Giang Duong – ID 014533857
Homework I – CS 271
Professor Mark Stamp

1.a
We have $\lambda = (A, B, \pi)$ where

		H	C	
A =	H	**0.7**	**0.3**	
	C	**0.4**	**0.6**	
		M	S	L
O =		1	0	2
		S	M	L
B =		**0.1**	**0.4**	**0.5**
		0.7	**0.2**	**0.1**
		H	C	
π =		**0**	**1**	

O (1, 0, 2)
P(HHH) = 0*0.4*0.7*0.1*0.7*0.5 = 0
P(HHC) = 0*0.4*0.7*0.1*0.3*0.1= 0
P(HCH) = 0*0.4*0.7*0.3*0.4*0.5 = 0
P(HCC) = 0*0.4*0.7*0.3*0.6*0.1 = 0
P(CHH) = 1*0.2*0.4*0.1*0.7*0.5 = 0.0028
P(CHC) = 1*0.2*0.4*0.1*0.3*0.1 = 0.00024
P(CCH) = 1*0.2*0.6*0.7*0.4*0.5 = 0.0168
P(CCC) = 1*0.2*0.6*0.7*0.6*0.1 = 0.00504

The desired probability is the sum of eight probabilities = 0**.024880**

b. Compute P using alphal pass
$\alpha 0(0) = 0.0 * 0.4 = 0$
$\alpha 0(1) = 1.0 * 0.2 = 0.2$
$\alpha 1(0) = (0 * 0.7 + 0.2 *0.4) * 0.1= 0.008$
$\alpha 1(1) = (0 * 0.3 + 0.2 *0.6) *0.7 = 0.084$
$\alpha 2(0)$ = (**0.008 * 0.7 + 0.084*0.4) *0.5 = 0.0196**
$\alpha 2(1) = (0.008 * 0.3+ 0.084* 0.6) * 0.1 = 0.00528$

$\Sigma = \alpha 2(0) + \alpha 2(1) =$ 0**.024880**

c. In term of N and T, and couting only mutiplcations, what is the work factor for the method in part a? what is the work factor of method b?

For the method a, we need to compute each probabilites each time and its will consume a lot of time and space to stores the caculated value. Requires $\mathbf{2TN^T}$

For the method b, we need to find out the each α pass. It will be a shortcut to save time and space but they will not have too much details on each probabilites which can be used in future.

Requires **N^2T**

2.

a. The best hidden state sequence for dynamic programming sense: **CCH**

b. The best hidden state in HMM**: CCH**

Sum the probability of each O (1, 0, 2):

	0	1	2
H	0	0.003040	0.019600
C	1	0.021840	0.005280

3.

