Experiment 10

NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
SUBJECT	**NLP Lab**

AIM: To analyze the importance of context and size of training corpus in POS. Comparative study of accuracies of Regex tagger, Stochastic(CRF) tagger, Transformation tagger and use of backoff technique.

THEORY:

In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is identification of words as nouns, verbs, adjectives, adverbs, etc. Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic

Hidden Markov Model :

In the mid 1980s, researchers in Europe began to use Hidden Markov models (HMMs) to disambiguate parts of speech. HMMs involve counting cases, and making a table of the probabilities of certain sequences. For example, once you've seen an article such as 'the', perhaps the next word is a noun 40% of the time, an adjective 40%, and a number 20%. Knowing this, a program can decide that "can" in "the can" is far more likely to be a noun than a verb or a modal. The same method can of course be used to benefit from knowledge about the following words.

More advanced ("higher order") HMMs learn the probabilities not only of pairs, but triples or even larger sequences. So, for example, if you've just seen an article and a verb, the next item may very likely be a preposition, article, or noun, but much less likely another verb.

When several ambiguous words occur together, the possibilities multiply. However, it is easy to enumerate every combination and to assign a relative probability to each one, by multiplying together the probabilities of each choice in turn.

It is worth remembering, as Eugene Charniak points out in Statistical techniques for natural language parsing, that merely assigning the most common tag to each known word and the tag "proper noun" to all unknowns, will approach 90% accuracy because many words are unambiguous.

HMMs underlie the functioning of stochastic taggers and are used in various algorithms.

When we can not observe the states themselves but only the result of some probability function(observation) of the states we utilize HMM. HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states.

Emission probabilities :

In the above example, feelings (Happy or Grumpy) can be only observed. A person can observe that a person has an 80% chance to be Happy given that the climate at the particular point of observation(or rather day in this case) is Sunny. Similarly the 60% chance of a person being Grumpy given that the climate is Rainy. Here mentioned 80% and 60% are Emission probabilities since they deal with observations.

Transition probabilities :

When we consider the climates (hidden states) that influence the observations there are correlations between consecutive days being Sunny or alternate days being Rainy. There is 80% for the Sunny climate to be in successive days whereas 60% chance for consecutive days being Rainy. The probabilities that explain the transition to/from hidden states are Transition probabilities.

Conditional Random Field :

Conditional Random Fields is a class of discriminative models best suited to prediction tasks where contextual information or state of the neighbors affect the current prediction. CRFs find their applications in named entity recognition, part of speech tagging, gene prediction, noise reduction and object detection problems, to name a few.

Conditional random fields (CRFs) are a class of statistical modeling methods often applied in machine learning, where they are used for structured prediction. Whereas an ordinary classifier predicts a label for a single sample without regard to "neighboring" samples, a CRF can take context into account. Since it can consider context, therefore CRF can be used in Natural Language Processing. Hence, Parts of Speech tagging is also possible. It predicts the POS using the lexicons as the context.

How CRFs differ from Hidden Markov Models :

If only one neighbour is considered as a context, then it is called bigram. Similarly, two neighbours as the context is called trigram. In this experiment, the size of the training corpus and context were varied to know their importance.

From the previous sections, it must be obvious how Conditional Random Fields differ from Hidden Markov Models. Although both are used to model sequential data, they are different algorithms. Hidden Markov Models are generative, and give output by modeling the joint probability distribution. On the other hand, Conditional Random Fields are discriminative, and model the conditional probability distribution. CRFs don't rely on the independence assumption (that the labels are independent of each other), and avoid label bias. One way to look at it is that Hidden Markov Models are a very specific case of Conditional Random Fields, with constant transition probabilities used instead. HMMs are based on Naive Bayes, which we say can be derived from Logistic Regression, from which CRFs are derived.

Input:

- Text Corpus of sufficient length. For example movie reviews, newspaper articles, etc

Output:

- Analysis on the importance of context and size of training corpus in POS and answer of the above mentioned questions
- Compare accuracies of different taggers, Regex, CRF, Transformation, Backoff

IDE USED: Google Colab

LIBRARIES USED:

Nltk:

The Natural Language Toolkit (NLTK) is a Python package for natural language processing. NLTK requires Python 3.7, 3.8, 3.9 or 3.10. It can be installed as :

pip install nltk

CODE & OUTPUT:

```
import nltk
import re
from collections import OrderedDict
from nltk.probability import FreqDist
from nltk.tokenize import sent_tokenize,word_tokenize
nltk.download('treebank')

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

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

```
text='''Japan's feudal era was characterized by the emergence and dominance of a ruling class of warriors, the samurai.[32] In 1185, following the defeat of the Taira clan in the Geng
During the 16th century, Portuguese traders and Jesuit missionaries reached Japan for the first time, initiating direct commercial and cultural exchange between Japan and the West.[25

sen=sent_tokenize(text)
```

```
# TRAIN DATA MAKING
import csv
from nltk.corpus import treebank
test_data = treebank.tagged_sents()[3000:]
# print(test_data)
header = ['ID','WORD','POS_TAG']

a=0
with open('corpus.csv', 'w', encoding='UTF8') as f:
    writer = csv.writer(f)
    data=[]
    writer.writerow(header)
    for i in test_data:
      for j in i:
        writer.writerow([a,j[0],j[1]])
        a+=1
```

