Hidden Markov Model

Abdul Khader, Syed

01 November 2022

```
# Importing libraries
import nltk
import time
import random
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# 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])
```

```
[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.
```

```
# print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
    print('==============')
    print(*sent, sep="\n")
```

```
==============
('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
```

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

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

```
# use set datatype to check how many unique tags are present in training data
tags = {tag for word, tag in train_tagged_words}
print('Number of tags:', len(tags))
print(tags)

# check total words in vocabulary
vocab = {word for word, tag in train_tagged_words}
print('Number of words in vocabulary:', len(vocab))

# creating the order of words based on alphabetical order
word_order = {word:idx for idx,word, in enumerate(sorted(list(vocab)))}

# creating the order of tags based on alphabetical order
tags_list = sorted(list(tags))
tag_order = {tag:idx for idx,tag, in enumerate(tags_list)}
```

```
Number of tags: 12
{'ADP', 'DET', 'PRON', 'ADV', 'NOUN', '.', 'CONJ', 'VERB', 'NUM', 'ADJ', 'X', 'PRT'}
Number of words in vocabulary: 11052
```

```
def emission_prob(vocab, tags, word_order=word_order, train_bag = train_tagged_words):
    B = {t:np.zeros(len(vocab)) for t in tags}

    for w,t in train_bag:
        B[t][word_order[w]] += 1
```

```
    for key in B:
        B[key] = B[key]/np.sum(B[key])

    return B

def state_prob(tags, word_order=word_order, tag_order=tag_order, train_bag = train_tagged_words):
    A = np.zeros((len(tags),len(tags)))

    for i in range(len(train_tagged_words)-1):
        # Getting the Tag of a Word
        w1 = train_tagged_words[i][1]
        # Getting the Tag of the subsequent Word
        w2 = train_tagged_words[i+1][1]
        A[tag_order[w1], tag_order[w2]] += 1

    return A/A.sum(axis=1, keepdims=True)

def state_init_prob(tags, tag_order=tag_order, train_bag = train_tagged_words):
    pi = np.zeros(len(tags))
    for w,t in train_bag:
        pi[tag_order[t]] += 1

    return pi/pi.sum()
```

```
# compute Emission Probability
B = emission_prob(vocab, tags, word_order, train_tagged_words)
def word_given_tag(word, tag, word_order=word_order):
    return B[tag][word_order[word]] #or 1e-4



# compute Transition Probability
A = state_prob(tags, word_order, tag_order, train_tagged_words)
def t2_given_t1(t2, t1):
    return A[t1,t2]

pi = state_init_prob(tags, tag_order, train_tagged_words)
```

```
# you can also convert the matrix to a pandas dataframe for better readability
pd.DataFrame(A, index=tag_order.keys(),columns=tag_order.keys())
```

	.	ADJ	ADP	ADV	CONJ	DET	NOUN	NUM	PRON	PRT	VERB
.	0.092382	0.046137	0.092918	0.052575	0.060086	0.172210	0.218562	0.078219	0.068777	0.002790	0.089700
ADJ	0.066019	0.063301	0.080583	0.005243	0.016893	0.005243	0.696893	0.021748	0.000194	0.011456	0.011456
ADP	0.038724	0.107062	0.016958	0.014553	0.001012	0.320931	0.323589	0.063275	0.069603	0.001266	0.008479
ADV	0.139255	0.130721	0.119472	0.081458	0.006982	0.071373	0.032196	0.029868	0.012025	0.014740	0.339022
CONJ	0.035126	0.113611	0.055982	0.057080	0.000549	0.123491	0.349067	0.040615	0.060373	0.004391	0.150384
DET	0.017393	0.206411	0.009918	0.012074	0.000431	0.006037	0.635906	0.022855	0.003306	0.000287	0.040247
NOUN	0.240094	0.012584	0.176827	0.016895	0.042454	0.013106	0.262344	0.009144	0.004659	0.043935	0.149134
NUM	0.119243	0.035345	0.037487	0.003570	0.014281	0.003570	0.351660	0.184220	0.001428	0.026062	0.020707
PRON	0.041913	0.070615	0.022323	0.036902	0.005011	0.009567	0.212756	0.006834	0.006834	0.014123	0.484738

	.	ADJ	ADP	ADV	CONJ	DET	NOUN	NUM	PRON	PRT	VERB
PRT	0.045010	0.082975	0.019569	0.009393	0.002348	0.101370	0.250489	0.056751	0.017613	0.001174	0.401174
VERB	0.034807	0.066390	0.092357	0.083886	0.005433	0.133610	0.110589	0.022836	0.035543	0.030663	0.167956
X	0.160869	0.017682	0.142226	0.025754	0.010379	0.056890	0.061695	0.003075	0.054200	0.185086	0.206419

```
def Viterbi(words, train_bag = train_tagged_words):
    bk_track = np.zeros((len(words), len(tags)))

    # Probabilities at time T=1
    curr_layer = np.array([
        pi[idx]*word_given_tag(words[0], tag) for idx, tag in enumerate(tags_list)
    ])
    # curr_layer = (curr_layer)/(curr_layer).sum()
    next_layer = np.zeros(len(tags))

    # Loop over all the words
    for idx, word in enumerate(words[1:], start=1):
        # Looping over all states in next time step
        for j in range(len(next_layer)):
            temp = np.zeros(len(tags))
            wgt = word_given_tag(word, tags_list[j]) if word in word_order else 1
            # Looping over all current layer(time) states
            # to determine values of a state in next layer
            for k in range(len(curr_layer)):
                # Probability of the new state, coming from a state from prev layer
                # and the word(observation) given new state.
                temp[k] = curr_layer[k] * t2_given_t1(k,j) * wgt
            next_layer[j] = temp.max()
            # Adding the state index coming from prev layer in back track matrix
            bk_track[idx, j] = temp.argmax()
        next_layer = next_layer/next_layer.sum()
        curr_layer = next_layer.copy()

    path = []
    pointer = curr_layer.argmax()
    path.append((words[-1], tags_list[pointer]))
    # Backtracking
    for row, word in zip(np.flipud(bk_track), words[-2::-1]):
        path.append( (word, tags_list[int(row[pointer])]) )
        pointer = int(row[pointer])

    return path[::-1]
```

Random 10 sentences Test Accuracy

```
# test the 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
np.random.seed(1234)

# choose random 10 numbers
```

```
rndom = [random.randint(1, len(test_set)) for x in range(10)]

# list of 10 sentencess on which to 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]

# testing 10 sentences to check the accuracy
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end - start

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

# accuracy should be good enough (> 90%) to be a satisfactory model
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: 0.06524825096130371
Viterbi Algorithm Accuracy: 92.82296650717703
```

Test Accuracy for the entire Test Set

```
# test the 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
np.random.seed(1234)

# 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]

# testing 10 sentences to check the accuracy
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end - start

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

# accuracy should be good enough (> 90%) to be a satisfactory model
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:', round(accuracy * 100))
```

```
Time taken in seconds: 3.193380355834961
Viterbi Algorithm Accuracy: 90
```

Hot Cold State Problem

Two Hidden States: **Hot** and **Cold**

Three Observations: **1**, **2**, and **3**

Probability of Sequence 3-1-3 Occuring?

Ans: **2.86%**

Most Probable States Sequence given the Observation 3-1-3

Ans: **Hot-Cold-Hold**