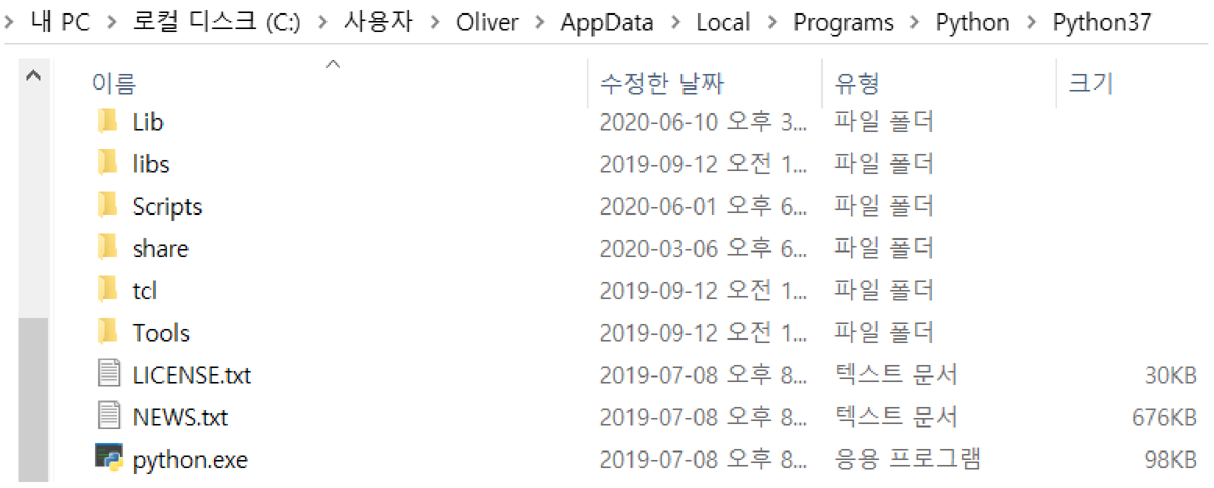
◆ Implementing a speech recognition engine using HMM ◆

Goal: After implementing a speech recognition HMM, obtain a word sequence using the Viterbi Algorithm and create a confusion matrix. This process must be repeated until deletion errors and insertion errors are similar.

It was carried out using Python IDLE and JetBrains PyCharm Community Edition 2019.2.2 x64.

The interpreter is set to python.exe in the Python37 folder in the corresponding path.



› 내 PC › 로컬 디스크 (C:) › 사용자 › Oliver › AppData › Local › Programs › Python › Python37

이름	수정한 날짜	유형	크기
Lib	2020-06-10 오후 3...	파일 폴더	
libs	2019-09-12 오전 1...	파일 폴더	
Scripts	2020-06-01 오후 6...	파일 폴더	
share	2020-03-06 오후 6...	파일 폴더	
tcl	2019-09-12 오전 1...	파일 폴더	
Tools	2019-09-12 오전 1...	파일 폴더	
LICENSE.txt	2019-07-08 오후 8...	텍스트 문서	30KB
NEWS.txt	2019-07-08 오후 8...	텍스트 문서	676KB
python.exe	2019-07-08 오후 8...	응용 프로그램	98KB

ORDER:

1. Read hmm.txt.

2. Read dictionary.txt.

3. Configure word HMM.

4. Read unigram.txt and bigram.txt.

5. Configure the universal utterance HMM.

6. Implement the Viterbi Algorithm for Universal Speech HMM

7. For each test data file in tst.zip,

    ① Read the test data file.

    ② Run the Viterbi algorithm and find the most probable state sequence.

    ③ Convert the state sequence to the corresponding word sequence.

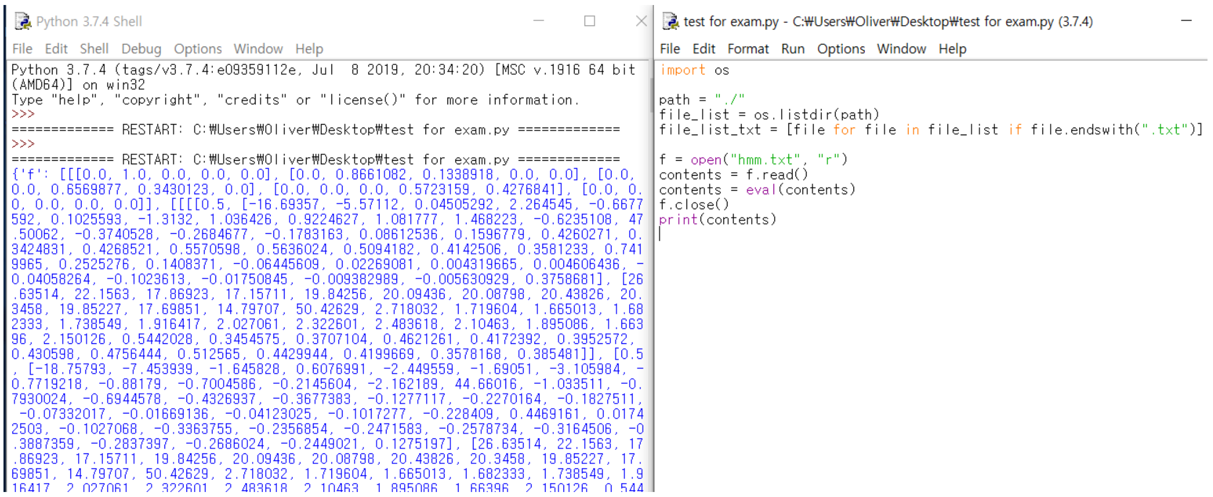
    ④ Output the word sequence.

