**인공지능 기말 과제**

고려대학교 컴퓨터학과

2015410113 신채호

**1. 실행 방법**

1). Final.py를 실행 (워드 시퀀스 찾아서 recognized.txt에 쓰기)

2). test.py를 실행 (recognized.txt를 reference.txt 형식으로 변환, 현재 recognized.txt에는 이미 변환된 내용이 들어가있음)

3). Final은 tst폴더 안에 tst가 있어야 실행이 되지만, HResults.exe는 tst 폴더 안에 바로 f와 m폴더가 있어야 돌아가므로 tst 폴더 안에있는 tst 폴더를 한 단계전 폴더인 AIFinal 폴더로 옮겨준다.

3). HResults.exe를 실행시켜 confusion-matrix를 생성한다.

Final을 돌릴 때 tst 파일 경로:

\AIFinal\tst\tst

뒤에 f폴더와 m폴더가 들어가게된다.

HResults.exe를 실행할 때 tst 파일 경로:

\AIFinal\tst

뒤에 f폴더와 m폴더가 들어가게된다.

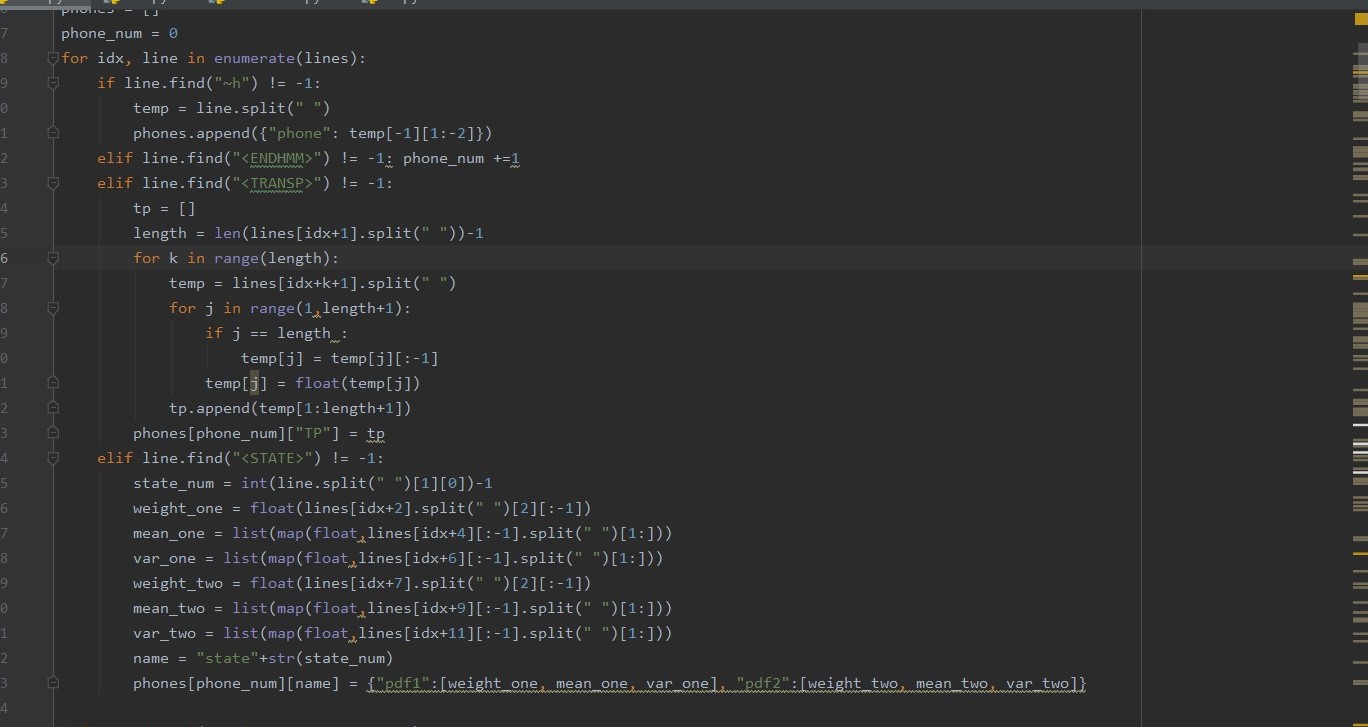
만약 Final.py를 돌리지 않고 바로 HResults.exe를 실행시키려면 test.py를 돌릴 필요가 없음.

