

# Machine Learning Homework Assignment

Name: Kaustubh Shivshankar Shejole

Enrollment Id: BT20CSE112

Topic: *Use of HMM in partitioning the alphabet set into 2 sets using text from Shakespeare's writings.*

Dataset source: <https://www.gutenberg.org/files/100/100-0.txt>

Possible Candidate Algorithms:

1. Hidden Markov Models
2. Dynamic Programming
3. Artificial Neural Network

Why HMM was preferred?

1. Hidden Markov Models are highly efficient to get trained given an observation sequence.
2. HMMs give more optimal answer than Dynamic Programming.
3. HMMs can be understood by humans unlike ANNs.
4. Data was sequential and hence I preferred to choose Hidden Markov Model over others.

Python Library used:

1. numpy (import numpy)
2. requests (import requests)
3. BeautifulSoup (from bs4)
4. re (import re)

Initial Model:

$\pi = [0.51316, 0.48684]$

$a = \begin{bmatrix} 0.47468, & 0.52532, \\ 0.51656, & 0.48344 \end{bmatrix}$

$b^T$ :

	class 1	class 2
a	0.03876	0.03813
b	0.03707	0.03684
c	0.03865	0.038
d	0.03614	0.03327
e	0.03889	0.03451
f	0.0355	0.03769
g	0.03847	0.04088
h	0.03673	0.03643
i	0.03432	0.03459
j	0.03396	0.04147
k	0.03818	0.03805
l	0.03765	0.03932
m	0.0356	0.03987
n	0.03527	0.03454
o	0.03681	0.03803
p	0.04026	0.0379
q	0.03637	0.03904
r	0.03272	0.03917
s	0.0362	0.03731
t	0.03534	0.03363
u	0.03971	0.03663
v	0.03624	0.0348
w	0.03679	0.03335
x	0.03907	0.03238
y	0.03532	0.03988
z	0.03782	0.03687
whitespace	0.04217	0.03741

Code:

```
b = [[0.03876073241828919,
0.03707088923577125,
0.0386474368230154,
0.03614128011739876,
0.038889996009833824,
0.035499096239621775,
0.03847097614177666,
0.03672944790202319,
0.034317225295089904,
0.0339572530502138,
0.038179441095574904,
0.037649737601381954,
0.035601204102646666,
0.03526471596395519,
0.03680774548791198,
0.04026013320803973,
```

```
0.03637354591216267,  
0.03272324169918124,  
0.03620429652155185,  
0.035340046856218554,  
0.03971053915510608,  
0.03623566423686421,  
0.036788097847819445,  
0.03907307328798778,  
0.035316988443362604,  
0.037816278618950976,  
0.042170916728250483],  
[0.03812821037687526,  
0.03684258634756626,  
0.038000100073947767,  
0.0332710750690177,  
0.034507994491999636,  
0.03768937106823203,  
0.04088384112004803,  
0.03642640052598796,  
0.03458909587313996,  
0.041471924856658196,  
0.03805112437591366,  
0.039320300639768026,  
0.0398711815219594,  
0.03454296198593447,  
0.038032840747423136,  
0.037902089336905025,  
0.03904358950995742,  
0.03917161351023882,  
0.03730802606048594,  
0.03362787678277796,  
0.03663024092078671,  
0.03480209646912969,  
0.03335292757944987,  
0.03237989270699796,  
0.03987473112621363,  
0.036872324073064976,  
0.03740558284952068]]  
#initialize  
pi = [0.51316, 0.48684]  
a = [[ 0.47468,0.52532],[0.51656,0.48344]]  
# b = [[0.32,0.33,0.35],[0.34,0.329,0.331]]  
#occurrence on which a model is to be trained  
# O = [2,1,2,0,1]  
T = len(O)  
N = len(a[0])  
M = len(b[0])  
old_logProb = -1000000000
```

```

new_logProb = -100000000
alpha = []
for _ in range(T):
    list1 = []
    for i in range(N):
        list1.append([])
    alpha.append(list1)
beta = []
for _ in range(T):
    list1 = []
    for i in range(N):
        list1.append([])
    beta.append(list1)
gamma_t_i_j = []
for t in range(T-1):
    list1 = []
    for i in range(N):
        list2 = []
        for j in range(N):
            list2.append([])
        list1.append(list2)
    gamma_t_i_j.append(list1)
gamma = []
for _ in range(T):
    list1 = []
    for i in range(N):
        list1.append(0)
    gamma.append(list1)
import numpy
#for epoch in range(200):
epoch = 0
# while (epoch <= 100 and (new_logProb - old_logProb) > 0.0001):
while (epoch <= 100 and (new_logProb > old_logProb)):
    print(f'after {epoch}')
    print(pi)
    print(a)
    print(b)

    c_values = [0 for i in range(T)]
    c_values[0] = 0
    for i in range(N):
        alpha[0][i] = pi[i]*b[i][0[0]]
        c_values[0] += alpha[0][i]

    # scale the alpha[0][i]
    c_values[0] = 1/c_values[0]
    for i in range(N):
        alpha[0][i] = c_values[0]*alpha[0][i]