8. Run HResult.exe to generate a confusion matrix.

9. Repeat steps 7 and 8 with various values of language model weight, balancing deletion and insertion errors to find the best recognition accuracy.

<1. Load hmm.txt file >

First of all, I initially did the method of loading the hmm.txt file itself.

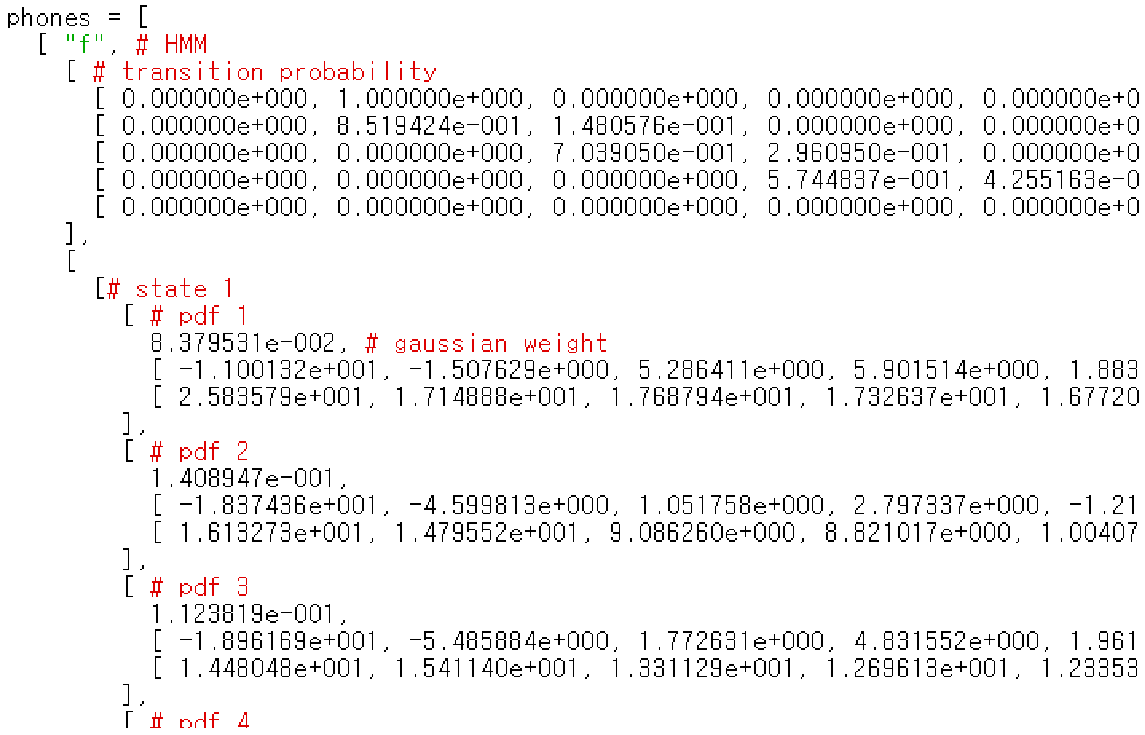


```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:\Users\Oliver\Desktop\test for exam.py ============
>>>
============= RESTART: C:\Users\Oliver\Desktop\test for exam.py ============
{'f': [[[0.0, 1.0, 0.0, 0.0, 0.0], [0.0, 0.8661082, 0.1338918, 0.0, 0.0], [0.0,
0.0, 0.6569877, 0.3430123, 0.0], [0.0, 0.0, 0.0, 0.5723159, 0.4276841], [0.0, 0.
0, 0.0, 0.0, 0.0]], [[[[0.5, [-16.69357, -5.57112, 0.04505292, 2.264545, -0.6677
592, 0.1025593, -1.3132, 1.036426, 0.9224627, 1.081777, 1.468223, -0.6235108, 47
.50062, -0.3740528, -0.2684677, -0.1783163, 0.08612536, 0.1596779, 0.4260271, 0.
3424831, 0.4268521, 0.5570598, 0.5636024, 0.5094182, 0.4142506, 0.3581233, 0.741
9965, 0.2525276, 0.1408371, -0.06445609, 0.02269081, 0.004319665, 0.004606436, -
0.04058264, -0.1023613, -0.01750845, -0.009382989, -0.005630929, 0.3758681], [26
.63514, 22.1563, 17.86923, 17.15711, 19.84256, 20.09436, 20.08798, 20.43826, 20.
3458, 19.85227, 17.69851, 14.79707, 50.42629, 2.718032, 1.719604, 1.665013, 1.68
2333, 1.738549, 1.916417, 2.027061, 2.322601, 2.483618, 2.10463, 1.895086, 1.663
96, 2.150126, 0.5442028, 0.3454575, 0.3707104, 0.4621261, 0.4172392, 0.3952572,
0.430598, 0.4756444, 0.512565, 0.4429944, 0.4199669, 0.3578168, 0.385481]], [0.5
, [-18.75793, -7.453939, -1.645828, 0.6076991, -2.449559, -1.69051, -3.105984, -
0.7719218, -0.88179, -0.7004586, -0.2145604, -2.162189, 44.66016, -1.033511, -0.
7930024, -0.6944578, -0.4326937, -0.3677383, -0.1277117, -0.2270164, -0.1827511,
 -0.07332017, -0.01669136, -0.04123025, -0.1017277, -0.228409, 0.4469161, 0.0174
2503, -0.1027068, -0.3363755, -0.2356854, -0.2471583, -0.2578734, -0.3164506, -0
.3887359, -0.2837397, -0.2686024, -0.2449021, 0.1275197], [26.63514, 22.1563, 17
.86923, 17.15711, 19.84256, 20.09436, 20.08798, 20.43826, 20.3458, 19.85227, 17.
69851, 14.79707, 50.42629, 2.718032, 1.719604, 1.665013, 1.682333, 1.738549, 1.9
16417, 2.027061, 2.322601, 2.483618, 2.10463, 1.895086, 1.66396, 2.150126, 0.544
```

```python
import os

path = "./"
file_list = os.listdir(path)
file_list_txt = [file for file in file_list if file.endswith(".txt")]

f = open("hmm.txt", "r")
contents = f.read()
contents = eval(contents)
f.close()
print(contents)
```

