Krishna Kant Verma

Roll No 2211CS19

```
import pprint
import numpy as np
import pandas as pd
import nltk
import re
import warnings
warnings.filterwarnings("ignore")
import requests
import matplotlib.pyplot as plt
import seaborn as sns
import time
import random
from google.colab import files
from sklearn.model_selection import train_test_split
from nltk.tokenize import word_tokenize
```

```
df = pd.read_csv("NER-Dataset-Train.csv")
```

```
df.head()
```

	@LewisDixon\tO
0	Trust\tO
1	me\tO
2	!\tO
3	im\tO
4	gonna\tO

```
df.shape
```

```
(1199, 1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1199 entries, 0 to 1198
Data columns (total 1 columns):
 #   Column         Non-Null Count  Dtype
```

```
dtypes: object(1)
memory usage: 9.5+ KB
```

```
df.describe()
```

	@LewisDixon\tO
count	1142
unique	662
top	,\tO
freq	36

```
df.isnull().sum()
```

```
@LewisDixon\tO    57
dtype: int64
```

```
sentences_words=[]
sentences_tags=[]
file = open('NER-Dataset-Train.txt', 'r')
lines = file.readlines()
temp_sentence_words=[]
temp_sentence_tags=[]
for line in lines:
    if line=="\n":#Sentences ends at every blank line
        if len(temp_sentence_words)==0:#If zero length sentence is formed, then ignore
            continue
        sentences_words.append(temp_sentence_words)
        temp_sentence_words=[]
        sentences_tags.append(temp_sentence_tags)
        temp_sentence_tags=[]
        continue
    temp=line.split("\t")#splitting to get the tag and the word
    temp[1]=temp[1].split("\n")[0]
    temp_sentence_words.append(temp[0])
    temp_sentence_tags.append(temp[1])
file.close()
```

```
sentences_words
```

```
[['@LewisDixon',
  'Trust',
  'me',
  '!',
  'im',
```

```
          '…',
          'gonna',
          'be',
          'bringing',
          'out',
          'music',
          'like',
          'theres',
          'no',
          'tomorrow',
          ',',
          'Be',
          'doing',
          'pure',
          'blog',
          'videos',
          '&amp;',
          'freestyle',
          'videos',
          '#Moesh',
          '!'],
         ['@joshHnumber1fan',
          'its',
          'okay',
          'then',
          '..',
          'make',
          'it',
          'when',
          'it',
          'works',
          ':D'],
         ['Asprin',
          ',',
          'check',
          ',',
          'cup',
          'of',
          'tea',
          ',',
          'check',
          ',',
          'pillow',
          ',',
          'check',
          ',',
          'warm',
          'sleeping',
          'bag',
          ',',
          'check',
          ',',
          'fanfiction',
          'on',

sum(len(new) for new in sentences_words)
```

```
sum(len(row) for row in sentences_words)
sum(len(row) for row in sentences_tags)
len(sentences_words[0])
```

```
25
```

```
list5=[]
for i in range(len(sentences_words)):
    list4=[]
    for j in range(len(sentences_words[i])):
        list1=[]
        list1.append(sentences_words[i][j])
        list1.append(sentences_tags[i][j])
        list4.append(tuple(list1))
    list5.append(list4)
```

```
len(list5)
```

```
900
```

```
sum(len(row) for row in list5)
```

```
17480
```

```
# Splitting into train and test
import random
random.seed(1)
train_set, test_set = train_test_split(list5,test_size=0.30)
print(len(train_set))
print(len(test_set))
```

```
630
270
```

```
# Getting list of tagged words
Tagged_words = [tup for sent in train_set for tup in sent]
len(Tagged_words)
```

```
12383
```

```
# Word_Token
Word_Token = [pair[0] for pair in Tagged_words]
print(len(Word_Token))
```

```
12383
```

```
# vocabulary
```

```
V = set(Word_Token)
print(len(V))
```

```
3955
```

```
# number of tags
T = set([pair[1] for pair in Tagged_words])
print(len(T))
T
```

```
3
{'B', 'I', 'O'}
```

Emission Probabilities P(w/t)

```
#Calculating P(w/t)
t = len(T)
v = len(V)
w_given_t = np.zeros((t, v))

#Calculating  Probability of a word given a tag: Emission Probability
def prob_of_word_given_tag(word, tag, train_bag = Tagged_words):
    tag_list = [pair for pair in train_bag if pair[1]==tag]
    count_tag = len(tag_list)
    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)
```

Transition Probabilities P(t2/t1)

```
#Calculating the Probability of a tag given a tag: P(t2/t1) i.e. Transition Probability

def t2_given_t1(t2, t1, train_bag = Tagged_words):
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])      #Counting number of occurences of t1
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2: #Counting number of times t2 follows t1
            count_t2_t1 += 1
    return (count_t2_t1, count_t1)
```

Transition matrix : Containing Probabilities of Transition From Tag1 to Tag2

```
# We will now create a Transition matrix of tags of dimension t x t
# Considering each column t2 and each row as t1
```

```
# considering each column t2 and each row as t1
#Thus element M(i, j) is equivalent to Probability of tj given ti : P(tj given ti)

tags_matrix = np.zeros((len(T), len(T)), dtype='float32')
for i, t1 in enumerate(list(T)):
    for j, t2 in enumerate(list(T)):
        tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]


tags_matrix
```

```
array([[0.33207548, 0.01509434, 0.6528302 ],
       [0.45501286, 0.        , 0.54498714],
       [0.        , 0.03273936, 0.96717536]], dtype=float32)
```

```
# convert the matrix to a df for better readability
tags_df = pd.DataFrame(tags_matrix, columns = list(T), index=list(T))


tags_df
```

