##POS Tagging

NLTK POS Tagger

A very popular Penn Treebank : lists all the possible parts of speech i.e. tags (categories that a word can possibly belong to). It can vary for every word since it depends on the local context

UD : Universal Dependency Treebank

NLTK: contains the following taggers CRFTagger Perceptron tagger BrillTagger (rule-based approach) Stanfod POS Tagger

DT () NN

```
import nltk
nltk.download('punkt')

=)

import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
sentence = "He was being opposed by her without any reason.\
 A plan is being prepared by charles for next project"
for sent in sent_tokenize(sentence):
    wordtokens = word_tokenize(sent)
    print(nltk.pos_tag(wordtokens),end='\n\n')
```

Spacy POS Tagger

```
import en_core_web_sm
nlp = en_core_web_sm.load()

import spacy
nlp1 = spacy.load(nlp)
sentence = "He was being opposed by her without any reason.\
 A plan is being prepared by charles for next project"
for token in nlp1(sentence):
    print(f'{token.text:{10}} {token.tag_:>{10}}\
t{spacy.explain(token.tag_):<{50}} {token.pos_:>{5}}')
```

##Hidden Markov Models

```
!pip install nltk

import nltk
nltk.download('treebank')

nltk.download('universal_tagset')
```

D-> C

Generative model: class: pet/family-> essay/ data

```
import nltk
from sklearn.model_selection import train_test_split
tagged_sentences =
nltk.corpus.treebank.tagged_sents(tagset='universal')#loading corpus
traindataset , testdataset = train_test_split(tagged_sentences,
shuffle=True, test_size=0.2) #Splitting test and train dataset
#Test data size is 20% of whole tagged sentences
#Training HMMModel
HmmModel = nltk.HiddenMarkovModelTagger.train(traindataset)
#Correct labels from test dataset
correct_labels = [tag for sentences in testdataset for word, tag in
sentences] ##('He', 'PRP')
#predicting labels of testdataset
predicted_labels=[]
for sentences in testdataset:
    predicted_labels += [tag for _, tag in HmmModel.tag([word for
word, _ in sentences])]
#Classification report
from sklearn.metrics import classification_report
print (classification_report(correct_labels, predicted_labels))
```

1: getting the corpus for input -> tagged corpus 2: train: test split 3: training the model 4: getting the actual/ corrected labels 5: evaluating the model performance

Visualization of dependency:

Displacy Dependency Visualizer — https://explosion.ai/demos/displacy

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("He was being opposed by her without any reason")
spacy.displacy.serve(doc,style="dep")

#Saving this dependency image

from pathlib import Path
svg = spacy.displacy.render(doc, style="dep",jupyter=False)
output_path = Path("dependency_plot.svg") #path with file name
output_path.open("w", encoding="utf-8").write(svg)
```

spaCy custom pos pattern matcher

```
import spacy
from spacy.matcher import Matcher
nlp = spacy.load("en_core_web_sm") #loading spacy model

frstword = lambda x: x[0] #Func. to take 1st item in iterative item
# Example  [('August', 'NNP'), ('with', 'IN'), ('U.S.', 'NNP')] -->
```

```
#['August','with','U.S.']
joiner = lambda x: ' '.join(list(map(frstword,x)))
#joiner will call frstword and joins string like this -->
#'August with U.S.'
corpus = ""
for sentences in traindataset:
    corpus+=joiner(sentences)+'\n'
```

spaCy custom pos pattern matcher

```
import spacy
from spacy.matcher import Matcher
nlp = spacy.load("en_core_web_sm") #loading spacy model

[('August', 'NNP'), ('with', 'IN'), ('U.S.', 'NNP')]-
>['August',"with","U.S."]

['August',"with","U.S."]->August with U.S.

frstword = lambda x: x[0] #Func. to take 1st item in iterative item
# Example  [('August', 'NNP'), ('with', 'IN'), ('U.S.', 'NNP')] -->
#['August','with','U.S.']
joiner = lambda x: ' '.join(list(map(frstword,x)))
#joiner will call frstword and joins string like this -->
#'August with U.S.'
corpus = ""
for sentences in traindataset:
    corpus+=joiner(sentences)+'\n'

doc = nlp(corpus)
matcher = Matcher(doc.vocab)

matchlist = matcher(doc)

##Pre-processing the ouput
import pandas as pd
df = pd.DataFrame(matchlist)
df.drop_duplicates(subset=[1], keep='last', inplace=True)
df.drop_duplicates(subset=[2], keep='last', inplace=True)
for _,indx in df[[1,2]].iterrows():
    print(doc[indx[1]:indx[2]],end='\n\n')
```