Through the eval() function, dictionary and list forms can be used as they are, but the form of 0.000000e+000 is automatically calculated. It also had the downside of being messy to look at. So, instead of calling the txt file and using it right away, I created a file called header.py as shown below, saved it there, and used it by importing it.

I created a list of phones and put the transition probability as follows:



```
phones = [
  [ "f", # HMM
    [ # transition probability
      [ 0.000000e+000, 1.000000e+000, 0.000000e+000, 0.000000e+000, 0.000000e+0
      [ 0.000000e+000, 8.519424e-001, 1.480576e-001, 0.000000e+000, 0.000000e+0
      [ 0.000000e+000, 0.000000e+000, 7.039050e-001, 2.960950e-001, 0.000000e+0
      [ 0.000000e+000, 0.000000e+000, 0.000000e+000, 5.744837e-001, 4.255163e-0
      [ 0.000000e+000, 0.000000e+000, 0.000000e+000, 0.000000e+000, 0.000000e+0
    ],
    [
      [# state 1
        [ # pdf 1
          8.379531e-002, # gaussian weight
          [ -1.100132e+001, -1.507629e+000, 5.286411e+000, 5.901514e+000, 1.883
          [ 2.583579e+001, 1.714888e+001, 1.768794e+001, 1.732637e+001, 1.67720
        ],
        [ # pdf 2
          1.408947e-001,
          [ -1.837436e+001, -4.599813e+000, 1.051758e+000, 2.797337e+000, -1.21
          [ 1.613273e+001, 1.479552e+001, 9.086260e+000, 8.821017e+000, 1.00407
        ],
        [ # pdf 3
          1.123819e-001,
          [ -1.896169e+001, -5.485884e+000, 1.772631e+000, 4.831552e+000, 1.961
          [ 1.448048e+001, 1.541140e+001, 1.331129e+001, 1.269613e+001, 1.23353
        ],
        [ # pdf 4
```

<u><2. Load dictionary.txt file></u>

I used the following python command: I used the code because it is convenient that it is not closed separately by using with rather than through a variable (example, f.open) method.

By using as dic: it can be used at any time as a variable called dic. The first line of the Dictionary.txt file starts with <s> and sil.

```
with open('dictionary.txt', 'r') as dic:
    for line in dic:
        line = line.split()
        if line[0] == "<s>":
            continue
        else:
            words[line[0]] = line
            words[line[0]][0] = 'sil'
```

<3-5. Loading unigram.txt and bigram.txt, constructing word HMM and UUHMM >

In a similar way as above,

```
with open('bigram.txt', 'r') as bigr:
    for line in bigr:
        line = line.split()
        bigram[(line[0], line[1])] = float(line[2])

with open('unigram.txt', 'r') as unigr:
    for line in unigr:
        line = line.split()
        unigram[line[0]] = float(line[1])
```

The bigram.txt and unigram.txt files are also imported through the with open function. The contents are also declared as variables called bigr and unigr for convenience.

The above two are used when configuring Universal Utterance HMM.

First of all, the HMM class was constructed using __init__ to structure the structure.

```
class HMM():
    def __init__(self, nstates):
        self.nstates = nstates
        self.tran = np.zeros((nstates, nstates))
        self.mean = {}
        self.variance = {}
        self.gconst = {}
        self.weight = {}

    # setter
    def set_state_num(self, val):
        self._nstates = val

    def set_tran(self, val):
        self._tran = val

    def set_mean(self, val):
        self._mean = val

    def set_variance(self, val):
        self._variance = val

    def set_gconst(self, val):
        self._gconst = val

    def set_weight(self, val):
        self._weight = val

    # getter
    def get_nstates(self):
        return self._nstates

    def get_tran(self):
        return self._tran

    def get_mean(self):
        return self._mean

    def get_variance(self):
        return self._variance

    def get_gconst(self):
        return self._gconst

    def get_weight(self):
        return self._weight
```

As above, define a function to receive variables such as nstates, val, mean, variance, and weight.

```
def build_phone_hmm(phones_hmm):
    for obj in range(len(header.phones)):
        phone = header.phones[obj][0]
        nstates = len(header.phones[obj][1])
        phones_hmm[phone] = HMM(nstates)

        # Transition probability
        phones_hmm[phone].tran = np.array(header.phones[obj][1])

        # mean, variance, weight setting
        for state in range(len(header.phones[obj][2])):
            for pdf in range(nPDF):
                phones_hmm[phone].mean[(state+1, pdf+1)] = header.phones[obj][2][state][pdf][1]
                phones_hmm[phone].variance[(state+1, pdf+1)] = header.phones[obj][2][state][pdf][2]
                phones_hmm[phone].weight[(state+1, pdf+1)] = header.phones[obj][2][state][pdf][0]

    for hmm in phones_hmm.keys():
        phones_hmm[hmm].calculate_gconst()
```

Here, it is a function that configures phone hmm. PDF stands for probability density function. Find the mean, variance, and weight of phone hmm through the data in the header file.

```
words = {}
unigram = {}
bigram = {}
```

At the beginning of the code, an empty dictionary to be used in HMM is also declared.

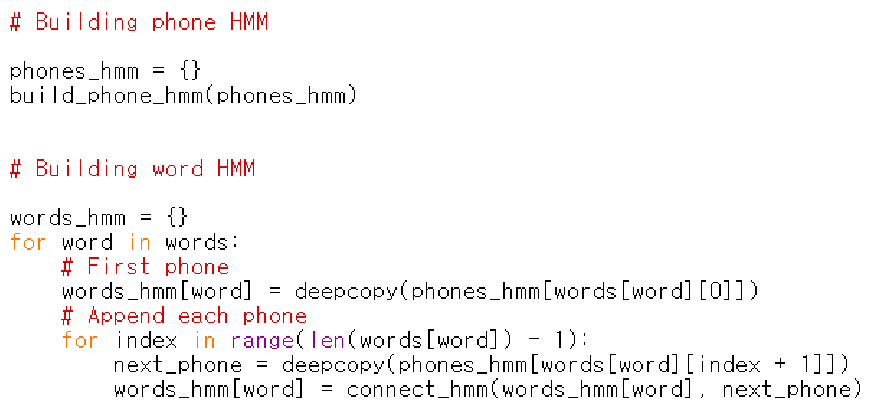
```
# Property setting
nstates = property(get_nstates, set_state_num)
tran = property(get_tran, set_tran)
mean = property(get_mean, set_mean)
variance = property(get_variance, set_variance)
gconst = property(get_gconst, set_gconst)
weight = property(get_weight, set_weight)
```

This part sets the properties supported by Python.

Enter the transition, mean, variance, gconst, and weight of set and get.

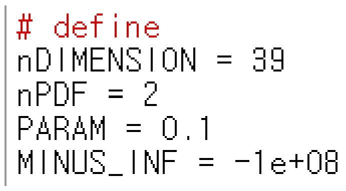
```
def connect_hmm(former, next):
    # connnecting two HMM
    nS1 = former.nstates
    nS2 = next.nstates
    if former.nstates == 0:
        return next;

    nstates = nS1 + nS2 - 2
    conn_hmm = HMM(nstates)

    # Add former_hmm weight, mean, variance
    conn_hmm.weight = deepcopy(former.weight)
    conn_hmm.mean = deepcopy(former.mean)
    conn_hmm.variance = deepcopy(former.variance)

    # Append next_hmm weight, mean, variance
    for state, comp in next.mean.keys():
        conn_hmm.weight[(state + nS1 - 2, comp)] = deepcopy(next.weight[state, comp])
        conn_hmm.mean[(state + nS1 - 2, comp)] = deepcopy(next.mean[state, comp])
        conn_hmm.variance[(state + nS1 - 2, comp)] = deepcopy(next.variance[state, comp])

    # Upadte transition matrix
    conn_hmm.tran[0:former.nstates-1, 0:former.nstates-1] = former.tran[0:former.nstates-1, 0:former.nstates-1]
    conn_hmm.tran[former.nstates-1:nstates, (former.nstates-1):nstates] = next.tran[1:next.nstates, 1:next.nstates]
    next.tran[0] *= former.tran[former.nstates - 2, former.nstates - 1]
    conn_hmm.tran[former.nstates - 2, former.nstates - 1:nstates] = next.tran[0][1:]
    conn_hmm.tran[former.nstates - 2] = normalize(conn_hmm.tran[former.nstates - 2])

    # cal gconst
    conn_hmm.calculate_gconst()

    return conn_hmm
```

This part is the universal utterance part, and it is the part that connects hmm to each other. I defined nS1 and nS2, and nstates was called nS1 + nS2 – 2 by subtracting 2 as described in the assignment. Below is a picture of using in using the properties of a dictionary and adding weight, mean, and variance of next_hmm through state and comp in next.mean.keys(). We used deep copy through the deepcopy method in module copy. After the calculation expression below is finished, the conn_hmm.calculate_gconst() method is called, gconst is obtained, and conn_hmm is returned, and the function ends.



```
# Building phone HMM

phones_hmm = {}
build_phone_hmm(phones_hmm)


# Building word HMM

words_hmm = {}
for word in words:
    # First phone
    words_hmm[word] = deepcopy(phones_hmm[words[word][0]])
    # Append each phone
    for index in range(len(words[word]) - 1):
        next_phone = deepcopy(phones_hmm[words[word][index + 1]])
        words_hmm[word] = connect_hmm(words_hmm[word], next_phone)
```

Call the function called build_phone_hmm created above. Then put it in the dictionary form of phones_hmm. Below is a picture of creating word HMM and adding phone to words_hmm through the for statement.



```
# define
nDIMENSION = 39
nPDF = 2
PARAM = 0.1
MINUS_INF = -1e+08
```

This part is the part that defines.