a.

```
import math
A = [[0.7, 0.3],
     [0.4, 0.6]]
B = [[0.1, 0.4, 0.5],
     [0.7, 0.2, 0.1]]
pi= [0.6, 0.4]

S = 0;M = 1;L = 2
Set1 = [0, 1, 2]

O = []
sum = []
totalsum = 0

K = []
for a in range(3):
    for b in range(3):
        for c in range(3):
            for d in range(3):
                O.append([a, b, c, d])
print(O)



for i in range(0, 81):
    prob = []
    #HHHH

prob.append((pi[0]*B[0][O[i][0]])*(A[0][0]*B[0][O[i][1]])*(A[0][0]*B[0][O[i][2]])*(A[0
][0]*B[0][O[i][3]]))
    #HHHC

prob.append((pi[0]*B[0][O[i][0]])*(A[0][0]*B[0][O[i][1]])*(A[0][0]*B[0][O[i][2]])*(A[0
][1]*B[1][O[i][3]]))
    #HHCH

prob.append((pi[0]*B[0][O[i][0]])*(A[0][0]*B[0][O[i][1]])*(A[0][1]*B[1][O[i][2]])*(A[1
][0]*B[0][O[i][3]]))
    #HHCC

prob.append((pi[0]*B[0][O[i][0]])*(A[0][0]*B[0][O[i][1]])*(A[0][1]*B[1][O[i][2]])*(A[1
][1]*B[1][O[i][3]]))
    #HCHH
```

```
prob.append((pi[0]*B[0][O[i][0]])*(A[0][1]*B[1][O[i][1]])*(A[1][0]*B[0][O[i][2]])*(A[0][0]*B[0][O[i][3]]))
    #HCHC

prob.append((pi[0]*B[0][O[i][0]])*(A[0][1]*B[1][O[i][1]])*(A[1][0]*B[0][O[i][2]])*(A[0][1]*B[1][O[i][3]]))
    #HCCH

prob.append((pi[0]*B[0][O[i][0]])*(A[0][1]*B[1][O[i][1]])*(A[1][1]*B[1][O[i][2]])*(A[1][0]*B[0][O[i][3]]))
    #HCCC

prob.append((pi[0]*B[0][O[i][0]])*(A[0][1]*B[1][O[i][1]])*(A[1][1]*B[1][O[i][2]])*(A[1][1]*B[1][O[i][3]]))
    #CHHH

prob.append((pi[1]*B[1][O[i][0]])*(A[1][0]*B[0][O[i][1]])*(A[0][0]*B[0][O[i][2]])*(A[0][0]*B[0][O[i][3]]))
    #CHHC

prob.append((pi[1]*B[1][O[i][0]])*(A[1][0]*B[0][O[i][1]])*(A[0][0]*B[0][O[i][2]])*(A[0][1]*B[1][O[i][3]]))
    #CHCH

prob.append((pi[1]*B[1][O[i][0]])*(A[1][0]*B[0][O[i][1]])*(A[0][1]*B[1][O[i][2]])*(A[1][0]*B[0][O[i][3]]))
    #CHCC

prob.append((pi[1]*B[1][O[i][0]])*(A[1][0]*B[0][O[i][1]])*(A[0][1]*B[1][O[i][2]])*(A[1][1]*B[1][O[i][3]]))
    #CCHH

prob.append((pi[1]*B[1][O[i][0]])*(A[1][1]*B[1][O[i][1]])*(A[1][0]*B[0][O[i][2]])*(A[0][0]*B[0][O[i][3]]))
    #CCHC

prob.append((pi[1]*B[1][O[i][0]])*(A[1][1]*B[1][O[i][1]])*(A[1][0]*B[0][O[i][2]])*(A[0][1]*B[1][O[i][3]]))
    #CCCH

prob.append((pi[1]*B[1][O[i][0]])*(A[1][1]*B[1][O[i][1]])*(A[1][1]*B[1][O[i][2]])*(A[1][0]*B[0][O[i][3]]))
    #CCCC

prob.append((pi[1]*B[1][O[i][0]])*(A[1][1]*B[1][O[i][1]])*(A[1][1]*B[1][O[i][2]])*(A[1][1]*B[1][O[i][3]]))
    print("This is the observation set :", O[i])
    #print(prob)

    sum = math.fsum(prob)
    print("Sum = : ", sum, "\n")
    totalsum += sum

print("Total Sum = " , totalsum)
```

3.b

```python
A = [[0.7, 0.3],
     [0.4, 0.6]]
B = [[0.1, 0.4, 0.5],
     [0.7, 0.2, 0.1]]
pi = [0.6, 0.4]

S = 0;
M = 1;
L = 2
Set1 = [0, 1, 2]

O = []
totalsum = 0

for a in range(3):
    for b in range(3):
        for c in range(3):
            for d in range(3):
                O.append([a, b, c, d])

N = len(pi)  # total states
M = len(O[0])  # total observations
#print(N, M)
#print(O)
print("M = ",M)
# compute alpha zero
for step in range(81):
    alpha = [[0.0 for i in range(N)] for _ in range(M)]
    sum = 0.0
    # print(alpha)
    for i in range(N):
        alpha[0][i] = (pi[i] * B[i][O[step][0]])
    # compute alpha (i)
    for t in range(1, M):
        for i in range(N):
            alpha[t][i] = 0
            for j in range(0, N):
                alpha[t][i] = alpha[t][i] + alpha[t - 1][j] * A[j][i]
            alpha[t][i] = alpha[t][i] * B[i][O[step][t]]
            sum += alpha[M-1][i]
        totalsum += sum
    print("Observation : ", O[step])
    print("Alpha = ",alpha)
    print("Sum = ", sum)


print("Total Sum = ",totalsum)
```