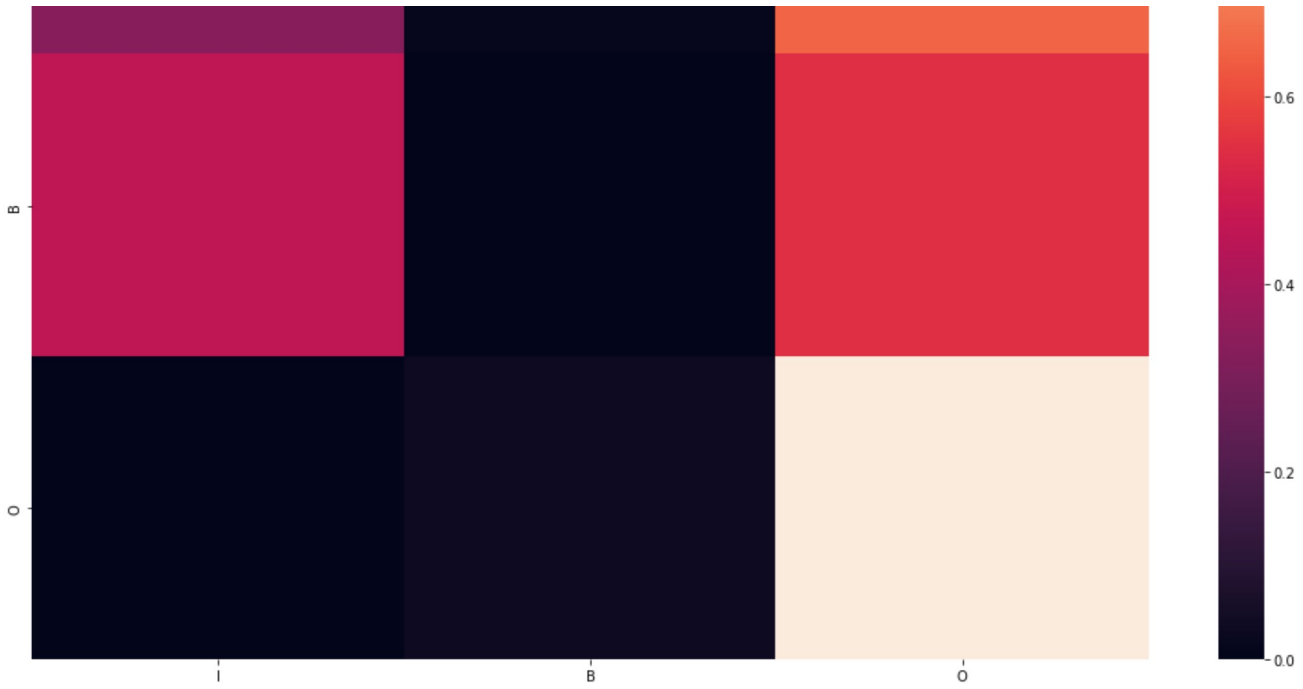
	I	B	O
I	0.332075	0.015094	0.652830
B	0.455013	0.000000	0.544987
O	0.000000	0.032739	0.967175

```
tags_df.loc['O', :]
```

```
I    0.000000
B    0.032739
O    0.967175
Name: O, dtype: float32
```

Visualizing the Transition Matrix on Heat Map for better intuition

```
# Heatmap of Tags matrix where T(i, j) = P(tag j given tag i)
plt.figure(figsize=(18, 12))
sns.heatmap(tags_df)
plt.show()
```




```
len(train_set)
```

```
630
```

Viterbi Algorithm

```
# Viterbi_Algorithm Function !
def Viterbi_Algorithm(words, train_bag = Tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))

    for key, word in enumerate(words):
        #initializing a list of probability column for a given observation
        p = []
        for tag in T:
            if key == 0:
                transition_probability = tags_df.loc['O', tag]      # P(tag|start) = P(tag
            else:
```

```
                transition_probability = tags_df.loc[state[-1], tag]

            #Calculating emission and state probabilities
            emission_probability = prob_of_word_given_tag(words[key], tag)[0]/prob_of_word_
            state_probability = emission_probability * transition_probability
            p.append(state_probability)

        pmax = max(p)
        # Finding the state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
```