The number of dimensions is 39,

Number of PDFs 2

Parameter value 0.1

And MINUS_INF required for calculation has been set to -1e+08.

```
# Building final structure

hmm_dict = {}
start = 1
end = 0

hmm = HMM(0)
for word in words:
    hmm = connect_hmm(hmm, words_hmm[word])
    end = hmm.nstates - 1
    hmm_dict[word] = (start, end)
    start = end + 1

combination = [(word_1, word_2) for word_1 in hmm_dict for word_2 in hmm_dict if word_1 != word_2]

for word_1, word_2 in combination:
    (stt1, end1) = hmm_dict[word_1]
    hmm.tran[0, stt1] = unigram[word_1]
    escape = words_hmm[word_1].nstates - 1
    (stt2, end2) = hmm_dict[word_2]
    hmm.tran[end1 - 1, stt2] = 0
    hmm.tran[end1, stt2] = 0

    if word_1 == "zero2" and word_1 != "zero2":
        hmm.tran[end1 - 1, stt2 + 3] = bigram[("zero", word_2)] * words_hmm[word_1].tran[escape - 2, escape] * PARAM
        hmm.tran[end1, stt2 + 3] = bigram[("zero", word_2)] * words_hmm[word_1].tran[escape - 1, escape] * PARAM
    elif word_2 == "zero2" and word_1 != "zero2":
        hmm.tran[end1 - 1, stt2 + 3] = bigram[(word_1, "zero")] * words_hmm[word_1].tran[escape - 2, escape] * PARAM
        hmm.tran[end1, stt2 + 3] = bigram[(word_1, "zero")] * words_hmm[word_1].tran[escape - 1, escape] * PARAM
    elif word_1 == "zero2" and word_2 == "zero2":
        hmm.tran[end1 - 1, stt2 + 3] = bigram[("zero", "zero")] * words_hmm[word_1].tran[escape - 2, escape] * PARAM
        hmm.tran[end1, stt2 + 3] = bigram[("zero", "zero")] * words_hmm[word_1].tran[escape - 1, escape] * PARAM
    elif word_1 != "zero2" and word_2 != "zero2":
        if (word_1, word_2) != ('oh', 'zero') and (word_1, word_2) != ('zero', 'oh'):
            hmm.tran[end1 - 1, stt2 + 3] = bigram[(word_1, word_2)] * words_hmm[word_1].tran[escape - 2, escape] * PARAM
            hmm.tran[end1, stt2 + 3] = bigram[(word_1, word_2)] * words_hmm[word_1].tran[escape - 1, escape] * PARAM
```

Finally, it is the part responsible for the final structure. The functions implemented above are also used. Combination variables make calculations easier. For example, as shown below, if the first word is zero and the second word is also zero, if the first word is not zero and the second word is zero, if the first word is zero but the second word is not zero, and both words are not zero The case is shown below. The conditions of the lowest elif statement, word_1 != "zero2" and word_2 != "zero2", do not work properly and generate an error, so if (word_1, word_2) != ('oh', 'zero') and (word_1, word_2)) != ('zero', 'oh').

Through header.py and functions, HMM is roughly configured as above. However, we still need one more to create a speech recognition HMM. The following describes the formula used with Viterbi.

<6. Implementing functions that compute the Viterbi Algorithm>

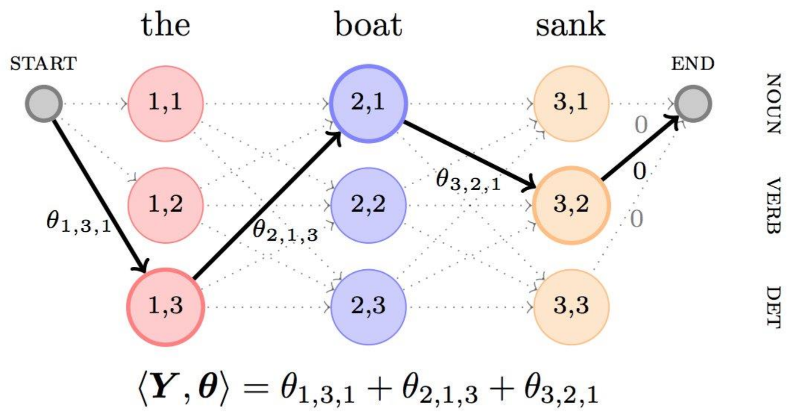


Figure 2: Computational graph of the Viterbi algorithm.

Data source: google search

The figure shows the structure of the Viterbi algorithm. The purpose is to find the most suitable (high probability) sequence after implementing the Viterbi Algorithm using HMM.

The modules used in the hmm.py file are as follows.

```
import numpy as np
from copy import deepcopy
import operator
import os
import header
from calc import *
```

numpy for handling dimensional arrays easily;

copy for copying,

operator for operation,

os to read the file,

header module for configuring the phone,

And the calc module for complex mathematical calculations was called.

First, let's look at the part needed for the calculation.

```
# Calculate gconst : Normalizing constant
def calculate_gconst(self):
    for state, comp in self.weight.keys():
        self.gconst[state, comp] = np.prod(np.sqrt(self.variance[state, comp]))
        self.gconst[state, comp] = 1 / (np.sqrt(2 * np.pi) * self.gconst[state, comp])

def emss(self, state, vec):
    output = MINUS_INF
    result = {}
    for a, b in self.weight.keys():
        if a != state: continue
        result[b] = 0
        for i in range(nDIMENSION):
            result[b] += np.power(vec[i] - self.mean[a, b][i], 2) / self.variance[a, b][i]
        result[b] = log(self.weight[a, b] * self.gconst[a, b] * exp(-0.5 * result[b]))
    for k in result:
        output = logsum(output, result[k])
    return output
```

That part is used to calculate gconst. The formula itself has data and is written in Python. The MINUS_INF value declared above is used here.

```
import numpy as np

MINUS_INF = -1e+08

def exp(x):
    if x == MINUS_INF:
        return 0
    else:
        return np.exp(x)

def log(x):
    if x == 0:
        return MINUS_INF
    else:
        return np.log(x)


def logproduct(x, y):
    if x == MINUS_INF or y == MINUS_INF:
        return MINUS_INF
    else:
        return x + y


def logsum(x, y):
    if x == -1e+08 or y == -1e+08:
        if x != -1e+08:
            return x
        else:
            return y
    elif x > y:
        return x + log(1 + np.exp(y - x))
    elif x < y:
        return y + log(1 + np.exp(x - y))


def normalize(x):
    return x / np.sum(x)


def matprint(mat, fmt="g"):
    col_maxes = [max([len(("{:"+fmt+"}").format(x)) for x in col]) for col in mat.T]
    for x in mat:
        for i, y in enumerate(x):
            print(("{:"+str(col_maxes[i])+fmt+"}").format(y), end="  ")
        print("")
```

That part is calc.py. Again, the value of MINUS_INF is used, and the addition, multiplication, and normalization of exp, log, and log are calculated here. The bottom part is the part responsible for the max of the column (argmax). The function is used in hmm.py by calling. The formula and method refer to Internet resources.

The Viterbi algorithm finds the previous state with the highest probability of transitioning to the current state at all times and for all states.

```python
def viterbi(hmm, x):
    V = {}
    traj = {}
    L = len(x)
    for j in range(hmm.nstates):
        V[(1, j)] = logproduct(log(hmm.tran[0, j]), hmm.emss(j, x[1]))
        traj[(1, j)] = 0
    for t in range(2, L + 1):
        for j in range(hmm.nstates):
            V[(t, j)] = MINUS_INF
            traj[(t, j)] = 0
            for i in range(hmm.nstates):
                if V[(t, j)] < logproduct(V[(t - 1, i)], log(hmm.tran[i, j])):
                    V[(t, j)] = logproduct(V[(t - 1, i)], log(hmm.tran[i, j]))
                    traj[(t, j)] = i
            if j == hmm.nstates - 1:
                V[(t, j)] = MINUS_INF
            else:
                V[(t, j)] = logproduct(V[(t, j)], hmm.emss(j, x[t]))

    V[(L, hmm.nstates - 1)] = MINUS_INF
    for j in range(hmm.nstates):
        if V[(L, hmm.nstates - 1)] < logproduct(V[(L, j)], log(hmm.tran[j, hmm.nstates - 1])):
            V[(L, hmm.nstates - 1)] = logproduct(V[L, j], log(hmm.tran[j, hmm.nstates - 1]))
            traj[(L, hmm.nstates - 1)] = j

    q = [0 for i in range(L)]
    q[L - 1] = traj[L, hmm.nstates - 1]
    for t in range(L - 2, -1, -1):
        q[t] = traj[t + 1, q[t + 1]]
    return q
```

This is the Viterbi algorithm implemented by receiving hmm and x as variables.

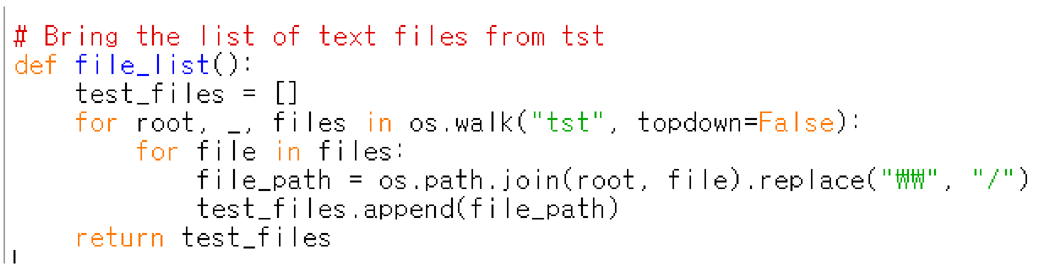
$$
\begin{aligned}
x_1, x_2, \cdots, x_N &= \underset{x \in X}{\arg\max}\, P(x_1, x_2, \cdots, x_N \mid y_1, y_2, \cdots, y_N) \\
&= \underset{x \in X}{\arg\max} \prod_{i=1}^{N} P(y_i \mid x_i) \cdot P(x_i \mid x_{i-1})
\end{aligned}
$$

The Viterbi algorithm should be implemented using the following formula, and I referred to a lot of related materials. HMM uses log to reduce complexity because the computer can come up with very large or small numbers. Assuming that the maximum value finally obtained while calculating through the above algorithm is x, the most optimal path is obtained by going backwards from this x to the place you have stepped on so far.

```python
# prediction from Viterbi Algorithm
def state_word(state_seq):
    seq_word = []
    current_word = ""
    for i in range(len(state_seq) - 1):
        for word in hmm_dict:
            (a, b) = hmm_dict[word]
            if state_seq[i] <= b and state_seq[i] >= a + 3:
                if current_word != word and state_seq[i] > state_seq[i - 1]:
                    current_word = word
                    if current_word == "zero2":
                        seq_word.append("zero")
                    else:
                        seq_word.append(current_word)
                elif state_seq[i] < state_seq[i - 1]:
                    current_word = ""
    return seq_word
```

Lastly, the corresponding part is a function executed in main.py. The most likely path can be predicted through its formula using the Viterbi algorithm.

<8. Load the file in tst.zip and run the program>



```python
# Bring the list of text files from tst
def file_list():
    test_files = []
    for root, _, files in os.walk("tst", topdown=False):
        for file in files:
            file_path = os.path.join(root, file).replace("₩₩", "/")
            test_files.append(file_path)
    return test_files
```

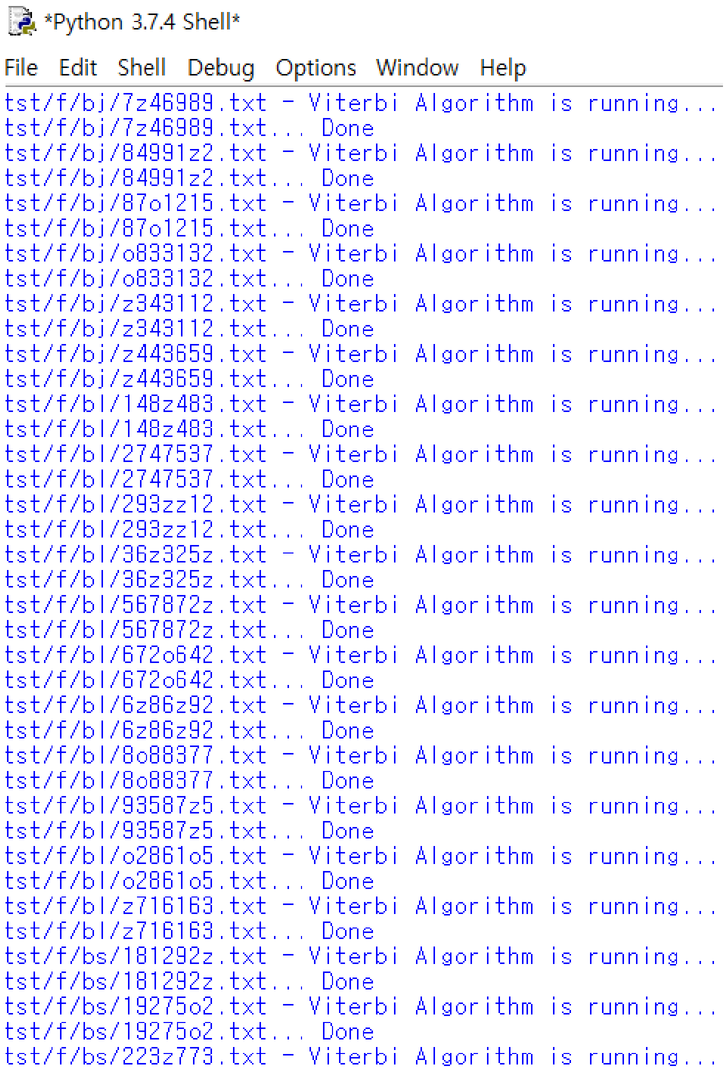
1. Through this function, files in tst can be called one by one by using a for statement.

2. Viterbi algorithm found the order of the most probable state (algorithm).

3. Converted state sequence to word sequence (algorithm).

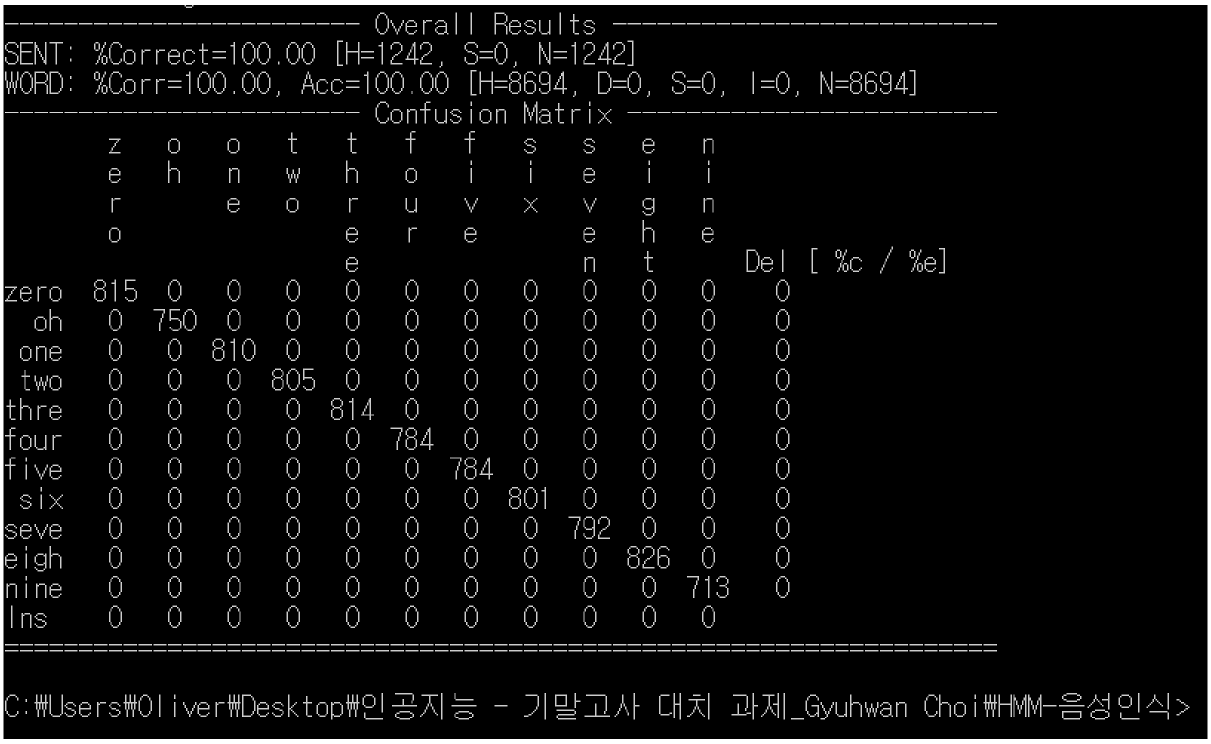
4. The word sequence was output and saved in the recognized.txt file.

Below is the screen displayed when the code is executed:



```
tst/f/bj/7z46989.txt - Viterbi Algorithm is running...
tst/f/bj/7z46989.txt... Done
tst/f/bj/84991z2.txt - Viterbi Algorithm is running...
tst/f/bj/84991z2.txt... Done
tst/f/bj/87o1215.txt - Viterbi Algorithm is running...
tst/f/bj/87o1215.txt... Done
tst/f/bj/o833132.txt - Viterbi Algorithm is running...
tst/f/bj/o833132.txt... Done
tst/f/bj/z343112.txt - Viterbi Algorithm is running...
tst/f/bj/z343112.txt... Done
tst/f/bj/z443659.txt - Viterbi Algorithm is running...
tst/f/bj/z443659.txt... Done
tst/f/bl/148z483.txt - Viterbi Algorithm is running...
tst/f/bl/148z483.txt... Done
tst/f/bl/2747537.txt - Viterbi Algorithm is running...
tst/f/bl/2747537.txt... Done
tst/f/bl/293zz12.txt - Viterbi Algorithm is running...
tst/f/bl/293zz12.txt... Done
tst/f/bl/36z325z.txt - Viterbi Algorithm is running...
tst/f/bl/36z325z.txt... Done
tst/f/bl/567872z.txt - Viterbi Algorithm is running...
tst/f/bl/567872z.txt... Done
tst/f/bl/672o642.txt - Viterbi Algorithm is running...
tst/f/bl/672o642.txt... Done
tst/f/bl/6z86z92.txt - Viterbi Algorithm is running...
tst/f/bl/6z86z92.txt... Done
tst/f/bl/8o88377.txt - Viterbi Algorithm is running...
tst/f/bl/8o88377.txt... Done
tst/f/bl/93587z5.txt - Viterbi Algorithm is running...
tst/f/bl/93587z5.txt... Done
tst/f/bl/o2861o5.txt - Viterbi Algorithm is running...
tst/f/bl/o2861o5.txt... Done
tst/f/bl/z716163.txt - Viterbi Algorithm is running...
tst/f/bl/z716163.txt... Done
tst/f/bs/181292z.txt - Viterbi Algorithm is running...
tst/f/bs/181292z.txt... Done
tst/f/bs/19275o2.txt - Viterbi Algorithm is running...
tst/f/bs/19275o2.txt... Done
tst/f/bs/223z773.txt - Viterbi Algorithm is running...
```

The screen below is the screen you can see when you run HResults. My text file did not run the program to the end, so I thought it was pointless to compare it, so I tested it after converting the reference to recognized.



```
------------------------ Overall Results --------------------------
SENT: %Correct=100.00 [H=1242, S=0, N=1242]
WORD: %Corr=100.00, Acc=100.00 [H=8694, D=0, S=0, I=0, N=8694]
------------------------ Confusion Matrix -------------------------
       z   o   o   t   t   f   f   s   s   e   n
       e   h   n   w   h   o   i   i   e   i   i
       r       e   o   r   u   v   x   v   g   n
       o               e   r   e       e   h   e
                       e               n   t      Del [ %c / %e]
zero  815  0   0   0   0   0   0   0   0   0   0    0
  oh   0  750  0   0   0   0   0   0   0   0   0    0
 one   0   0  810  0   0   0   0   0   0   0   0    0
 two   0   0   0  805  0   0   0   0   0   0   0    0
thre   0   0   0   0  814  0   0   0   0   0   0    0
four   0   0   0   0   0  784  0   0   0   0   0    0
five   0   0   0   0   0   0  784  0   0   0   0    0
 six   0   0   0   0   0   0   0  801  0   0   0    0
seve   0   0   0   0   0   0   0   0  792  0   0    0
eigh   0   0   0   0   0   0   0   0   0  826  0    0
nine   0   0   0   0   0   0   0   0   0   0  713    0
Ins    0   0   0   0   0   0   0   0   0   0   0
===================================================================

C:\Users\Oliver\Desktop\인공지능 - 기말고사 대치 과제_Gyuhwan Choi\HMM-음성인식>
```

Thank you