4. a,

N = 2

```python
"""
Name: Giang Duong
# Homework 1:
3. Write a HMM program for the English text problem in Section 9.2 of Chapter 9.
Test your each program on each of the following cases.

"""
```

```python
import re
import string
import math
from sys import maxsize
import numpy as np
import random

def FABS(x):
    if x < 0.0:
        x = -x
    else:
        x = x
    return x

def setupfilename():
    # remove all characters except alphabets and word-space, save the new file to new_brown
    with open("brown.txt", "r") as filename, open("newbrown.txt", "w") as newline:
        x = filename.read()
        result = re.sub("[^a-z\s]", "", x, 0, re.IGNORECASE | re.MULTILINE)
        result = result.lower()
        newline.write(result)
        filename.close()

    occurrences = {}  # count the alphabet
    with open('newbrown.txt', 'r') as file:
        letters = string.ascii_lowercase
        for i in file:
            text_lower = i.lower()
            for letter in letters:
                if letter in text_lower:
                    occurrences[letter] = occurrences.get(letter, 0) + 1
        sorted(occurrences)
    # for word in occurrences:
    # print(word, ":", occurrences[word], "times.")
    # stringAZ = list(string.ascii_lowercase +' ')
    # print(stringAZ)

    count = 0
    with open('newbrown.txt', 'r') as file:
        while (count < countto):
            c = file.read(1)
            count += 1
            if not c:
                print("End of File")
                break
            char_to_num = ord(c) - 97
            if (char_to_num < 0):
                char_to_num = 26  # for white space (DEC = 32); 32-97 < 0 then put it to the last : a-z then space
            stringObservation.append(char_to_num)
    # print(stringObservation)
    return stringObservation


def getRandom(N):
    r = 1.0 / N
```

```python
    diff = 0.1 * r
    t = r + (-diff + random.random() * 2 * diff)
    t = np.round(t, decimals=2)
    return t

def rowstocastic(numlist):
    s = sum(numlist)
    norm = [float(i)/s for i in numlist]
    norm = np.round(norm, decimals=2)
    return norm

def createTableA(number_of_state):
    A = []
    for i in range(number_of_state):
        A.append([])
        for j in range (number_of_state):
            A[i].append(getRandom(number_of_state))
        A[i] = rowstocastic(A[i])

    print(np.matrix(A))
    #A = [[intialA for x in range(number_of_state)] for x in range(number_of_state)]

    return A
def createTableB(number_of_state, symbols):
    B = []
    for i in range(number_of_state):
        B.append([])
        for j in range(symbols):
            B[i].append(getRandom(number_of_state))
        B[i] = rowstocastic(B[i])

    print(np.matrix(B))
    #B = [[intialB] * symbols] * number_of_state  # B = NxM matrix
    return B

def createTablepi(number_of_state):
    pi = []
    for i in range(number_of_state):
        pi.append(getRandom(number_of_state))
    pi = rowstocastic(pi)
    print(pi)
    #pi = [intialA, intialA]  # Pi = 1xN
    return pi

"""
Forward algorithm
"""



def forward():
    c[0] = 0.0
    for i in range(number_of_state):
        alpha[0][i] = (pi[i] * B[i][stringObservation[0]])
        c[0] += alpha[0][i]
    """scale alpha """
```

```
    c[0] = (1 / c[0])
    for i in range(number_of_state):
        alpha[0][i] = c[0] * alpha[0][i]
        # alpha[0][i] = alpha[0][i]/ c[0]

    """compute alpha (i)"""
    for t in range(1, T):
        c[t] = 0.0
        for i in range(number_of_state):
            alpha[t][i] = 0.0
            for j in range(number_of_state):
                alpha[t][i] = alpha[t][i] + alpha[t - 1][j] * A[j][i]
            alpha[t][i] = alpha[t][i] * B[i][stringObservation[t]]
            c[t] = c[t] + alpha[t][i]
        c[t] = 1 / c[t]
        # scale A[t][i]
        for i in range(number_of_state):
            alpha[t][i] = c[t] * alpha[t][i]
            # alpha[t][i] = alpha[t][i] / c[t]


""" Backward algorithm"""


def backward():
    Tbackward = T - 1
    for i in range(number_of_state):
        beta[Tbackward][i] = 1.0 * c[Tbackward]
    # compute beta (i)
    for t in reversed(range(T - 1)):
        for i in range(number_of_state):
            beta[t][i] = 0.0
