Branch: master ▾ Find file Copy path

HMM_POS_Tagging / HMM_based_POS_tagging-applying Viterbi Algorithm.ipynb

saneshashank ok

fd2b30a on Aug 4, 2018

1 contributor

Raw Blame History

1869 lines (1869 sloc) 58.8 KB

POS tagging using modified Viterbi Algorithm:

In this project we apply techniques to improve Vanilla Viterbi Algorithm. Following are the steps followed:

1. plain vanilla Viterbi Algorithm is developed.
2. Viterbi Modification-Technique I - approach:
   - Transition probability is considered in case of unknown words.
   - The above approach is modified to consider transition probability weighted by tag occurrence probability in training set.
3. Viterbi Modification-Technique II - approach:
   - backoff to a rule based tagger in case of an unknown word.
   - The above technique is modified by using transition probability in approach 2 above if rule based tagger returns default noun tag ('NN').
4. The modified Viterbi algorithms are first tested on sampled test data for comparison.
5. The final algorithm and vanilla Viterbi Algorithm are the tested on full testing data for comparison.

Data Preparation

In [320]:
```
# 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
```

In [321]:
```
# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
```

In [322]:
```
# let's check some of the tagged data
print(nltk_data[1:5])
```

```
[[('Mr.', 'NOUN'), ('Vinken', 'NOUN'), ('is', 'VERB'), ('chairman', 'NOUN'), ('of', 'ADP'), ('Elsevier',
'NOUN'), ('N.V.', 'NOUN'), (',', '.'), ('the', 'DET'), ('Dutch', 'NOUN'), ('publishing', 'VERB'), ('grou
p', 'NOUN'), ('.', '.')], [('Rudolph', 'NOUN'), ('Agnew', 'NOUN'), (',', '.'), ('55', 'NUM'), ('years', 'N
OUN'), ('old', 'ADJ'), ('and', 'CONJ'), ('former', 'ADJ'), ('chairman', 'NOUN'), ('of', 'ADP'), ('Consolid
ated', 'NOUN'), ('Gold', 'NOUN'), ('Fields', 'NOUN'), ('PLC', 'NOUN'), (',', '.'), ('was', 'VERB'), ('name
d', 'VERB'), ('*-1', 'X'), ('a', 'DET'), ('nonexecutive', 'ADJ'), ('director', 'NOUN'), ('of', 'ADP'), ('t
his', 'DET'), ('British', 'ADJ'), ('industrial', 'ADJ'), ('conglomerate', 'NOUN'), ('.', '.')], [('A', 'DE
T'), ('form', 'NOUN'), ('of', 'ADP'), ('asbestos', 'NOUN'), ('once', 'ADV'), ('used', 'VERB'), ('*', 'X'),
('*', 'X'), ('to', 'PRT'), ('make', 'VERB'), ('Kent', 'NOUN'), ('cigarette', 'NOUN'), ('filters', 'NOUN'),
('has', 'VERB'), ('caused', 'VERB'), ('a', 'DET'), ('high', 'ADJ'), ('percentage', 'NOUN'), ('of', 'ADP'),
('cancer', 'NOUN'), ('deaths', 'NOUN'), ('among', 'ADP'), ('a', 'DET'), ('group', 'NOUN'), ('of', 'ADP'),
('workers', 'NOUN'), ('exposed', 'VERB'), ('*', 'X'), ('to', 'PRT'), ('it', 'PRON'), ('more', 'ADV'), ('th
an', 'ADP'), ('30', 'NUM'), ('years', 'NOUN'), ('ago', 'ADP'), (',', '.'), ('researchers', 'NOUN'), ('repo
rted', 'VERB'), ('0', 'X'), ('*T*-1', 'X'), ('.', '.')], [('The', 'DET'), ('asbestos', 'NOUN'), ('fiber',
'NOUN'), (',', '.'), ('crocidolite', 'NOUN'), (',', '.'), ('is', 'VERB'), ('unusually', 'ADV'), ('resilien
t', 'ADJ'), ('once', 'ADP'), ('it', 'PRON'), ('enters', 'VERB'), ('the', 'DET'), ('lungs', 'NOUN'), (',',
'.'), ('with', 'ADP'), ('even', 'ADV'), ('brief', 'ADJ'), ('exposures', 'NOUN'), ('to', 'PRT'), ('it', 'PR
ON'), ('causing', 'VERB'), ('symptoms', 'NOUN'), ('that', 'DET'), ('*T*-1', 'X'), ('show', 'VERB'), ('up',
'PRT'), ('decades', 'NOUN'), ('later', 'ADJ'), (',', '.'), ('researchers', 'NOUN'), ('said', 'VERB'),
('0', 'X'), ('*T*-2', 'X'), ('.', '.')]]
```