```

```

for t in range(1,T):
    c_values[t] = 0
    for i in range(N):
        alpha[t][i] = 0
        for j in range(N):
            alpha[t][i] += alpha[t-1][j]*a[j][i]
        alpha[t][i] *= b[i][O[t]]
        c_values[t] = c_values[t] + alpha[t][i]

    #scale alpha[t][i]
    c_values[t] = 1/c_values[t]
    for i in range(N):
        alpha[t][i] = c_values[t]*alpha[t][i]

    # Let beta[T-1][i] = 1, scaled by c_values[T-1]
    for i in range(N):
        beta[T-1][i] = c_values[T-1]
    t = T-2
    while(t >= 0):
        for i in range(N):
            beta[t][i] = 0
            for j in range(N):
                beta[t][i] += a[i][j]*b[j][O[t+1]]*beta[t+1][j]
            beta[t][i] = c_values[t]*beta[t][i]
        t = t-1

    for t in range(T-1):
        for i in range(N):
            gamma[t][i] = 0
            for j in range(N):
                gamma_t_i_j[t][i][j] =
alpha[t][i]*a[i][j]*b[j][O[t+1]]*beta[t+1][j]
            gamma[t][i] = gamma[t][i] + gamma_t_i_j[t][i][j]
        for i in range(N):
            gamma[T-1][i] = alpha[T-1][i]

    #Re-estimate the model lambda
    for i in range(N):
        pi[i] = gamma[0][i]

    for i in range(N):
        for j in range(N):
            numer = 0
            denom = 0
            for t in range(T-1):
                numer += gamma_t_i_j[t][i][j]
                denom += gamma[t][i]

```

```

        a[i][j] = numer/denom
    for i in range(N):
        for j in range(M):
            numer = 0
            denom = 0
            for t in range(T):
                if O[t] == j:
                    numer = numer + gamma[t][i]
                    denom = denom + gamma[t][i]
            b[i][j] = numer/denom
    logProb = 0
    for i in range(T):
        logProb += numpy.log(c_values[i])
    logProb = -1 * logProb
    print(logProb)

    old_logProb = new_logProb
    new_logProb = logProb

epoch = epoch + 1

```

A. Used observation sequence as the following sentence in english.

'the quick brown fox jumps over the lazy dog and then rests peacefully'  
and made B with the observation states as 27, 26 lowercase alphabets with 1  
as whitespace.

Initialized B with each value approximately  $1/27$  ensuring it is row-  
stochastic.

Then from the sentence we get O as `[19, 7, 4, 26, 16, 20, 8, 2, 10, 26, 1, 17, 14, 22, 13, 26, 5, 14, 23, 26, 9, 20, 12, 15, 18, 26, 14, 21, 4, 17, 26, 19, 7, 4, 26, 11, 0, 25, 24, 26, 3, 14, 6, 26, 0, 13, 3, 26, 19, 7, 4, 13, 26, 17, 4, 18, 19, 18, 26, 15, 4, 0, 2, 4, 5, 20, 11, 11, 24]`

So we trained it similarly and got the following results:

after 41

$\pi = [1.0000000000000002, 8.356337684774142e-253]$

$A = [[3.982444907951401e-07, \quad 0.9999996017555091],$

[0.7981057649355989, 0.20189423506440118]]

Looking at A

[[3.982444907951401e-07, 0.9999996017555091],  
[0.7981057649355989, 0.20189423506440118]]

Gives us that there are 2 classes:

Such that after Class1 there is high probability approximately 0.9999 that Class2 will appear,

Similarly Class1 after Class2 : 0.798

Class2 after Class2: 0.2018

We can guess that Class1 is of Vowels and class2 is of consonants.

As observation sequence was small the results of b are not that much useful and some of the alphabets are missing.

# Results for $b^T$ matrix:

	vowel	consonant
a	0.096213	0
b	0.032071	0
c	0	0.052883
d	0.064142	0
e	0.178133	0.038226
f	0.041537	0.018637
g	0.032071	0
h	0	0.079325
i	0.032071	0
j	0.032071	0
k	0.032071	0
l	0.02925	0.055209
m	0	0.026442
n	0.03106	0.053717
o	0.059493	0.056716
p	0.064142	0
q	0.032071	0
r	0	0.079325
s	0	0.079325
t	0.128284	0
u	0.000001	0.079324
v	0	0.026442
w	0	0.026442
x	0.032071	0
y	0.064142	0
z	0	0.026442
whitespace	0.019108	0.301546