            for j in range(number_of_state):
                beta[t][i] += (A[i][j] * B[j][stringObservation[t + 1]] * beta[t + 1][j])
            beta[t][i] = c[t] * beta[t][i]


""" Compute the gammas and di-gammas"""


def gammasAnddigammas():
    for t in range(0, T - 1):
        denom = 0.0
        temp2 = 0.0
        for i in range(number_of_state):
            gammas[t][i] = 0
            for j in range(number_of_state):
                digammas[t][i][j] = (alpha[t][i] * A[i][j] * B[j][stringObservation[t + 1]] * beta[t + 1][j])  # /denom
                gammas[t][i] += digammas[t][i][j]
            # check gammas, check if gamm[i] == alpha[i] *beta[i] / sum(alpha[j] *beta[j]
            temp2 += gammas[t][i]
            temp = 0.0
            for j in range(number_of_state):
                temp += alpha[t][j] * beta[t][j]
            temp = (alpha[t][i] * beta[t][i]) / temp
```

```python
        if ((FABS(temp - gammas[t][i])) > EPSILON):
            print("gammas ", i, "=", gammas[t][i], temp, "Error!!!")
    if (FABS(1.0 - temp2) > EPSILON):
        print("Sum of gammas's = ", temp2, "should sum to 1.0. \n")

    # special case for gammas[T-1](i)
    for i in range(number_of_state):
        gammas[Tbackward][i] = alpha[Tbackward][i]


""" Re-estimate the model π"""


def reestimate():
    for i in range(number_of_state):
        pi[i] = gammas[0][i]

    # Re-estimate matrix A: Note: follow the row stocastic
    for i in range(number_of_state):
        for j in range(number_of_state):
            numer = 0.0
            denom = 0.0
            for t in range(Tbackward):
                numer += digammas[t][i][j]
                denom += gammas[t][i]
            # if numer == 0:
            #    A[i][j] = 0
            # else:
            A[i][j] = numer / denom

    # Re-estimate matrix B: Note: follow the row stocastic
    for i in range(number_of_state):
        for j in range(symbols):
            numer = 0.0
            denom = 0.0
            for t in range(T):
                if (stringObservation[t] == j):
                    numer += gammas[t][i]
                denom += gammas[t][i]
            # if numer == 0:
            #    B[i][j] = 0
            # else:
            B[i][j] = numer / denom
    return A, B, pi


def computeLog(c):
    """
    Compute log P(O|lambda)
    """
    logProb = 0
    for t in range(T):
        logProb += math.log(c[t])
    return -logProb
```

```python
EPSILON = 0.001
symbols = 27  # 26 characters and word-space; M = 27
number_of_state = 2  # N = 2 because of vowel and consonant
intialA = 1 / number_of_state
intialB = 1 / symbols
T = 50000  # length of observation sequences
Tbackward = T - 1
countfrom = 0
countto = 50000
maxIters = 200
minIters = 20
c = np.zeros((T,), dtype=np.float32)
alpha = np.zeros((T, number_of_state), dtype=np.float32)
beta = np.zeros((T, number_of_state), dtype=np.float32)
gammas = np.zeros((T, number_of_state), dtype=np.float32)
digammas = np.zeros((T, number_of_state, number_of_state), dtype=np.float32)

stringObservation = []
stringObservation = setupfilename()
# observationsequence = symbols ** T
# observation sequence = M^T
A = createTableA(number_of_state)
B = createTableB(number_of_state, symbols)
pi = createTablepi(number_of_state)

iters = 0
logProb = -1.0
newLogProb = 0.0
diff = maxsize
while ((iters < maxIters) and (newLogProb > logProb)):
    print("Iteration: ", iters)
    logProb = newLogProb
    forward()
   #print("Alpha pass. Done!")
    backward()
    #print("Beta pass. Done!")
    gammasAnddigammas()
    #print("Gammas, Digammas pass. Done!")
    reestimate()
    #print("Reestimate pass. Done!")

    logProb = computeLog(c)
    # trick so no initial logProb is requires
    if iters == 0:
        logProb = newLogProb - 1.0
    #print("A=,", A)
    #print("B=,", B)
    #print("pi=,", pi)

    print("Iteration = %d, score log [P(observation |lambda)] = %s" % (iters, logProb))
    iters += 1

# HMM training finished
print("HMM training finished. Model:")
A = np.round(A, decimals=5)
```

```
B = np.round(B, decimals=5)
pi = np.round(pi, decimals=5)
print("Final  ==A==", np.matrix(A))
print("Final  ==B== ", np.matrix(B))
print("Final  ==pi== %s", np.matrix(pi))
```

a. The program genarates final **B^T** matrix. We can see there some B[i][j] in a rows has 0.000 probability which implies there 2 hidden states.
   Its implied that there are 2 separate kind of letters in English which is vowels and consonants.
   After testing with couple test, the word-space is not has zero probability which means it is not belong to the 2 hidden states(vowels and consonants)

   Final ==B==

[[	[	
0.14	0.	
0.	0.023	
0.001	0.054	
0.	0.068	
0.215	0.	
0.	0.035	
0.004	0.024	
0.	0.072	
0.123	0.	
0.	0.004	
0.002	0.006	
0.	0.071	
0.	0.038	
0.	0.112	
0.132	0.112	
0.002	0. 034	
0.	0. 002	
0.	0.1	
0.	0.108	
0.021	0.132	
0.045	0.132	
0.	0.016	
0.	0.022	
0.	0.004	
0.005	0.021	
0.	0.001	
0.31]	0.05	]]

   Final ==pi== %s

[[1. 0.]]

4.b

N = 3

The program genarates final **B^T** matrix. We can see zero probability occurs once(except B[0]) so we cannot claim that there will be 3 hidden states.
Also, there only word-spaces is not has zero which means it is not belong to the 2 hidden states(vowels and consonants). Prove it one more time with N = 3 that it does not belong to any states.

4.c

N =4

The program generates the final **B^T** matrix which has a few zero probability in each row which mean they will not implies about having 4 hidden states.
Once again, word-space has no zero probability. Implies it does not belong to any hidden states.

Final ==B==

[[0. 0.054 0.068 0.041 0. 0.053 0.009 0.235 0.042 0.011 0.005 0.06 0.057 0.027 0. 0.04 0.002 0.111 0.045 0.062 0. 0.012 0.06 0. 0.005 0. 0.]
[0.22 0. 0. 0. 0.337 0. 0. 0.001 0.166 0. 0. 0. 0. 0. 0.207 0. 0. 0.001 0. 0. 0.053 0. 0. 0. 0.015 0. 0.001]
[0. 0.003 0.034 0.016 0. 0. 0.037 0. 0.006 0. 0.007 0.003 0. 0. 0. 0.033 0. 0. 0.073 0.197 0.002 0. 0. 0. 0.012 0. 0.576]
[0. 0.01 0.034 0.103 0. 0.043 0.007 0. 0. 0.001 0.006 0.116 0.047 0.233 0. 0.01 0.002 0.148 0.116 0.036 0.022 0.027 0.009 0.01 0.017 0.002 0.]]

4.d

N = 26

The program generates the final **B^T,** took longer than to execute. Matrix implies that there is not a 26 hidden states.

final B^T =
a 0.00000 0.00000 0.00239 0.00000 0.00000 0.00001 0.00000 0.00000 0.00000 0.00000 0.18363 0.00000 0.00000 0.91975 0.00012 0.00529 0.01074 0.01063 0.00000 0.75773 0.00000 0.00000 0.00167 0.00000 0.03291 0.00000
b 0.00000 0.09849 0.00000 0.03846 0.00000 0.00183 0.00000 0.00000 0.00000 0.00080 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00150 0.00000 0.09108 0.00000 0.04168 0.00000 0.06687 0.00000 0.00276
c 0.08547 0.12496 0.00000 0.02961 0.00000 0.07246 0.00000 0.46297 0.00000 0.00118 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00028 0.00001 0.00000 0.00001 0.00000 0.19295 0.00000 0.00000 0.00000 0.00003
d 0.12990 0.00000 0.00000 0.00199 0.02301 0.08386 0.00000 0.00000 0.00000 0.00975 0.00074 0.00000 0.00000 0.00000 0.50532 0.00000 0.00143 0.00000 0.00000 0.00000 0.00000 0.01916 0.00000 0.13610 0.00000 0.00006