In [323]:
```
# split data into training and validation set in the ratio 95:5

train_set,test_set = train_test_split(nltk_data,train_size=0.95,test_size=0.05,random_state = 101)
```

In [324]:
```
# 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[0] for sent in test_set for tup in sent]
print(len(train_tagged_words))
print(len(test_tagged_words))
```

```
95547
5129
```

In [325]:
```
# check some of the tagged words.
train_tagged_words[1:5]
```

Out[325]:
```
[('confirmed', 'VERB'), ('the', 'DET'), ('filing', 'NOUN'), ('but', 'CONJ')]
```

In [326]:
```
# let's check how many unique tags are present in training data
tags = {tag for word,tag in train_tagged_words}
print(len(tags))
print(tags)
```

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

In [327]:
```
# let's check how many words are present in vocabulary
vocab = {word for word,tag in train_tagged_words}
```

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

```
12100
```

POS Tagging algorithm using Hidden Markov Model (HMM)

We'll use the HMM algorithm to tag the words. Given a sequence of words to be tagged, the task is to assign the most probable tag to the word. In other words, to every **word w**, assign **the tag t** that maximises the likelihood **P(t/w).**

Since P(t/w) = P(w/t). P(t) / P(w), after ignoring P(w), we have to compute P(w/t) and P(t).

Now:

- **P(w/t): is the emission probability** of a given word for a given tag. This can be computed based on the fraction of given word for given tag to the total count of that tag, ie: P(w/t) = count(w, t) / count(t).
- **P(t): is the probability of tag (also transition probability)**, and in a tagging task, we assume that a tag will depend only on the previous tag (Markov order 1 assumption). In other words, the probability of say a tag being NN will depend only on the previous tag t(n-1). So for e.g. if t(n-1) is a JJ, then t(n) is likely to be an NN since adjectives often precede a noun (blue coat, tall building etc.).

Build the Vanilla Viterbi based POS tagger

Function to compute emission probabilties for a given word

In [328]:
```
# compute emission probability for a given word for a given tag
def word_given_tag(word,tag,train_bag= train_tagged_words):
    taglist = [pair for pair in train_bag if pair[1] == tag]
    tag_count = len(taglist)
    w_in_tag = [pair[0] for pair in taglist if pair[0]==word]
    word_count_given_tag = len(w_in_tag)

    return (word_count_given_tag,tag_count)
```

Function to compute transition probabilties for a given tag and previous tag

In [329]:
```
# compute transition probabilities of a previous and next tag
def t2_given_t1(t2,t1,train_bag=train_tagged_words):
    tags = [pair[1] for pair in train_bag]

    t1_tags = [tag for tag in tags if tag==t1]

    count_of_t1 = len(t1_tags)

    t2_given_t1 = [tags[index+1] for index in range(len(tags)-1) if tags[index] == t1 and tags[index+1] ==
t2]

    count_t2_given_t1 = len(t2_given_t1)

    return(count_t2_given_t1,count_of_t1)
```

In [330]:
```
t2_given_t1('NOUN','DET')
```

Out[330]:
```
(5284, 8281)
```

In [331]:
```
# creating t x t transition matrix of tags
# each column is t2, each row is t1
# thus M(i, j) represents P(tj given ti)

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]
```

In [332]:
```
# convert the matrix to a df for better readability
tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))
```

In [333]:
```
tags_df
```

Out[333]:

	NUM	ADP	ADV	.	NOUN	ADJ	PRT	PRON	DET	VERB	CONJ	X
NUM	0.184932	0.036033	0.002978	0.117332	0.350208	0.034247	0.026504	0.001489	0.003276	0.018761	0.013699	0.21
ADP	0.062226	0.016893	0.014006	0.039025	0.320967	0.107024	0.001390	0.070031	0.324709	0.008340	0.000962	0.03
ADV	0.030474	0.118582	0.080490	0.137131	0.031467	0.129182	0.014243	0.014906	0.069891	0.343491	0.006956	0.02
.	0.081003	0.091342	0.052324	0.093320	0.222242	0.043963	0.002427	0.066349	0.173335	0.089095	0.057538	0.02
NOUN	0.009542	0.176514	0.017074	0.240604	0.263564	0.012248	0.043397	0.004607	0.012942	0.147667	0.042666	0.02
ADJ	0.021256	0.078267	0.004778	0.063931	0.690621	0.066403	0.010710	0.000330	0.004943	0.011609	0.016971	0.02

ADJ	0.021256	0.078267	0.004778	0.063931	0.699621	0.066403	0.010710	0.000330	0.004943	0.011699	0.016971	0.02
PRT	0.056672	0.020099	0.010214	0.043822	0.247776	0.083031	0.001647	0.017792	0.097858	0.405272	0.002306	0.01
PRON	0.006508	0.022971	0.034074	0.040965	0.210949	0.073124	0.013017	0.007657	0.009954	0.485452	0.005360	0.08
DET	0.022220	0.009540	0.012438	0.017993	0.638087	0.204323	0.000242	0.003744	0.005676	0.039850	0.000483	0.04
VERB	0.022851	0.092022	0.081952	0.034934	0.110070	0.064988	0.030674	0.035786	0.134392	0.169249	0.005577	0.21
CONJ	0.039981	0.052534	0.055323	0.034868	0.349140	0.118085	0.004649	0.058113	0.121339	0.156671	0.000465	0.00
X	0.002864	0.142584	0.024984	0.163590	0.062381	0.017187	0.185232	0.055538	0.054742	0.203851	0.010662	0.07

Viterbi Algorithm

The steps are as follows:

1. Given a sequence of words
2. iterate through the sequence
3. for each word (starting from first word in sequence) calculate the product of emission probabilties and transition probabilties for all possible tags.
4. assign the tag which has maximum probability obtained in step 3 above.
5. move to the next word in sequence to repeat steps 3 and 4 above.

In [334]:
```
# Vanilla Viterbi Algorithm
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))
```

Testing Vanilla Viterbi Algorithm on sampled test data

In [461]:
```
# Let's test our Viterbi algorithm on a few sample sentences of test dataset

random.seed(1234)

# 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_run for tup in sent]

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

In [418]:
```
# tagging the test sentences
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('Vanilla Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  39.22344136238098
Vanilla Viterbi Algorithm Accuracy:  89.38053097345133
```

In [419]:
```
# let's check the incorrectly tagged words
[j for i, j in enumerate(zip(tagged_seq, test_run_base)) if j[0] != j[1]]
```

Out[419]:
```
[(('Contra', 'NUM'), ('Contra', 'NOUN')),
 (('command', 'VERB'), ('command', 'NOUN')),
 (('Honduras', 'NUM'), ('Honduras', 'NOUN')),
 (('Sandinista', 'NUM'), ('Sandinista', 'NOUN')),
 (('offensive', 'NUM'), ('offensive', 'NOUN')),
 (('rebel', 'NUM'), ('rebel', 'NOUN')),
 (('Bucking', 'NUM'), ('Bucking', 'VERB')),
 (('drew', 'NUM'), ('drew', 'VERB')),
 (('Eveready', 'NUM'), ('Eveready', 'NOUN')),
 (('*T*-252', 'NUM'), ('*T*-252', 'X')),
 (('complaining', 'NUM'), ('complaining', 'VERB')),
 (('up', 'ADV'), ('up', 'PRT'))]
```

Solve the problem of unknown words

We can see that all of unknown words have been tagged as 'NUM' as 'NUM' is the first tag in tag list and is assigned if unknown word is encountered (emission probability =0).

Viterbi Modification-Technique I

- **First solution for unknown words**: assign based on transition probabilities only in case of unknown words as emission probability for unknown word is zero.

In [421]:
```
# use transition probability of tags when emission probability is zero (in case of unknown words)

def Viterbi_1(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 = []
        p_transition =[] # list for storing transition probabilities
        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)
            p_transition.append(transition_p)

        pmax = max(p)
        state_max = T[p.index(pmax)]