Evaluating on Test Set

Testing

5-fold cross validation

```
num_sents = len(list5)
k = 5
foldsize = int(num_sents/k)
foldsize
```

```
180
```

```
fold_accurracies = []
fold_incorrect_tags =[]
timetaken=[]
tagged_seq_collection=[]
test_seq_collection=[]
for f in range(5):
    # Locate the test set in the fold.
    test_set = list5[f*foldsize:f*foldsize+foldsize]
    # Use the rest of the sent not in test for training.
    train_set = list5[:f*foldsize] + list5[f*foldsize+foldsize:]


    # Getting list of tagged words
    train_tagged_words = [tup for sent in train_set for tup in sent]
    #len(train_tagged_words)

    # tokens
    tokens = [pair[0] for pair in train_tagged_words]

    # vocabulary
    V = set(tokens)
```

```
# print(len(V))

# number of tags
T = set([pair[1] for pair in train_tagged_words])
#len(T)

#Calculating P(w/t)
t = len(T)
v = len(V)
w_given_t = np.zeros((t, v))

#Calculating the Probability of a tag given a tag: P(t2/t1) i.e. Transition Probabilit
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])      #Counting number of occurences of t
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2: #Counting number of times t2 follow
            count_t2_t1 += 1
    return (count_t2_t1, count_t1)

# We will now create a Transition matrix of tags of dimension t x t
# Considering each column t2 and each row as t1
#Thus element M(i, j) is equivalent to Probability of tj given ti : P(tj given ti)

tags_matrix = np.zeros((len(T), len(T)), dtype='float32')
for i, t1 in enumerate(list(T)):
    for j, t2 in enumerate(list(T)):
        tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]

tags_df = pd.DataFrame(tags_matrix, columns = list(T), index=list(T))

# Running the Viterbi algorithm on a few sample sentences

random.seed(1)

# choose random 5 sents
#rndom = [random.randint(1,len(test_set)) for x in range(5)]

# list of sents
#test_run = [test_set[i] for i in rndom]

# list of tagged words
test_run_base = [tup for sent in test_set for tup in sent]

# list of untagged words
test_tagged_words = [tup[0] for sent in test_set for tup in sent]

#test_run
# tagging the test sentences
```

```
    # tagging the test sentences
    start = time.time()
    tagged_seq = Viterbi_Algorithm(test_tagged_words)
    tagged_seq_collection.append(tagged_seq)
    test_seq_collection.append(test_set)
    end = time.time()
    difference = end-start
    timetaken.append(difference)

    # accuracy
    check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
    accuracy = len(check)/len(tagged_seq)
    fold_accurracies.append(accuracy)

    #Incorrect Tagging Tracker
    incorrect_tagged_cases = [[test_run_base[i-1],j] for i, j in enumerate(zip(tagged_seq,
    fold_incorrect_tags.append(incorrect_tagged_cases)


    print("Fold", f)
    print('From ', f*foldsize, 'to', f*foldsize+foldsize)
    print('Accuracy =', accuracy )
    print("Time Taken :",timetaken[f])
```

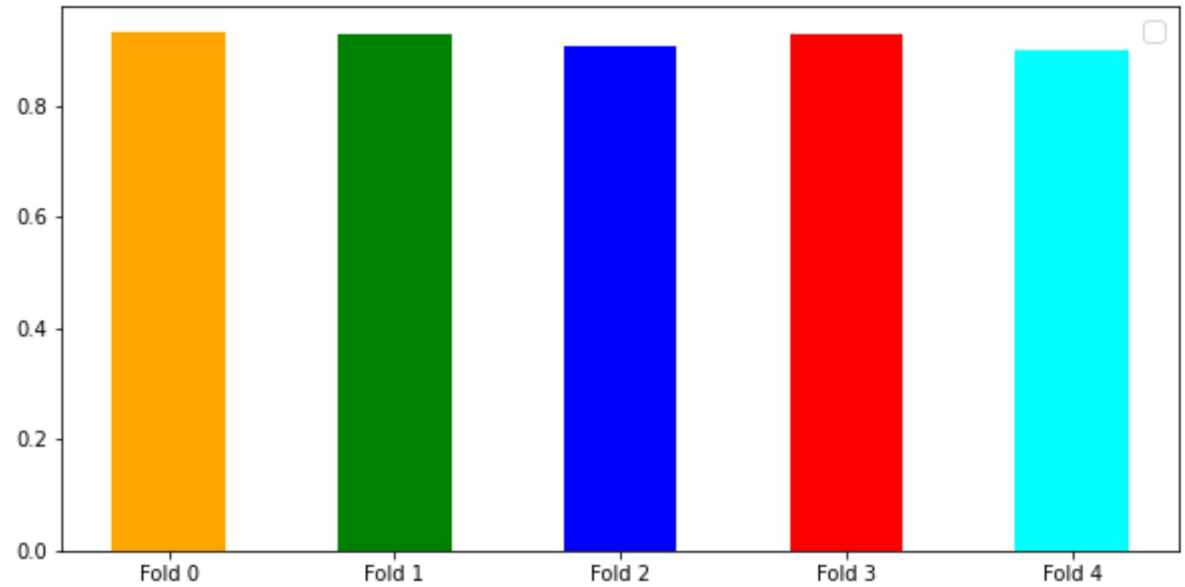
```
 Fold 0
 From  0 to 180
 Accuracy = 0.9319327731092437
 Time Taken : 30.72916316986084
 Fold 1
 From  180 to 360
 Accuracy = 0.9268085106382978
 Time Taken : 28.111690521240234
 Fold 2
 From  360 to 540
 Accuracy = 0.9060950714494022
 Time Taken : 27.83026385307312
 Fold 3
 From  540 to 720
 Accuracy = 0.928084138715179
 Time Taken : 27.41340947151184
 Fold 4
 From  720 to 900
 Accuracy = 0.8993600930773705
 Time Taken : 27.794015884399414
```

```
a1=fold_accurracies[0]
a2=fold_accurracies[1]
a3=fold_accurracies[2]
a4=fold_accurracies[3]
a5=fold_accurracies[4]

labels=['Fold 0','Fold 1','Fold 2','Fold 3', 'Fold 4']
```

```
f, ax = plt.subplots(figsize=(10,5)) # set the size that you'd like (width, height)
plt.bar(labels, [a1,a2,a3,a4,a5], color=['orange','green','blue', 'red','cyan'],width=0.5)
ax.legend(fontsize = 14)
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
<matplotlib.legend.Legend at 0x7f508541c6d0>
```




```
fold_incorrect_tags
```

```
[[[('of', 'O'), (('partying', 'I'), ('partying', 'O'))],
  [('to', 'O'), (('tumblr', 'I'), ('tumblr', 'B'))],
  [('#partyyy', 'O'), (('@GOBLUE_FUCKosu', 'I'), ('@GOBLUE_FUCKosu', 'O'))],
  [('our', 'O'), (('10th', 'I'), ('10th', 'O'))],
  [('10th', 'O'), (('grade', 'I'), ('grade', 'O'))],
  [('.', 'O'), (('smh', 'I'), ('smh', 'O'))],
  [('with', 'O'), (('the', 'O'), ('the', 'B'))],
  [('daughter', 'O'), (('@daxx_d24', 'I'), ('@daxx_d24', 'O'))],
  [('.', 'O'), (('Got', 'I'), ('Got', 'O'))],
  [('next', 'O'), (('question', 'I'), ('question', 'O'))],
  [('few', 'O'), (('moments', 'I'), ('moments', 'O'))],
  [('the', 'O'), (('answer', 'I'), ('answer', 'O'))],
  [('on', 'O'), (('JavaMonkeys', 'I'), ('JavaMonkeys', 'O'))],
  [('...', 'O'),
   (('http://fb.me/GTRhPujh', 'I'), ('http://fb.me/GTRhPujh', 'O'))],
  [('huge', 'O'), (('player', 'I'), ('player', 'O'))],
  [(':', 'O'), (('http://bit.ly/cNarLp', 'I'), ('http://bit.ly/cNarLp', 'O'))],
  [('http://bit.ly/cNarLp', 'O'), (('The', 'I'), ('The', 'O'))],
  [(':', 'O'), (('A', 'O'), ('A', 'B'))],
  [('!', 'O'), (('@BeliebinMinajj', 'I'), ('@BeliebinMinajj', 'O'))],
  [('OMG', 'O'), (('MILLIONS', 'I'), ('MILLIONS', 'O'))],
  [('on', 'O'), (('repeat', 'I'), ('repeat', 'O'))],
  [('last', 'O'), (('lesson', 'I'), ('lesson', 'O'))],
  [('around', 'O'), (('9pm', 'I'), ('9pm', 'O'))],
  [('Gaye', 'I'), (('Day', 'I'), ('Day', 'O'))],
  [('radio', 'O'), (('station', 'I'), ('station', 'O'))],
  [('there', 'O'), (('number', 'I'), ('number', 'O'))],
  [('number', 'O'), (('1', 'I'), ('1', 'O'))],
```

```
[('number', 'O'), (('1', 'I'), ('1', 'O'))],
[('heart', 'O'), (('vacancy', 'I'), ('vacancy', 'O'))],
[('#fail', 'O'), (('photo', 'I'), ('photo', 'O'))],
[('from', 'O'), (('@joshbuisch', 'I'), ('@joshbuisch', 'O'))],
[('!', 'O'), (('http://bit.ly/bA3lUl', 'I'), ('http://bit.ly/bA3lUl', 'O'))],
[('http://www.forexcrunch.com/forex-articles-for-the-weekend-september-18lo/',
  'O'),
 (('#uknouugly', 'I'), ('#uknouugly', 'O'))],
[('when', 'O'), (('#TeamFollowBack', 'I'), ('#TeamFollowBack', 'O'))],
[('Is', 'O'), (('sad', 'I'), ('sad', 'O'))],
[('missing', 'O'), (('Cowboy', 'I'), ('Cowboy', 'B'))],
[('Cowboy', 'B'), (('Mouth', 'O'), ('Mouth', 'I'))],
[('RT', 'O'), (('@fredthompson', 'I'), ('@fredthompson', 'O'))],
[(':', 'O'), (('WH', 'I'), ('WH', 'O'))],
[('WH', 'O'), (('rejects', 'I'), ('rejects', 'O'))],
[('"', 'O'), (('global', 'I'), ('global', 'O'))],
[('global', 'O'), (('warming', 'I'), ('warming', 'O'))],
[('warming', 'O'), (('",', 'I'), ('",', 'O'))],
[('",', 'O'), (('favors', 'I'), ('favors', 'O'))],
[('favors', 'O'), (('term', 'I'), ('term', 'O'))],
[('"', 'O'), (('global', 'I'), ('global', 'O'))],
[('global', 'O'), (('climate', 'I'), ('climate', 'O'))],
[('climate', 'O'), (('disruption', 'I'), ('disruption', 'O'))],
[('".', 'O'), (('Ya', 'I'), ('Ya', 'O'))],
[('we', 'O'), (('used', 'I'), ('used', 'O'))],
[('N', 'I'), (('.', 'O'), ('.', 'I'))],
[('N', 'I'), (('.', 'O'), ('.', 'I'))],
[(';)', 'O'), (('Hurry', 'I'), ('Hurry', 'O'))],
[('!', 'O'), (('Santy', 'I'), ('Santy', 'B'))],
[('be', 'O'), (('Leaving', 'I'), ('Leaving', 'O'))],
[('-', 'O'), (('#Eskorte', 'I'), ('#Eskorte', 'O'))],
```