e 0.00000 0.00000 0.00031 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.13143 0.00000 0.00000 0.04967 0.00304 0.99322 0.00000 0.00004 0.00000 0.00006
0.57231 0.00000 0.60117 0.00002 0.00000 0.00000
f 0.00000 0.00000 0.00000 0.00002 0.00000 0.00041 0.00000 0.00000 0.00000 0.07207
0.00000 0.00000 0.00000 0.00000 0.00956 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.15423 0.00000 0.00075 0.00000 0.00101
g 0.01953 0.00000 0.00000 0.01998 0.00083 0.10261 0.00000 0.17541 0.00000 0.00000
0.00368 0.00000 0.00000 0.00000 0.10703 0.00000 0.00000 0.00000 0.00311 0.00000
0.00000 0.03679 0.00000 0.01750 0.00003 0.00000
h 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.91356 0.00000 0.00000 0.00000 0.00000 0.02895 0.00004 0.00000 0.00000
0.00000 0.03400 0.00000 0.03799 0.00306 0.00284
i 0.00000 0.00000 0.92198 0.00000 0.00000 0.00025 0.00000 0.00000 0.00000 0.00108
0.14526 0.00034 0.06096 0.02772 0.00000 0.00000 0.00000 0.00000 0.00000 0.14232
0.00000 0.00000 0.19679 0.00000 0.10956 0.00000
j 0.00000 0.00000 0.00000 0.00028 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00066 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00132 0.00000 0.00573
0.00000 0.02406 0.00000 0.00000 0.00000 0.00000
k 0.02690 0.00000 0.00000 0.00000 0.00000 0.04796 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00951 0.00001 0.00000 0.03353 0.00000 0.00000
0.00000 0.01458 0.00000 0.00001 0.00000 0.00003
l 0.00923 0.19269 0.00000 0.00000 0.00000 0.05620 0.00000 0.00132 0.00000 0.08479
0.00000 0.00002 0.00000 0.00000 0.00373 0.00000 0.00025 0.27943 0.00000 0.00132
0.00000 0.00304 0.00000 0.11790 0.00000 0.29418
m 0.10263 0.00000 0.00000 0.00000 0.00011 0.00239 0.00000 0.00000 0.00000 0.04586
0.00337 0.00000 0.00000 0.00000 0.00985 0.00000 0.00000 0.00000 0.00000 0.00175
0.00000 0.06381 0.00000 0.06644 0.00000 0.31385
n 0.03329 0.00000 0.00000 0.00672 0.00000 0.07887 0.00000 0.00000 0.00000 0.44154
0.00000 0.00000 0.00000 0.00000 0.00020 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.05040 0.00000 0.04407 0.00000 0.03347
o 0.00000 0.00000 0.01407 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00025
0.52924 0.00235 0.00000 0.00270 0.00000 0.00000 0.02331 0.00000 0.00867 0.00000
0.40519 0.00000 0.14699 0.00000 0.06233 0.00000
p 0.00074 0.05167 0.00000 0.00784 0.01450 0.01779 0.00000 0.04568 0.00000 0.00933
0.00000 0.00000 0.00000 0.00000 0.00003 0.00000 0.00000 0.04641 0.00000 0.00000
0.00000 0.19749 0.00000 0.00351 0.00000 0.00064
q 0.00000 0.00000 0.00000 0.00808 0.00000 0.00000 0.00000 0.03089 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
r 0.11812 0.00000 0.00000 0.00122 0.00034 0.00006 0.00000 0.00000 0.00000 0.25945
0.00000 0.04330 0.00000 0.00000 0.00001 0.00000 0.00000 0.00007 0.00000 0.00000
0.00000 0.00000 0.00000 0.34886 0.00000 0.12224
s 0.10846 0.53151 0.00000 0.02112 0.00000 0.00000 0.00000 0.06480 0.00000 0.01878
0.00000 0.00000 0.00000 0.00000 0.25671 0.00000 0.50526 0.07629 0.00000 0.00000
0.00000 0.06735 0.00000 0.07156 0.00000 0.00029

t 0.13773 0.00052 0.00000 0.79016 0.00000 0.53095 0.00000 0.21402 0.00000 0.02298
0.00000 0.00000 0.00000 0.00000 0.06911 0.00000 0.01100 0.54885 0.00000 0.00000
0.01968 0.00009 0.00000 0.00016 0.00000 0.21462
u 0.00000 0.00000 0.06124 0.00000 0.00000 0.00017 0.00000 0.00000 0.00000 0.00087
0.00199 0.03246 0.00000 0.00016 0.00000 0.00000 0.00000 0.00172 0.00000 0.00000
0.00000 0.00000 0.04639 0.00000 0.77881 0.00000
v 0.20611 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.02747 0.00000 0.00461 0.00000 0.01398
w 0.00000 0.00000 0.00000 0.07265 0.00000 0.00000 0.00000 0.00000 0.00000 0.01534
0.00000 0.00798 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.06988 0.00000 0.06070 0.00290 0.00000
x 0.00045 0.00015 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.01620 0.01494
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00100 0.00000 0.00000 0.00000 0.00000
y 0.00596 0.00000 0.00000 0.00000 0.00000 0.00419 0.00000 0.00000 0.00000 0.00099
0.00000 0.00000 0.00000 0.00000 0.02578 0.00000 0.41774 0.00000 0.00114 0.00000
0.00280 0.00203 0.00002 0.02294 0.01039 0.00000
z 0.01547 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00490 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00082 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00001 0.00187 0.96121 0.00000 1.00000 0.00001 0.98380 0.00000
0.00000 0.00000 0.93904 0.00000 0.00000 0.00147 0.00022 0.00017 0.98708 0.00000
0.00001 0.00000 0.00697 0.00001 0.00000 0.00000

REFERENCES

1. M. Stamp, "A revealing introduction to hidden Markov models."
2.