        # if probability is zero (unknown word) then use transition probability
        if(pmax==0):
            pmax = max(p_transition)
            state_max = T[p_transition.index(pmax)]

        else:
            state_max = T[p.index(pmax)]

        state.append(state_max)
    return list(zip(words, state))
```

In [422]:
```
# tagging the test sentences
start = time.time()
tagged_seq = Viterbi_1(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('Modified Viterbi_1 Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  37.800609827041626
Modified Viterbi_1 Accuracy:  94.69026548672566
```

Adding Tag occurance probability weights: we will apply weights based on the probability of tag occurance to the transition probabilities of tags and then use the resulting probability for predicting unknown words

and then use the resulting probability for predicting unknown words.

This scheme will also take into account that some POS tags are more likely to occur as compared to others.

In [342]:
```
# lets create a list containing tuples of POS tags and POS tag occurance probability, based on training da
ta
tag_prob = []
total_tag = len([tag for word,tag in train_tagged_words])
for t in tags:
    each_tag = [tag for word,tag in train_tagged_words if tag==t]
    tag_prob.append((t,len(each_tag)/total_tag))

tag_prob
```

Out[342]:
```
[('NUM', 0.035145007169246546),
 ('ADP', 0.0978889970381069),
 ('ADV', 0.031597015081582885),
 ('.', 0.11641391147812072),
 ('NOUN', 0.2862674913916711),
 ('ADJ', 0.06351847781719991),
 ('PRT', 0.03176447193527793),
 ('PRON', 0.0273373313657153),
 ('DET', 0.08666938784053921),
 ('VERB', 0.1351167488251855),
 ('CONJ', 0.02251248076862696),
 ('X', 0.06576867928872701)]
```

In [423]:
```
def Viterbi_1(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 = []
        p_transition =[] # list for storing transition probabilities

        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)

            # find POS tag occurance probability
            tag_p = [pair[1] for pair in tag_prob if pair[0]==tag ]

            # calculate the transition prob weighted by tag occurance probability.
            transition_p = tag_p[0]*transition_p
            p_transition.append(transition_p)

        pmax = max(p)
        state_max = T[p.index(pmax)]


        # if probability is zero (unknown word) then use weighted transition probability
        if(pmax==0):
            pmax = max(p_transition)
            state_max = T[p_transition.index(pmax)]

        else:
            state_max = T[p.index(pmax)]

        state.append(state_max)
    return list(zip(words, state))
```

In [424]:
```
# tagging the test sentences
start = time.time()
tagged_seq = Viterbi_2(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('Modified Viterbi_1 Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  38.04473161697388
Modified Viterbi_1 Accuracy:  95.57522123893806
```

Thus, we see that we have got a much better accuracy by using weighted transition probabilties.

In [425]:
```
# let's check the incorrectly tagged words
[j for i, j in enumerate(zip(tagged_seq, test_run_base)) if j[0] != j[1]]
```

Out[425]:
```
[(('command', 'VERB'), ('command', 'NOUN')),
 (('Sandinista', 'VERB'), ('Sandinista', 'NOUN')),
 (('Eveready', 'VERB'), ('Eveready', 'NOUN')),
 (('*T*-252', 'VERB'), ('*T*-252', 'X')),
 (('up', 'ADV'), ('up', 'PRT'))]
```

The following list of words have been **correctly POS tagged by Viterbi_1** as compared to vanilla Viterbi Algorithm:

- Contra:correctly tagged as NOUN
- Honduras:correctly tagged as NOUN
- complaining: correctly tagged as VERB
- Bucking: correctly tagged as VERB

Viterbi Modification-Technique II

second solution for unknown words:

- backoff to rule based tagger in case of unknown words.
- we further observe that POS tag 'X' can be easily encapsulated in regex rule, so we extract it only based on ruled based tagged.

Let's define a rule based tagger as below:

In [462]:
```
# specify patterns for tagging
patterns = [
    (r'.*ing$', 'VERB'),              # gerund
    (r'.*ed$', 'VERB'),               # past tense
    (r'.*es$', 'VERB'),               # verb
    (r'.*\'s$', 'NOUN'),              # possessive nouns
    (r'.*s$', 'NOUN'),                # plural nouns
    (r'\*T?\*?-[0-9]+$', 'X'),        # X
    (r'^-?[0-9]+(.[0-9]+)?$', 'NUM'), # cardinal numbers
    (r'.*', 'NOUN')                   # nouns
]

# rule based tagger
rule_based_tagger = nltk.RegexpTagger(patterns)
```

In [463]:
```
# Modification in Viterbi Algorithm : Backoff to rule based tagger in case unknown word is encountered.
def Viterbi_2(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)
        state_max = rule_based_tagger.tag([word])[0][1]


        if(pmax==0):
            state_max = rule_based_tagger.tag([word])[0][1] # assign based on rule based tagger
        else:
            if state_max != 'X':
                # getting state for which probability is maximum
                state_max = T[p.index(pmax)]


        state.append(state_max)
    return list(zip(words, state))
```

In [465]:
```
# tagging the test sentences
start = time.time()
tagged_seq = Viterbi_2(test_tagged_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)
```

```
Time taken in seconds:  34.20582127571106
```

In [466]:
```
# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
accuracy = len(check)/len(tagged_seq)
print('Modified Viterbi_2 Accuracy: ',accuracy*100)
```

```
Modified Viterbi_2 Accuracy:  97.34513274336283
```

In [430]:
```
# let's check the incorrectly tagged words
[j for i, j in enumerate(zip(tagged_seq, test_run_base)) if j[0] != j[1]]
```

Out[430]:
```
[(('command', 'VERB'), ('command', 'NOUN')),
 (('drew', 'NOUN'), ('drew', 'VERB')),
 (('up', 'ADV'), ('up', 'PRT'))]
```

The following list of words have been correctly POS tagged by Viterbi_2 as compared to vanilla Viterbi Algorithm:

- Contra:correctly tagged as NOUN
- Honduras:correctly tagged as NOUN
- complaining: correctly tagged as VERB
- Bucking: correctly tagged as VERB

the following list of words has been correctly tagged by Viterbi_2 as compared to Viterbi_1

- Sandinista: correctly tagged as NOUN
- Eveready: correctly tagged as NOUN
- *T*-252: correctly tagged as 'X'

further modification in Viterb_2: We know that the rule based tagger assigns 'NOUN' by default if word does not fall in any rule, to correct this let's assign the tags for any such word based purely on transition probability of tags.

So, first we will modify the rule based tagger to output 'NN' instead of 'NOUN' in case word does not satisfy any rules. We also observe that any capitalized word can still be defaulted as 'NOUN' so will add one more rule for that case.

In [432]:
```
# specify patterns for tagging
patterns = [
    (r'.*ing$', 'VERB'),              # gerund
    (r'.*ed$', 'VERB'),               # past tense
    (r'.*es$', 'VERB'),               # verb
    (r'.*\'s$', 'NOUN'),              # possessive nouns
    (r'.*s$', 'NOUN'),                # plural nouns
    (r'\*T?\*?-[0-9]+$', 'X'),        # X
    (r'^-?[0-9]+(.[0-9]+)?$', 'NUM'), # cardinal numbers
    (r'^[A-Z][a-z].*', 'NOUN'),       # NOUN
    (r'.*', 'NN')                     # default
]

# rule based tagger
rule_based_tagger = nltk.RegexpTagger(patterns)
```

In [437]:
```
# modified Viterbi
def Viterbi_2(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 = []
        p_transition =[] # for storing transition probabilities
        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)

            # find POS tag occurance probability
            tag_p = [pair[1] for pair in tag_prob if pair[0]==tag ]

            # calculate the transition prob weighted by tag occurance probability.
            transition_p = tag_p[0]*transition_p
            p_transition.append(transition_p)

        pmax = max(p)
        state_max = rule_based_tagger.tag([word])[0][1]


        # getting state for which probability is maximum
```

```
        if(pmax==0):
            state_max = rule_based_tagger.tag([word])[0][1] # assign based on rule based tagger

            # if unknown word does not satisfy any rule, find the tag with maximum transition probability
            if state_max == 'NN':
                pmax = max(p_transition)
                state_max = T[p_transition.index(pmax)]

        else:
             if state_max != 'X':
                # getting state for which probability is maximum
                state_max = T[p.index(pmax)]

        state.append(state_max)
    return list(zip(words, state))
```

In [438]:
```
# tagging the test sentences
start = time.time()
tagged_seq = Viterbi_2(test_tagged_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)
```

```
Time taken in seconds:  38.43364071846008
```

In [439]:
```
# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
accuracy = len(check)/len(tagged_seq)
print('Modified Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Modified Viterbi Algorithm Accuracy:  98.23008849557522
```

We observe that much better accuracy is obtained now.

In [440]:
```
# let's check the incorrectly tagged words
[j for i, j in enumerate(zip(tagged_seq, test_run_base)) if j[0] != j[1]]
```

Out[440]:
```
[(('command', 'VERB'), ('command', 'NOUN')), (('up', 'ADV'), ('up', 'PRT'))]
```

The following list of words have been correctly POS tagged by Viterbi_2 as compared to vanilla Viterbi Algorithm:

- Contra:correctly tagged as NOUN
- Honduras:correctly tagged as NOUN
- complaining: correctly tagged as VERB
- Bucking: correctly tagged as VERB

the following list of words has been correctly tagged by Viterbi_2 as compared to Viterbi_1

- Sandinista: correctly tagged as NOUN
- Eveready: correctly tagged as NOUN
- `*T*-252`: correctly tagged as 'X'
- drew: correctly tagged as VERB

Evaluate vanilla Viterbi and modified Viterbi on entire test data

In [458]:
```
test_tagged_words = [tup[0] for sent in test_set for tup in sent]
test_run_base = [tup for sent in test_set for tup in sent]
```

In [459]:
```
# tagging the test sentences
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('Modified Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  1647.6785893440247
Modified Viterbi Algorithm Accuracy:  91.51881458373951
```

In [460]:
```
# tagging the test sentences
start = time.time()
tagged_seq = Viterbi_2(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('Modified Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  2186.4629657268524
Modified Viterbi Algorithm Accuracy:  95.43770715539091
```

Evaluating tagging on sample 'Test_sentences.txt' file

In [442]:
```
f = open('Test_sentences.txt')
```

In [443]:
```
text = f.read()
```

In [444]:
```
sample_test_sent = text.splitlines()
```

In [445]:
```
f.close()
```

In [446]:
```
sample_test_sent = test_sentences[:-3]
```

In [447]:
```
sample_test_sent
```

Out[447]:
```
['Android is a mobile operating system developed by Google.',
 'Android has been the best-selling OS worldwide on smartphones since 2011 and on tablets since 2013.',
 "Google and Twitter made a deal in 2015 that gave Google access to Twitter's firehose.",
 'Twitter is an online news and social networking service on which users post and interact with messages k
nown as tweets.',
 'Before entering politics, Donald Trump was a domineering businessman and a television personality.',
 'The 2018 FIFA World Cup is the 21st FIFA World Cup, an international football tournament contested once
every four years.',
 'This is the first World Cup to be held in Eastern Europe and the 11th time that it has been held in Euro
pe.',
 'Show me the cheapest round trips from Dallas to Atlanta']
```

In [448]:
```
# list of untagged words
sample_test_words = [word for sent in sample_test_sent for word in sent.split()]
```

In [449]:
```
# tagging the test sentences
start = time.time()
sample_tagged_seq = Viterbi(sample_test_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)
```

```
Time taken in seconds:  41.50299906730652
```

In [450]:
```
sample_tagged_seq
```

Out[450]:
```
[('Android', 'NUM'),
 ('is', 'VERB'),
 ('a', 'DET'),
 ('mobile', 'ADJ'),
 ('operating', 'NOUN'),
 ('system', 'NOUN'),
 ('developed', 'VERB'),
 ('by', 'ADP'),
 ('Google.', 'NUM'),
 ('Android', 'NUM'),
 ('has', 'VERB'),
 ('been', 'VERB'),
 ('the', 'DET'),
 ('best-selling', 'ADJ'),
 ('OS', 'NUM'),
 ('worldwide', 'NUM'),
 ('on', 'ADP'),
 ('smartphones', 'NUM'),
 ('since', 'ADP'),
 ('2011', 'NUM'),
 ('and', 'CONJ'),
 ('on', 'ADP'),
 ('tablets', 'NOUN'),
 ('since', 'ADP'),
 ('2013.', 'NUM'),
 ('Google', 'NUM'),
 ('and', 'CONJ'),
 ('Twitter', 'NUM'),
 ('made', 'VERB'),
 ('a', 'DET'),
 ('deal', 'NOUN'),
 ('in', 'ADP'),
 ('2015', 'NUM'),
 ('that', 'ADP'),
```

```
('gave', 'VERB'),
('Google', 'NUM'),
('access', 'NOUN'),
('to', 'PRT'),
("Twitter's", 'NUM'),
('firehose.', 'NUM'),
('Twitter', 'NUM'),
('is', 'VERB'),
('an', 'DET'),
('online', 'NUM'),
('news', 'NOUN'),
('and', 'CONJ'),
('social', 'ADJ'),
('networking', 'NOUN'),
('service', 'NOUN'),
('on', 'ADP'),
('which', 'DET'),
('users', 'NOUN'),
('post', 'NOUN'),
('and', 'CONJ'),
('interact', 'NUM'),
('with', 'ADP'),
('messages', 'NUM'),
('known', 'VERB'),
('as', 'ADP'),
('tweets.', 'NUM'),
('Before', 'ADP'),
('entering', 'VERB'),
('politics,', 'NUM'),
('Donald', 'NOUN'),
('Trump', 'NOUN'),
('was', 'VERB'),
('a', 'DET'),
('domineering', 'NUM'),
('businessman', 'NOUN'),
('and', 'CONJ'),
('a', 'DET'),
('television', 'NOUN'),
('personality.', 'NUM'),
('The', 'DET'),
('2018', 'NUM'),
('FIFA', 'NUM'),
('World', 'NOUN'),
('Cup', 'NUM'),
('is', 'VERB'),
('the', 'DET'),
('21st', 'NUM'),
('FIFA', 'NUM'),
('World', 'NOUN'),
('Cup,', 'NUM'),
('an', 'DET'),
('international', 'ADJ'),
('football', 'NOUN'),
('tournament', 'NUM'),
('contested', 'NUM'),
('once', 'ADV'),
('every', 'DET'),
('four', 'NUM'),
('years.', 'NUM'),
('This', 'DET'),
('is', 'VERB'),
('the', 'DET'),
('first', 'ADJ'),
('World', 'NOUN'),
('Cup', 'NUM'),
('to', 'PRT'),
('be', 'VERB'),
('held', 'VERB'),
('in', 'ADP'),
('Eastern', 'NOUN'),
('Europe', 'NOUN'),
('and', 'CONJ'),
('the', 'DET'),
('11th', 'ADJ'),
('time', 'NOUN'),
('that', 'ADP'),
('it', 'PRON'),
('has', 'VERB'),
('been', 'VERB'),
('held', 'VERB'),
('in', 'ADP'),
('Europe.', 'NUM'),
('Show', 'NOUN'),
('me', 'PRON'),
('the', 'DET'),
('cheapest', 'ADJ'),
('round', 'NOUN'),
```

```
('trips', 'NUM'),
('from', 'ADP'),
('Dallas', 'NOUN'),
('to', 'PRT'),
('Atlanta', 'NOUN')]
```

We can see that several words have been misclassified by vanilla Viterbi POS tagger, for example:

- Android as NUM
- Google as NUM
- OS as NUM

In [451]:
```
# tagging the test sentences
start = time.time()
sample_tagged_seq = Viterbi_2(sample_test_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)
```

```
Time taken in seconds:  40.423513650894165
```

In [452]:
```
sample_tagged_seq
```

Out[452]:
```
[('Android', 'NOUN'),
 ('is', 'VERB'),
 ('a', 'DET'),
 ('mobile', 'ADJ'),
 ('operating', 'NOUN'),
 ('system', 'NOUN'),
 ('developed', 'VERB'),
 ('by', 'ADP'),
 ('Google.', 'NOUN'),
 ('Android', 'NOUN'),
 ('has', 'VERB'),
 ('been', 'VERB'),
 ('the', 'DET'),
 ('best-selling', 'ADJ'),
 ('OS', 'NOUN'),
 ('worldwide', 'NOUN'),
 ('on', 'ADP'),
 ('smartphones', 'VERB'),
 ('since', 'ADP'),
 ('2011', 'NUM'),
 ('and', 'CONJ'),
 ('on', 'ADP'),
 ('tablets', 'NOUN'),
 ('since', 'ADP'),
 ('2013.', 'NOUN'),
 ('Google', 'NOUN'),
 ('and', 'CONJ'),
 ('Twitter', 'NOUN'),
 ('made', 'VERB'),
 ('a', 'DET'),
 ('deal', 'NOUN'),
 ('in', 'ADP'),
 ('2015', 'NUM'),
 ('that', 'ADP'),
 ('gave', 'VERB'),
 ('Google', 'NOUN'),
 ('access', 'NOUN'),
 ('to', 'PRT'),
 ("Twitter's", 'NOUN'),
 ('firehose.', 'NOUN'),
 ('Twitter', 'NOUN'),
 ('is', 'VERB'),
 ('an', 'DET'),
 ('online', 'NOUN'),
 ('news', 'NOUN'),
 ('and', 'CONJ'),
 ('social', 'ADJ'),
 ('networking', 'NOUN'),
 ('service', 'NOUN'),
 ('on', 'ADP'),
 ('which', 'DET'),
 ('users', 'NOUN'),
 ('post', 'NOUN'),
 ('and', 'CONJ'),
 ('interact', 'NOUN'),
 ('with', 'ADP'),
 ('messages', 'VERB'),
 ('known', 'VERB'),
 ('as', 'ADP'),
 ('tweets.', 'NOUN'),
 ('Before', 'ADP'),
 ('entering', 'VERB'),
```

```
('entering', 'VERB'),
('politics,', 'NOUN'),
('Donald', 'NOUN'),
('Trump', 'NOUN'),
('was', 'VERB'),
('a', 'DET'),
('domineering', 'VERB'),
('businessman', 'NOUN'),
('and', 'CONJ'),
('a', 'DET'),
('television', 'NOUN'),
('personality.', 'NOUN'),
('The', 'DET'),
('2018', 'NUM'),
('FIFA', 'NOUN'),
('World', 'NOUN'),
('Cup', 'NOUN'),
('is', 'VERB'),
('the', 'DET'),
('21st', 'NOUN'),
('FIFA', 'NOUN'),
('World', 'NOUN'),
('Cup,', 'NOUN'),
('an', 'DET'),
('international', 'ADJ'),
('football', 'NOUN'),
('tournament', 'NOUN'),
('contested', 'VERB'),
('once', 'ADV'),
('every', 'DET'),
('four', 'NUM'),
('years.', 'NOUN'),
('This', 'DET'),
('is', 'VERB'),
('the', 'DET'),
('first', 'ADJ'),
('World', 'NOUN'),
('Cup', 'NOUN'),
('to', 'PRT'),
('be', 'VERB'),
('held', 'VERB'),
('in', 'ADP'),
('Eastern', 'NOUN'),
('Europe', 'NOUN'),
('and', 'CONJ'),
('the', 'DET'),
('11th', 'ADJ'),
('time', 'NOUN'),
('that', 'ADP'),
('it', 'PRON'),
('has', 'VERB'),
('been', 'VERB'),
('held', 'VERB'),
('in', 'ADP'),
('Europe.', 'NOUN'),
('Show', 'NOUN'),
('me', 'PRON'),
('the', 'DET'),
('cheapest', 'ADJ'),
('round', 'NOUN'),
('trips', 'NOUN'),
('from', 'ADP'),
('Dallas', 'NOUN'),
('to', 'PRT'),
('Atlanta', 'NOUN')]
```

All these cases were correctly POS tagged by Viterbi_2:

- Android as NOUN
- Google as NOUN
- OS as NOUN