Maximum Accuracy

```
print(max(fold_accurracies))
F=fold_accurracies.index(max(fold_accurracies))
print("Fold ",F)
```

```
0.9319327731092437
Fold  0
```

Class wise Accuracy

```
#We will calculate Class wise of Fold with maximum accuracy
fold_incorrect_tags[F]
```

```
[[('of', 'O'), (('partying', 'I'), ('partying', 'O'))],
 [('to', 'O'), (('tumblr', 'I'), ('tumblr', 'B'))],
 [('#partyyy', 'O'), (('@GOBLUE_FUCKosu', 'I'), ('@GOBLUE_FUCKosu', 'O'))],
 [('our', 'O'), (('10th', 'I'), ('10th', 'O'))],
 [('10th', 'O'), (('grade', 'I'), ('grade', 'O'))],
 [('', 'O'), (('smh', 'I'), ('smh', 'O'))],
```

```
[('.', 'O'), (('Smh', 'I'), ('Smh', 'O'))],
[('with', 'O'), (('the', 'O'), ('the', 'B'))],
[('daughter', 'O'), (('@daxx_d24', 'I'), ('@daxx_d24', 'O'))],
[('.', 'O'), (('Got', 'I'), ('Got', 'O'))],
[('next', 'O'), (('question', 'I'), ('question', 'O'))],
[('few', 'O'), (('moments', 'I'), ('moments', 'O'))],
[('the', 'O'), (('answer', 'I'), ('answer', 'O'))],
[('on', 'O'), (('JavaMonkeys', 'I'), ('JavaMonkeys', 'O'))],
[('...', 'O'),
 (('http://fb.me/GTRhPujh', 'I'), ('http://fb.me/GTRhPujh', 'O'))],
[('huge', 'O'), (('player', 'I'), ('player', 'O'))],
[(':', 'O'), (('http://bit.ly/cNarLp', 'I'), ('http://bit.ly/cNarLp', 'O'))],
[('http://bit.ly/cNarLp', 'O'), (('The', 'I'), ('The', 'O'))],
[(':', 'O'), (('A', 'O'), ('A', 'B'))],
[('!', 'O'), (('@BeliebinMinajj', 'I'), ('@BeliebinMinajj', 'O'))],
[('OMG', 'O'), (('MILLIONS', 'I'), ('MILLIONS', 'O'))],
[('on', 'O'), (('repeat', 'I'), ('repeat', 'O'))],
[('last', 'O'), (('lesson', 'I'), ('lesson', 'O'))],
[('around', 'O'), (('9pm', 'I'), ('9pm', 'O'))],
[('Gaye', 'I'), (('Day', 'I'), ('Day', 'O'))],
[('radio', 'O'), (('station', 'I'), ('station', 'O'))],
[('there', 'O'), (('number', 'I'), ('number', 'O'))],
[('number', 'O'), (('1', 'I'), ('1', 'O'))],
[('heart', 'O'), (('vacancy', 'I'), ('vacancy', 'O'))],
[('#fail', 'O'), (('photo', 'I'), ('photo', 'O'))],
[('from', 'O'), (('@joshbuisch', 'I'), ('@joshbuisch', 'O'))],
[('!', 'O'), (('http://bit.ly/bA3lUl', 'I'), ('http://bit.ly/bA3lUl', 'O'))],
[('http://www.forexcrunch.com/forex-articles-for-the-weekend-september-18lo/',
  'O'),
 (('#uknouugly', 'I'), ('#uknouugly', 'O'))],
[('when', 'O'), (('#TeamFollowBack', 'I'), ('#TeamFollowBack', 'O'))],
[('Is', 'O'), (('sad', 'I'), ('sad', 'O'))],
[('missing', 'O'), (('Cowboy', 'I'), ('Cowboy', 'B'))],
[('Cowboy', 'B'), (('Mouth', 'O'), ('Mouth', 'I'))],
[('RT', 'O'), (('@fredthompson', 'I'), ('@fredthompson', 'O'))],
[(':', 'O'), (('WH', 'I'), ('WH', 'O'))],
[('WH', 'O'), (('rejects', 'I'), ('rejects', 'O'))],
[('"', 'O'), (('global', 'I'), ('global', 'O'))],
[('global', 'O'), (('warming', 'I'), ('warming', 'O'))],
[('warming', 'O'), (('",', 'I'), ('",', 'O'))],
[('",', 'O'), (('favors', 'I'), ('favors', 'O'))],
[('favors', 'O'), (('term', 'I'), ('term', 'O'))],
[('"', 'O'), (('global', 'I'), ('global', 'O'))],
[('global', 'O'), (('climate', 'I'), ('climate', 'O'))],
[('climate', 'O'), (('disruption', 'I'), ('disruption', 'O'))],
[('".', 'O'), (('Ya', 'I'), ('Ya', 'O'))],
[('we', 'O'), (('used', 'I'), ('used', 'O'))],
[('N', 'I'), (('.', 'O'), ('.', 'I'))],
[('N', 'I'), (('.', 'O'), ('.', 'I'))],
[(';)', 'O'), (('Hurry', 'I'), ('Hurry', 'O'))],
[('!', 'O'), (('Santy', 'I'), ('Santy', 'B'))],
[('be', 'O'), (('Leaving', 'I'), ('Leaving', 'O'))],
[('-', 'O'), (('#Eskorte', 'I'), ('#Eskorte', 'O'))],
```

```
#Total Incorrect Tagging in the chosen Fold is :
```

```
ic=len(fold_incorrect_tags[F])
print("Total Incorrect Tagging in the chosen Fold is :",ic)
```

```
Total Incorrect Tagging in the chosen Fold is : 243
```

```
length=len((fold_incorrect_tags[F]))
length
```

```
243
```

```
lista=[]
for i in range(0,length):
    lista.append(((fold_incorrect_tags[F][i])[1])[0])
len(lista)
```

```
243
```

```
lista
```

```
[('partying', 'I'),
 ('tumblr', 'I'),
 ('@GOBLUE_FUCKosu', 'I'),
 ('10th', 'I'),
 ('grade', 'I'),
 ('smh', 'I'),
 ('the', 'O'),
 ('@daxx_d24', 'I'),
 ('Got', 'I'),
 ('question', 'I'),
 ('moments', 'I'),
 ('answer', 'I'),
 ('JavaMonkeys', 'I'),
 ('http://fb.me/GTRhPujh', 'I'),
 ('player', 'I'),
 ('http://bit.ly/cNarLp', 'I'),
 ('The', 'I'),
 ('A', 'O'),
 ('@BeliebinMinajj', 'I'),
 ('MILLIONS', 'I'),
 ('repeat', 'I'),
 ('lesson', 'I'),
 ('9pm', 'I'),
 ('Day', 'I'),
 ('station', 'I'),
 ('number', 'I'),
 ('1', 'I'),
 ('vacancy', 'I'),
 ('photo', 'I'),
 ('@joshbuisch', 'I'),
 ('http://bit.ly/bA3lUl', 'I'),
 ('#uknouugly', 'I'),
 ('#TeamFollowBack', 'I'),
```

```
 ('sad', 'I'),
 ('Cowboy', 'I'),
 ('Mouth', 'O'),
 ('@fredthompson', 'I'),
 ('WH', 'I'),
 ('rejects', 'I'),
 ('global', 'I'),
 ('warming', 'I'),
 ('",', 'I'),
 ('favors', 'I'),
 ('term', 'I'),
 ('global', 'I'),
 ('climate', 'I'),
 ('disruption', 'I'),
 ('Ya', 'I'),
 ('used', 'I'),
 ('.', 'O'),
 ('.', 'O'),
 ('Hurry', 'I'),
 ('Santy', 'I'),
 ('Leaving', 'I'),
 ('#Eskorte', 'I'),
 ('#Massasje', 'I'),
 ('#Norge', 'I'),
 ('remembered', 'I'),
```

```
#Using lista we are extracting the tags that were incorrectly attached to some word and sto
listb=[]
for i in range(0,len(lista)):
    listb.append((lista[i][1]))
```

```
#We will count number of times each tag in listb was incorrectly attached to some word
from collections import Counter
Counter(listb)
dicta=dict(Counter(listb))
print(dicta)
```

```
{'I': 220, 'O': 20, 'B': 3}
```

```
actual_freq_of_tag=[]
for i in range(0,len(test_seq_collection[F])):
    for j in range(0,len(test_seq_collection[F][i])):
        actual_freq_of_tag.append((test_seq_collection[F][i][j][1]))
```

```
from collections import Counter
Counter(actual_freq_of_tag)
dictb=dict(Counter(actual_freq_of_tag))
print(dictb)
```

```
{'O': 3375, 'B': 116, 'I': 79}
```

```
#of times a tag appeared in the dataset
appearance=[]
for i in range(0,len(tagged_seq_collection[F])):
        appearance.append((tagged_seq_collection[F][i][1]))
from collections import Counter
Counter(appearance)
dictappear=dict(Counter(appearance))
print(dictappear)
```

```
{'O': 3197, 'B': 85, 'I': 288}
```

```
keyList=dictappear.keys()
keyList
```

```
dict_keys(['O', 'B', 'I'])
```

```
#Creating a dictionary 'd' with all tags that were used through out the process and initia
d={}
for i in keyList:
    d[i] = 0
```

```
#With the help of dicta I will insert values in this new dict 'd'.
for i in dicta.keys():
    d[i] = dicta[i]
d
```

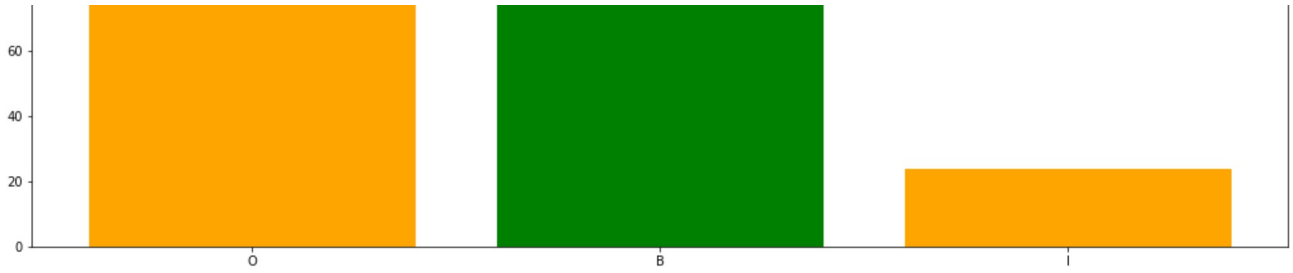
```
{'O': 20, 'B': 3, 'I': 220}
```

```
#Using both dictb and d , Calculating the class wise accuracy and storing it in a new dict
tag_ac={}
for i in d.keys():
    x = ((dictappear[i]-d[i])/(dictappear[i]))*100
    tag_ac[i]=x
tag_ac
```

```
{'O': 99.37441351266813, 'B': 96.47058823529412, 'I': 23.61111111111111}
```

```
import matplotlib.pylab as plt
f, ax = plt.subplots(figsize=(18,5)) # set the size that you'd like (width, height)
plt.bar(tag_ac.keys(), tag_ac.values(), color=['orange','green'],align='center')
ax.legend(fontsize = 14)
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
<matplotlib.legend.Legend at 0x7f5084d9f710>
```




```
#tagged_seq_collection is a list of tagged_sequences of each fold
#test_seq_collection is a list of test_seq of each fold
print(len(tagged_seq_collection))
print(len(test_seq_collection))
```

```
5
5
```

```
#Extracting the Tagged Sequences and Test Sequences of the maximum fold
tseq=[]
tset=[]
for i in range(0,len(test_seq_collection[F])):
    for j in range(0,len(test_seq_collection[F][i])):
        tset.append((test_seq_collection[F][i][j][1]))


for i in range(0,len(tagged_seq_collection[F])):
        tseq.append((tagged_seq_collection[F][i][1]))


len(tseq),len(tset)
```

```
(3570, 3570)
```

```
#Storing allocated tags in list ltseq
ltseq=tseq


#Storing  Actual tags of test_set in list ltset
ltset=tset
ltset
```

```
['O',
 'O',
 'O',
 'O',
```

```
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
```

```
#Ultseq is list of unique tags present in the ltseq
#Ultset is list of unique tags present in the ltset
Ultset=list(set(ltset))
Ultseq=list(set(ltseq))
max(len(Ultset),len(Ultseq))
```

```
3
```

```
Ultset
```