```
from nltk.tag import RegexpTagger
from nltk.corpus import treebank
```

```
# patterns = [(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),# cardinal numbers
#          (r'(The|the|A|a|An|an)$', 'AT'),   # articles
#          (r'.*able$', 'JJ'),                # adjectives
#          (r'.*ness$', 'NN'),                # nouns formed from adjectives
#          (r'.*ly$', 'RB'),                  # adverbs
#          (r'.*s$', 'NNS'),                  # plural nouns
#          (r'.*ing$', 'VBG'),                # gerunds
#          (r'.*ed$', 'VBD'),                 # past tense verbs
#          (r'.*', 'NN')                      # nouns (default)
#         ]


patterns = [(r'^\d+$', 'CD'),
            (r'.*ing$', 'VBG'),
            (r'.*ment$', 'NN'),
            (r'.*ful$', 'JJ')]
```

```
pos_tags=[]
for s in sen:
  pos_tags.append(nltk.pos_tag(nltk.word_tokenize(s)))
print(pos_tags)
```

```
[[('Japan', 'NNP'), ("'s", 'POS'), ('feudal', 'JJ'), ('era', 'NN'), ('was', 'VBD'), ('characterized', 'VBN'), ('by', 'IN'), ('the', 'DT'), ('emergence', 'NN'), ('and', 'CC'), ('dominar
```

```
test_data = pos_tags
tagger = RegexpTagger(patterns)
print ("Accuracy of Regex Tagger : ", tagger.evaluate(test_data))
```

```
Accuracy of Regex Tagger :  0.09747292418772563
```

F TAGGER

```
#importing all the needed libraries
import pandas as pd
import nltk
import sklearn
import sklearn_crfsuite
import scipy.stats
import math, string, re
```

```
data = {}
data['train'] = pd.read_csv('corpus.csv')
data['test'] = pd.read_csv('corpus1.csv',)
```

```
data['test'] = pd.read_csv('corpus1.csv',)

import numpy as np
data["train"]["POS_TAG"] = np.where(data["train"]["WORD"] == (',','or '!' or '.' or '?' or '-' or "'" or '"'), 'PUNCT',data["train"]["POS_TAG"])
data["test"]["POS_TAG"] = np.where(data["test"]["WORD"] == (',','or '!' or '.' or '?' or '-' or "'" or '"'), 'PUNCT',data["test"]["POS_TAG"])

print(data['train'], data['test'], sep = '\n\n')
```

```
          ID     WORD POS_TAG
0          0       At      IN
1          1    Tokyo     NNP
2          2        ,   PUNCT
3          3      the      DT
4          4   Nikkei     NNP
...      ...      ...     ...
23160  23160  quarter      NN
23161  23161       of      IN
23162  23162     next      JJ
23163  23163     year      NN
23164  23164        .       .

[23165 rows x 3 columns]

      ID    WORD POS_TAG
0      0   Japan     NNP
1      1      's     POS
2      2  feudal      JJ
3      3     era      NN
4      4     was     VBD
..   ...     ...     ...
272  272      in      IN
273  273    1592      CD
274  274     and      CC
```

```
def word2features(sent, i):
    word = sent[i][0]

    features = {
        'bias': 1.0,
        'word': word,
        'len(word)': len(word),
        'word[:4]': word[:4],
        'word[:3]': word[:3],
        'word[:2]': word[:2],
        'word[-3:]': word[-3:],
        'word[-2:]': word[-2:],
        'word[-4:]': word[-4:],
        'word.lower()': word.lower(),
        'word.stemmed': re.sub(r'(.{2,}?)([aeiougyn]+$)',r'\1', word.lower()),
        'word.ispunctuation': (word in string.punctuation),
        'word.isdigit()': word.isdigit(),
    }
    if i > 0:
        word1 = sent[i-1][0]
        features.update({
            '-1:word': word1,
            '-1:len(word)': len(word1),
            '-1:word.lower()': word1.lower(),
            '-1:word.stemmed': re.sub(r'(.{2,}?)([aeiougyn]+$)',r'\1', word1.lower()),
            '-1:word[:3]': word1[:3],
            '-1:word[:2]': word1[:2],
            '-1:word[-3:]': word1[-3:],
            '-1:word[-2:]': word1[-2:],
```

