```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time
#download the treebank corpus from nltk
nltk.download('treebank')
#download the universal tagset from nltk
nltk.download('universal_tagset')
# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
#print the first two sentences along with tags
print(nltk_data[:2])
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'
```

```
#print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
  for tuple in sent:
    print(tuple)
```

```
('Pierre', 'NOUN')
('Vinken', 'NOUN')
(',', '.')
('61', 'NUM')
('years', 'NOUN')
('old', 'ADJ')
(',', '.')
('will', 'VERB')
('join', 'VERB')
('the', 'DET')
('board', 'NOUN')
('as', 'ADP')
('a', 'DET')
('nonexecutive', 'ADJ')
('director', 'NOUN')
('Nov.', 'NOUN')
('29', 'NUM')
('.', '.')
('Mr.', 'NOUN')
('Vinken', 'NOUN')
('is', 'VERB')
('chairman', 'NOUN')
('of', 'ADP')
('Elsevier', 'NOUN')
('N.V.', 'NOUN')
(',', '.')
('the', 'DET')
('Dutch', 'NOUN')
('publishing', 'VERB')
('group', 'NOUN')
('.', '.')
```

```
# split data into training and validation set in the ratio 80:20
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))
```

```
80310
20366
```

```
train_tagged_words[:5]
```

```
[('Drink', 'NOUN'),
 ('Carrier', 'NOUN'),
 ('Competes', 'VERB'),
 ('With', 'ADP'),
 ('Cartons', 'NOUN')]
```

```
tags = {tag for word, tag in train_tagged_words}
print(len(tags))
print(tags)
```

```
vocab= {word for word , tag in train_tagged_words}
```

```
12
{'.', 'PRT', 'ADP', 'PRON', 'NOUN', 'ADV', 'X', 'CONJ', 'VERB', 'DET', 'NUM', 'ADJ'}
```

```
#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]
#now calculate the total number of times the passed word occurred as the passed tag.
  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)


# creating t x t transition matrix of tags, t= no of tags
# Matrix(i, j) represents P(jth tag after the ith tag)
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)
```

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

```
tags_df= pd.DataFrame(tags_matrix, columns = list(tags), index= list(tags))
display(tags_df)
```

	.	PRT	ADP	PRON	NOUN	ADV	X	CONJ	VERB	DET	NUM	ADJ
.	0.092372	0.002789	0.092908	0.068769	0.218539	0.052569	0.025641	0.060079	0.089690	0.172192	0.078210	0.046132
PRT	0.045010	0.001174	0.019569	0.017613	0.250489	0.009393	0.012133	0.002348	0.401174	0.101370	0.056751	0.082975
ADP	0.038724	0.001266	0.016958	0.069603	0.323589	0.014553	0.034548	0.001012	0.008479	0.320931	0.063275	0.107062
PRON	0.041913	0.014123	0.022323	0.006834	0.212756	0.036902	0.088383	0.005011	0.484738	0.009567	0.006834	0.070615
NOUN	0.240094	0.043935	0.176827	0.004659	0.262344	0.016895	0.028825	0.042454	0.149134	0.013106	0.009144	0.012584
ADJ	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
def Viterbi (words, train_bag = train_tagged_words):
  state = []
  T = list(set([pair[1] for pair in train_bag]))
  for key, word in enumerate(words):
    #initialise list of probability column for a given observation
    p = []
    for tag in T:
      if key == 0:
        transition_p = tags_df.loc['.', tag]
      else:
        transition_p = tags_df.loc[state[-1], tag]
      # compute emission and state probabilities
      emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
      state_probability = emission_p* transition_p
      p.append(state_probability)

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


#Let's test our Viterbi algorithm on a few sample sentences of test dataset
random.seed(1234) #define a random seed to get same sentences when run multiple times
#choose random 10 numbers
rndom = [random.randint(1, len(test_set)) for x in range(10)]
# list of 10 sents on which we test the model
test_run = [test_set[i] for i in rndom]
# list of tagged words
test_run_base = [tup for sent in test_run for tup in sent]
# list of untagged words
test_tagged_words = [tup[0] for sent in test_run for tup in sent]


#Here We will only test 10 sentences to check the accuracy
#as testing the whole training set takes huge amount of time
start = time.time()
tagged_seq= Viterbi (test_tagged_words)
end = time.time()
difference = end-start
print("Time taken in seconds: ", difference)
# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
accuracy = len(check)/len (tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  50.37351584434509
Viterbi Algorithm Accuracy:  93.77990430622009
```