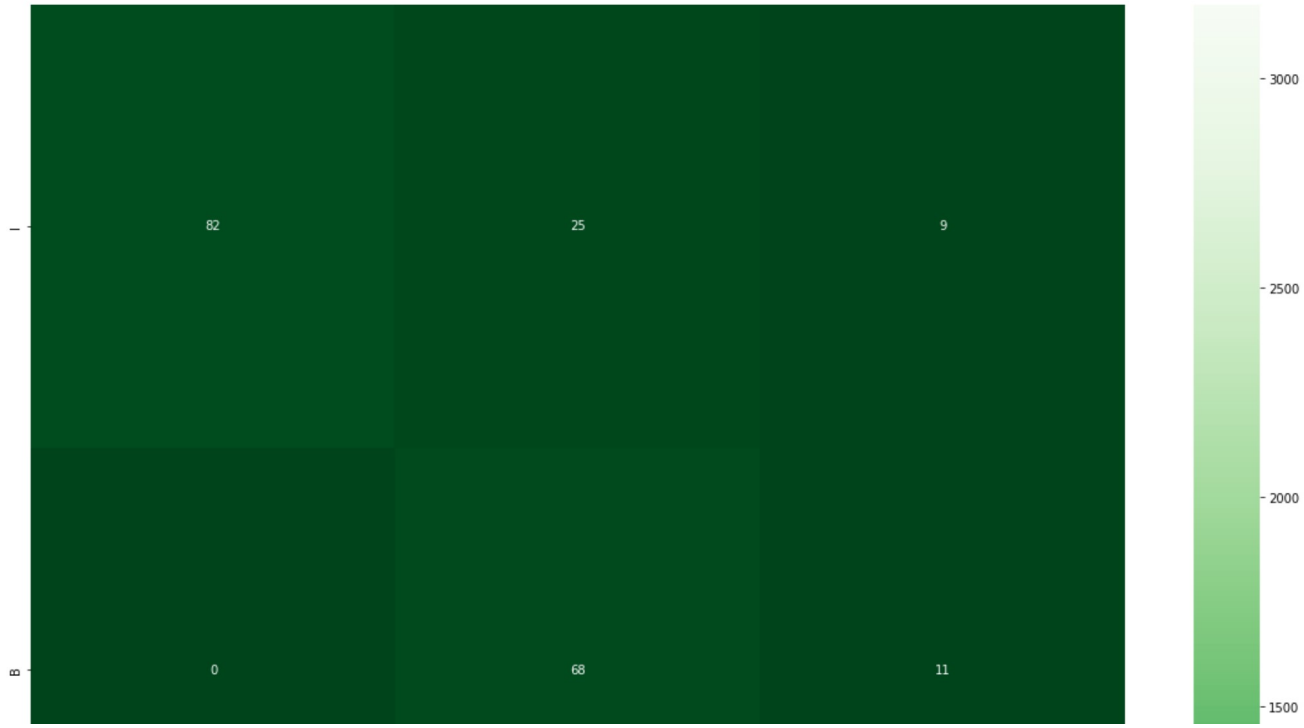
```
['I', 'B', 'O']
```

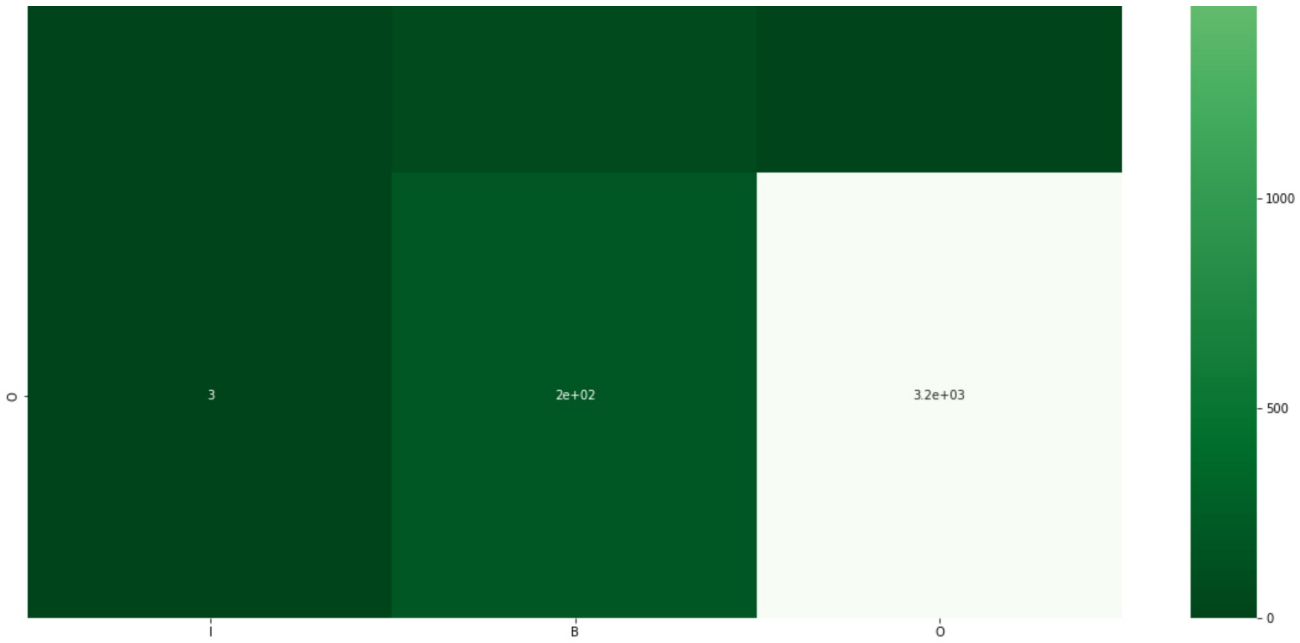
```
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
y_true = ltset
y_pred = ltseq
array=confusion_matrix(y_true, y_pred)
array
```

```
array([[  82,   25,    9],
       [   0,   68,   11],
       [   3,  195, 3177]])
```

```
df_cm = pd.DataFrame(array, index = [i for i in Ultset],columns = [i for i in Ultset])
plt.figure(figsize = (20,20))
sn.heatmap(df_cm, annot=True,cmap='Greens_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5084c62f50>
```






```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_true, y_pred))
```

```
                precision    recall  f1-score   support

             B       0.96      0.71      0.82       116
             I       0.24      0.86      0.37        79
             O       0.99      0.94      0.97      3375

      accuracy                           0.93      3570
     macro avg       0.73      0.84      0.72      3570
  weighted avg       0.98      0.93      0.95      3570
```

We will now feed the NER-Dataset--TestSet.csv into our model.

```
dafna=pd.read_csv("NER_Dataset_TestSet.csv")
```

```
dafra=pd.read_csv( NER-Dataset-TestSet.csv )
```

```
dafra.head()
```

	@SammieLynnsMom
0	@tg1.781
1	they
2	will
3	be
4	all

```
dafra.shape
```

```
(2001, 1)
```

```
dafra.isnull().sum()
```

```
@SammieLynnsMom    100
dtype: int64
```

```
dafralist=[]
for i in range(0,1891):
    dafralist.append(dafra["@SammieLynnsMom"][i])
```

```
len(dafralist)
```

```
1891
```

```
start = time.time()
Test_data_tagged_seq = Viterbi_Algorithm(dafralist)
end = time.time()
difference = end-start
print("Time Taken :",difference)
```

```
Time Taken : 15.965312719345093
```

```
Test_data_tagged_seq
```

```
[('@tg1.781', 'I'),
 ('they', 'O'),
 ('will', 'O'),
 ('be', 'O'),
 ('all', 'O'),
 ('done', 'O')
```

```
('done', 'O'),
('by', 'O'),
('Sunday', 'O'),
('trust', 'O'),
('me', 'O'),
('*wink*', 'I'),
(nan, 'I'),
('Made', 'I'),
('it', 'O'),
('back', 'O'),
('home', 'O'),
('to', 'O'),
('GA', 'B'),
('.', 'O'),
('It', 'O'),
('sucks', 'O'),
('not', 'O'),
('to', 'O'),
('be', 'O'),
('at', 'O'),
('Disney', 'I'),
('world', 'O'),
(',', 'O'),
('but', 'O'),
('its', 'O'),
('good', 'O'),
('to', 'O'),
('be', 'O'),
('home', 'O'),
('.', 'O'),
('Time', 'O'),
('to', 'O'),
('start', 'O'),
('planning', 'O'),
('the', 'O'),
('next', 'O'),
('Disney', 'I'),
('World', 'I'),
('trip', 'O'),
('.', 'O'),
(nan, 'I'),
("'", 'I'),
('Breaking', 'B'),
('Dawn', 'I'),
("'", 'I'),
('Returns', 'I'),
('to', 'O'),
('Vancouver', 'I'),
('on', 'O'),
('January', 'I'),
('11th', 'I'),
('http://bit.ly/dbDMs8', 'I'),
(nan, 'I'),
```

Colab paid products - Cancel contracts here