```
    if i > 1:
      word2 = sent[i-2][0]
      features.update({
          '-2:word': word2,
          '-2:len(word)': len(word2),
          '-2:word.lower()': word2.lower(),
          '-2:word[:3]': word2[:3],
          '-2:word[:2]': word2[:2],
          '-2:word[-3:]': word2[-3:],
          '-2:word[-2:]': word2[-2:],
          '-2:word.isdigit()': word2.isdigit(),
          '-2:word.ispunctuation': (word2 in string.punctuation),
      })

    if i < len(sent)-1:
        word1 = sent[i+1][0]
        features.update({
            '+1:word': word1,
            '+1:len(word)': len(word1),
            '+1:word.lower()': word1.lower(),
            '+1:word[:3]': word1[:3],
            '+1:word[:2]': word1[:2],
            '+1:word[-3:]': word1[-3:],
            '+1:word[-2:]': word1[-2:],
            '+1:word.isdigit()': word1.isdigit(),
            '+1:word.ispunctuation': (word1 in string.punctuation),
        })

    else:
        features['EOS'] = True
```

```
def sent2features(sent):
    return [word2features(sent, i) for i in range(len(sent))]

def sent2labels(sent):
    return [word[1] for word in sent]

def sent2tokens(sent):
    return [word[0] for word in sent]

def format_data(csv_data):
    sents = []
    a=0
    for i in range(len(csv_data)):
      # print(csv_data.iloc[i, 0])
      if math.isnan(csv_data.iloc[i, 0]):
          continue
      elif csv_data.iloc[i, 0] == 1.0:
        sents.append([[csv_data.iloc[i, 1], csv_data.iloc[i, 2]]])
      else:
        if a==0:
          sents.append([csv_data.iloc[i, 1], csv_data.iloc[i, 2]])
          a=1
        else:sents[-1].append([csv_data.iloc[i, 1], csv_data.iloc[i, 2]])
    for sent in sents:
      for i, word in enumerate(sent):
        if type(word[0]) != str:
          del sent[i]
    return sents
```

```
train_sents = format_data(data['train'])
# print(train_sents)
test_sents = format_data(data['test'])

Xtrain = [sent2features(s) for s in train_sents]
ytrain = [sent2labels(s) for s in train_sents]

Xtest = [sent2features(s) for s in test_sents]
ytest = [sent2labels(s) for s in test_sents]
```

```
crf = sklearn_crfsuite.CRF(
    algorithm = 'lbfgs',
    c1 = 0.25,
    c2 = 0.3,
    max_iterations = 100,
    all_possible_transitions=True
)
try:
    crf.fit(Xtrain, ytrain)
except AttributeError:
    pass
#training the model
```

+ Code + Text

```
from sklearn_crfsuite import metrics

ypred = crf.predict(Xtest)
print("Accuracy of CRF Tagger : ",metrics.flat_accuracy_score(ytest, ypred))
```

```
from nltk.tag import brill, brill_trainer, UnigramTagger, BigramTagger, TrigramTagger
from nltk.tag import DefaultTagger
```

```
def train_brill_tagger(initial_tagger, train_sents, **kwargs):
    templates = [
            brill.Template(brill.Pos([-1])),
            brill.Template(brill.Pos([1])),
            brill.Template(brill.Pos([-2])),
            brill.Template(brill.Pos([2])),
            brill.Template(brill.Pos([-2, -1])),
            brill.Template(brill.Pos([1, 2])),
            brill.Template(brill.Pos([-3, -2, -1])),
            brill.Template(brill.Pos([1, 2, 3])),
            brill.Template(brill.Pos([-1]), brill.Pos([1])),
            brill.Template(brill.Word([-1])),
            brill.Template(brill.Word([1])),
            brill.Template(brill.Word([-2])),
            brill.Template(brill.Word([2])),
            brill.Template(brill.Word([-2, -1])),
            brill.Template(brill.Word([1, 2])),
            brill.Template(brill.Word([-3, -2, -1])),
            brill.Template(brill.Word([1, 2, 3])),
            brill.Template(brill.Word([-1]), brill.Word([1])),
            ]
```

```
    trainer = brill_trainer.BrillTaggerTrainer(
            initial_tagger, templates, deterministic = True)

    return trainer.train(train_sents, **kwargs)

def backoff_tagger(train_sentences, tagger_classes, backoff=None):
   for cls in tagger_classes:
      backoff = cls(train_sentences, backoff=backoff)
   return backoff
```

```
# Initializing
default_tag = DefaultTagger('NN')

# initializing training and testing set
train_data = treebank.tagged_sents()[:3000]
test_data = pos_tags #training on my own corpus

initial_tag = backoff_tagger(train_data, [UnigramTagger, BigramTagger, TrigramTagger], backoff = default_tag)

a = initial_tag.evaluate(test_data)
print ("Accuracy of Backoff Tagger : ", a)


brill_tag = train_brill_tagger(initial_tag, train_data)
b = brill_tag.evaluate(test_data)

print ("Accuracy of Brill Tagger : ", b)
```

```
Accuracy of Backoff Tagger :  0.6642599277978339
Accuracy of Brill Tagger :  0.6642599277978339
```

REFERENCES:

1. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00561-y
2. https://towardsdatascience.com/markov-and-hidden-markov-model-3eec42298d75
3. https://medium.com/ml2vec/overview-of-conditional-random-fields-68a2a20fa541
4. https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba