POS Tagging using HMM

Name- Vedant Dawange

Roll No. - 82

ML Batch 2

```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
import random
nltk.download('treebank')
nltk_data = list(nltk.corpus.treebank.tagged_sents())
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
```

```
from sklearn.model_selection import train_test_split


# let's check some of the tagged data
print(nltk_data[0])
```

```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), (
```

```
# split data into training and validation set in the ratio 95:5
train_set,test_set = train_test_split(nltk_data,train_size=0.8,test_size=0.2,random_state

# 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))

# check some of the tagged words.
print(train_tagged_words[0:5])

# checking how many unique tags are present in training data
tags = {tag for word,tag in train_tagged_words}
print(len(tags))
print(tags)

# checking how many words are present in vocabulary
vocab = {word for word,tag in train_tagged_words}
print(len(vocab))

# 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)

# 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

    count_t2_given_t1 = len(t2_given_t1)

    return(count_t2_given_t1,count_of_t1)

t2_given_t1('NOUN','DET')

# 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]

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

```
 79930
 20746
 [('Edward', 'NNP'), ('L.', 'NNP'), ('Kane', 'NNP'), ('succeeded', 'VBD'), ('Mr.', 'NN
 46
 {'PRP$', 'VBP', 'RBR', 'IN', 'VBZ', '-NONE-', 'VBG', "''", 'PRP', 'RBS', 'NNS', 'VBD
 10958
```

```
tags_df
```

```
#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
            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 Viterbi algorithm on a few sample sentences of test dataset

random.seed(1234)

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

# list of sentences
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]


# tagging the test sentences
tagged_seq = Viterbi(test_tagged_words)

# 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)

     Viterbi Algorithm Accuracy:  88.59649122807018


# checking the incorrectly tagged words
[j for i, j in enumerate(zip(tagged_seq, test_run_base)) if j[0] != j[1]]

     [(('broke', 'PRP$'), ('broke', 'VBD')),
      (('cranked', 'PRP$'), ('cranked', 'VBD')),
      (('up', 'IN'), ('up', 'RP')),
      (('worthy', 'PRP$'), ('worthy', 'JJ')),
      (('murder', 'PRP$'), ('murder', 'NN')),
      (('incest', 'PRP$'), ('incest', 'NN')),
      (('dynamics', 'PRP$'), ('dynamics', 'NNS')),
```

```
(('transforming', 'PRP$'), ('transforming', 'VBG')),
(('it', 'PRP$'), ('it', 'PRP')),
(('packaging', 'VBG'), ('packaging', 'NN')),
(('130.6', 'PRP$'), ('130.6', 'CD')),
(('Hiroshi', 'PRP$'), ('Hiroshi', 'NNP')),
(('Asada', 'PRP$'), ('Asada', 'NNP'))]
```