B. We now took first 7000 characters from

<https://www.gutenberg.org/files/100/100-0.txt> and preprocessed them to

form O and then trained a model for 100 epochs:

Results:

```
pi = [1.00000000000000044, 2.4195963165377017e-182]
```

```
a =  
[[0.27402870608541646, 0.7259712939145779],  
 [0.758025999436877, 0.24197400056312807]]
```

$b^T$  is as follows

	class 1	class 2
a	0.092232	0.002084
b	0	0.028625
c	0	0.030086
d	0.002888	0.059492
e	0.166469	0.051593
f	0	0.04323
g	0.001312	0.033389
h	0.097191	0.024653
i	0.097862	0
j	0	0.002045
k	0.000002	0.013143
l	0.017057	0.058418
m	0	0.03622
n	0.000002	0.099894
o	0.11383	0.001137
p	0.00157	0.021435

q	0.000013	0.001155
r	0.001203	0.091338
s	0.007471	0.108157
t	0	0.17701
u	0.038081	0.017177
v	0	0.015189
w	0	0.044106
x	0.000887	0.001118
y	0.003475	0.038723
z	0	0.000584
whitespace	0.358454	0.000001

which then results in two classes:

```
[ 'a', 'e', 'h', 'i', 'o', 'u', 'whitespace' ]
[ 'b', 'c', 'd', 'f', 'g', 'j', 'k', 'l', 'm', 'n', 'p', 'q', 'r', 's',
  't', 'v', 'w', 'x', 'y', 'z' ]
```

So, by our experiment we conclude that

We have these two classes in english

We may call the first one a,e,h,i,o,u as vowels and other as consonants

h is only misclassified.

C. After training on 20,000 characters for 100 epochs:

```
pi = [1.00000000000001337, 6.334675146516008e-190]
a = [[0.29895489192080527, 0.7010451080792093],
      [0.7528803039963917, 0.24711969600360517]]
```

$b^T$  is given by

	class 1	class 2
a	0.10258	0.000069
b	0	0.027168
c	0	0.030071
d	0.000091	0.064192
e	0.182908	0.024241
f	0	0.040648
g	0.002491	0.030921
h	0.079113	0.035224
i	0.098299	0.000001
j	0	0.001452
k	0.000076	0.011947
l	0.007949	0.062597
m	0	0.045107
n	0	0.105041
o	0.119211	0.000045
p	0.001094	0.022675
q	0	0.001348
r	0.000002	0.096743
s	0.001145	0.112522
t	0	0.166531
u	0.039464	0.011956
v	0	0.018872
w	0	0.043136
x	0.000164	0.001483
y	0.000124	0.045181
z	0	0.00083
whitespace	0.365289	0.000001

Now the classes formed are

```
['a', 'e', 'h', 'i', 'o', 'u', 'whitespace']
['b', 'c', 'd', 'f', 'g', 'j', 'k', 'l', 'm', 'n', 'p', 'q', 'r', 's',
't', 'v', 'w', 'x', 'y', 'z']
```

D. Training on 50,000 characters:

Took about 3 minutes 30 seconds.

```
pi = [0.999999999999367, 5.630214799842337e-183]
a = [[0.3152499483862772, 0.6847500516137502],
      [0.7608142723368025, 0.23918572766313517]]
```

$b^T$  is given by

	class 1	class 2
a	0.104351	0.000036
b	0	0.028121
c	0	0.031541
d	0.000003	0.064937
e	0.178943	0.018402
f	0	0.039268
g	0.002238	0.029941
h	0.080885	0.033289
i	0.096096	0.000003
j	0	0.00152
k	0.000581	0.011345
l	0.008292	0.063284
m	0	0.047713
n	0.000001	0.103024
o	0.120676	0.000009
p	0.00153	0.021016
q	0	0.001393
r	0.000001	0.097029
s	0.00032	0.11424
t	0	0.168345
u	0.040703	0.010971
v	0	0.021154
w	0	0.044588
x	0.000629	0.00061
y	0.000001	0.047754
z	0	0.000464
whitespace	0.36475	0.000001

Two classes are as follows: when difference  $> 0.001$  is used

```
Class 1: ['a', 'e', 'h', 'i', 'o', 'u', 'whitespace']
Class 2: ['b', 'c', 'd', 'f', 'g', 'j', 'k', 'l', 'm', 'n', 'p', 'q',
          'r', 's', 't', 'v', 'w', 'y']
```

And  $\log(P(O|\lambda))$  was initially -164691.65128603834 , after the 100 iterations it was -136737.95099821006.

This shows that the model has been improved significantly.

#### References:

- [1] A Revealing Introduction to Hidden Markov Models Mark Stamp Department of Computer Science San Jose State University January 12,2018
- [2] <https://www.youtube.com/watch?v=PMIdkmXmZlg>
- [3] <https://www.youtube.com/watch?v=fZ28ZmroMy8>

Thank You Sir