  Wednesday/NNP  for/IN  abusing/VBG  its/PRP$  (NP power/NN)  in/IN  (NP the/DT mobile/JJ phone/NN)  (NP market/NN)  and/CC  ordered/VBD  (NP the/DT company/NN)  to/TO  alter/VB  its/PRP$  practices/NNS) |

|  |
| --- |
| from nltk.chunk import conlltags2tree, tree2conlltags from pprint import pprint iob\_tagged = tree2conlltags(cs) pprint(iob\_tagged) |

Output:

|  |
| --- |
| [('European', 'JJ', 'O'),  ('authorities', 'NNS', 'O'),  ('fined', 'VBD', 'O'),  ('Google', 'NNP', 'O'),  ('a', 'DT', 'B-NP'),  ('record', 'NN', 'I-NP'),  ('$', '$', 'O'),  ('5.1', 'CD', 'O'),  ('billion', 'CD', 'O'),  ('on', 'IN', 'O'),  ('Wednesday', 'NNP', 'O'), …….. ] |

> Download corpus

|  |
| --- |
| from nltk import ne\_chunk import nltk nltk.download('maxent\_ne\_chunker') nltk.download('words') |

|  |
| --- |
| ne\_tree = ne\_chunk(pos\_tag(word\_tokenize(ex))) print(ne\_tree) |

|  |
| --- |
| (S  (GPE European/JJ)  authorities/NNS  fined/VBD  (PERSON Google/NNP)  a/DT  record/NN  $/$  5.1/CD  billion/CD  on/IN  Wednesday/NNP  for/IN  abusing/VBG  its/PRP$  power/NN |

> Spacy library

|  |
| --- |
| import spacy from spacy import displacy from collections import Counter import en\_core\_web\_sm nlp = en\_core\_web\_sm.load()   doc = nlp('European authorities fined Google a record $5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter its practices') pprint([(X.text, X.label\_) for X in doc.ents]) |

Output:

|  |
| --- |
| [('European', 'NORP'),  ('Google', 'ORG'),  ('$5.1 billion', 'MONEY'),  ('Wednesday', 'DATE')] |

> Web Scraping

|  |
| --- |
| from bs4 import BeautifulSoup import requests import re def url\_to\_string(url):  res = requests.get(url)  html = res.text  soup = BeautifulSoup(html, 'html5lib')  for script in soup(["script", "style", 'aside']):  script.extract()  return " ".join(re.split(r'[\n\t]+', soup.get\_text())) ny\_bb = url\_to\_string('https://www.nytimes.com/2018/08/13/us/politics/peter-strzok-fired-fbi.html?hp&action=click&pgtype=Homepage&clickSource=story-heading&module=first-column-region&region=top-news&WT.nav=top-news') article = nlp(ny\_bb) len(article.ents) labels = [x.label\_ for x in article.ents] Counter(labels) |

Output:

|  |
| --- |
| Counter({'CARDINAL': 3,  'DATE': 23,  'GPE': 9,  'LOC': 1,  'NORP': 2,  'ORDINAL': 1,  'ORG': 38,  'PERSON': 77}) |

> most common entities in the scrapped web article

|  |
| --- |
| items = [x.text for x in article.ents] Counter(items).most\_common(3) |

Output:

|  |
| --- |
| [('Strzok', 29), ('F.B.I.', 19), ('Trump', 13)] |

|  |
| --- |
| sentences = [x for x in article.sents] print(sentences[20]) |

Output

|  |
| --- |
| spokeswoman for the F.B.I. did not respond to a message seeking comment about why Mr. Strzok was dismissed rather than demoted. |

> Display it on Jupyter NB

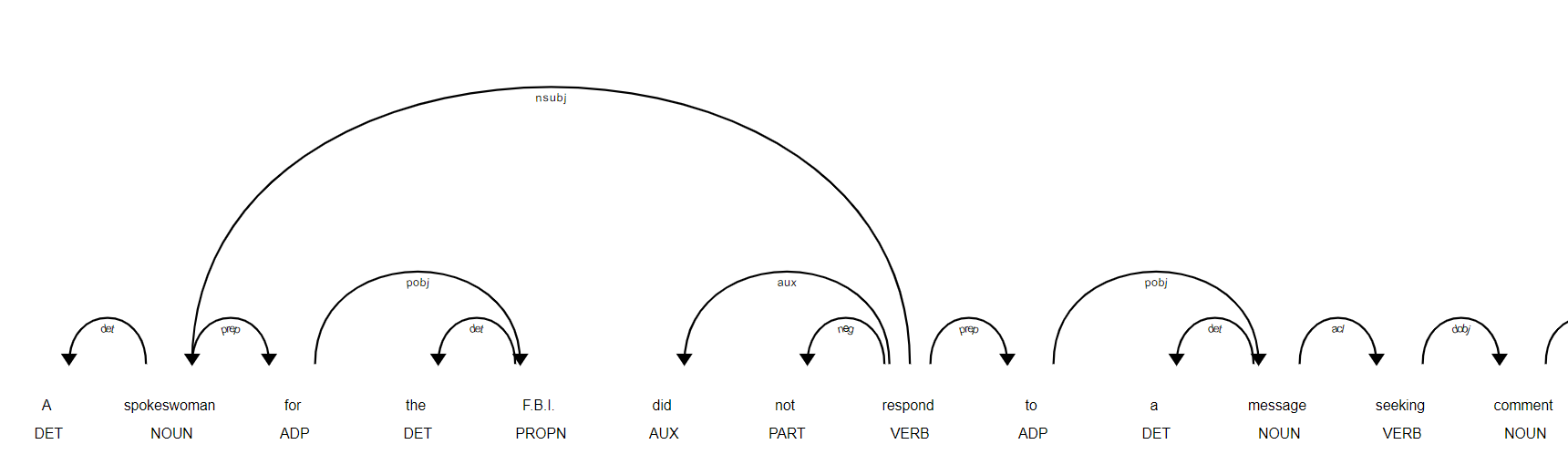
|  |
| --- |
| displacy.render(nlp(str(sentences[20])), jupyter=True, style='ent') |

Output



|  |
| --- |
| displacy.render(nlp(str(sentences[20])), style='dep', jupyter = True, options = {'distance': 120}) |

Output:



|  |
| --- |
| [(x.orth\_,x.pos\_, x.lemma\_) for x in [y   for y  in nlp(str(sentences[20]))   if not y.is\_stop and y.pos\_ != 'PUNCT']]  Output:  [('spokeswoman', 'NOUN', 'spokeswoman'),  ('F.B.I.', 'PROPN', 'F.B.I.'),  ('respond', 'VERB', 'respond'),  ('message', 'NOUN', 'message'),  ('seeking', 'VERB', 'seek'),  ('comment', 'NOUN', 'comment'),  ('Mr.', 'PROPN', 'Mr.'),  ('Strzok', 'PROPN', 'Strzok'),  ('dismissed', 'VERB', 'dismiss'),  ('demoted', 'VERB', 'demote')] |

|  |
| --- |
| dict([(str(x), x.label\_) for x in nlp(str(sentences[20])).ents])  Output: {'F.B.I.': 'ORG', 'Strzok': 'PERSON'}    print([(x, x.ent\_iob\_, x.ent\_type\_) for x in sentences[20]])  Output: [(A, 'O', ''), (spokeswoman, 'O', ''), (for, 'O', ''), (the, 'O', ''), (F.B.I., 'B', 'ORG'), (did, 'O', ''), (not, 'O', ''), (respond, 'O', ''), (to, 'O', '') |