**2. 코드 설명**

메인 파일은 Final.py이다. 먼저 맨 처음 for문을 통해 hmm.txt.를 읽어와 phones 배열을 만든다. phones 배열은 딕셔너리가 들어가며 hmm.txt에 들어있는 순서대로 phone\_num을 할당하고 (0부터 시작)

phones[phone\_num] = {”phone string”: {”pdf1”: {weight\_one, mean\_one, var\_one}, …}, “TP”: [Transition table]}

의 형식으로 해당 phone을 key로 갖고 value로 또 각 pdf의 weight, mean, variance가 들어가고, 두 번째 key로 “TP”, value로 transition table을 갖는 딕셔너리 형식이다.

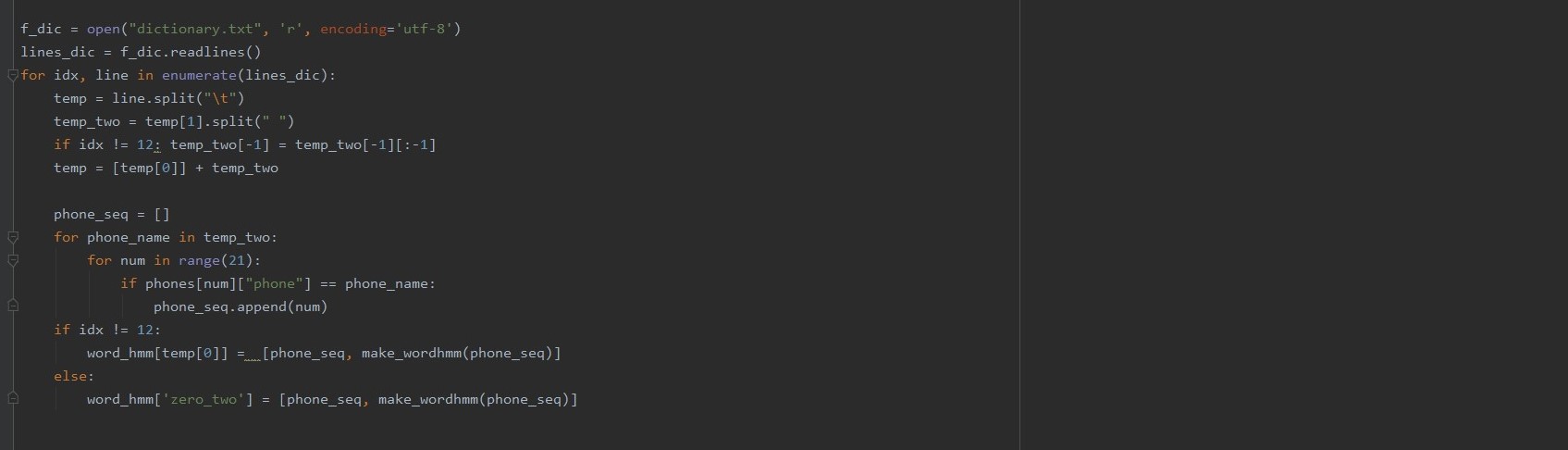


이렇게 phone 배열을 만든후 word\_hmm을 만들어주기 위해 다음과 같이 dictionary.txt에서 word와 구성하는 phone을 불러와 word\_hmm 딕셔너리를 또 만들었다. word\_hmm은

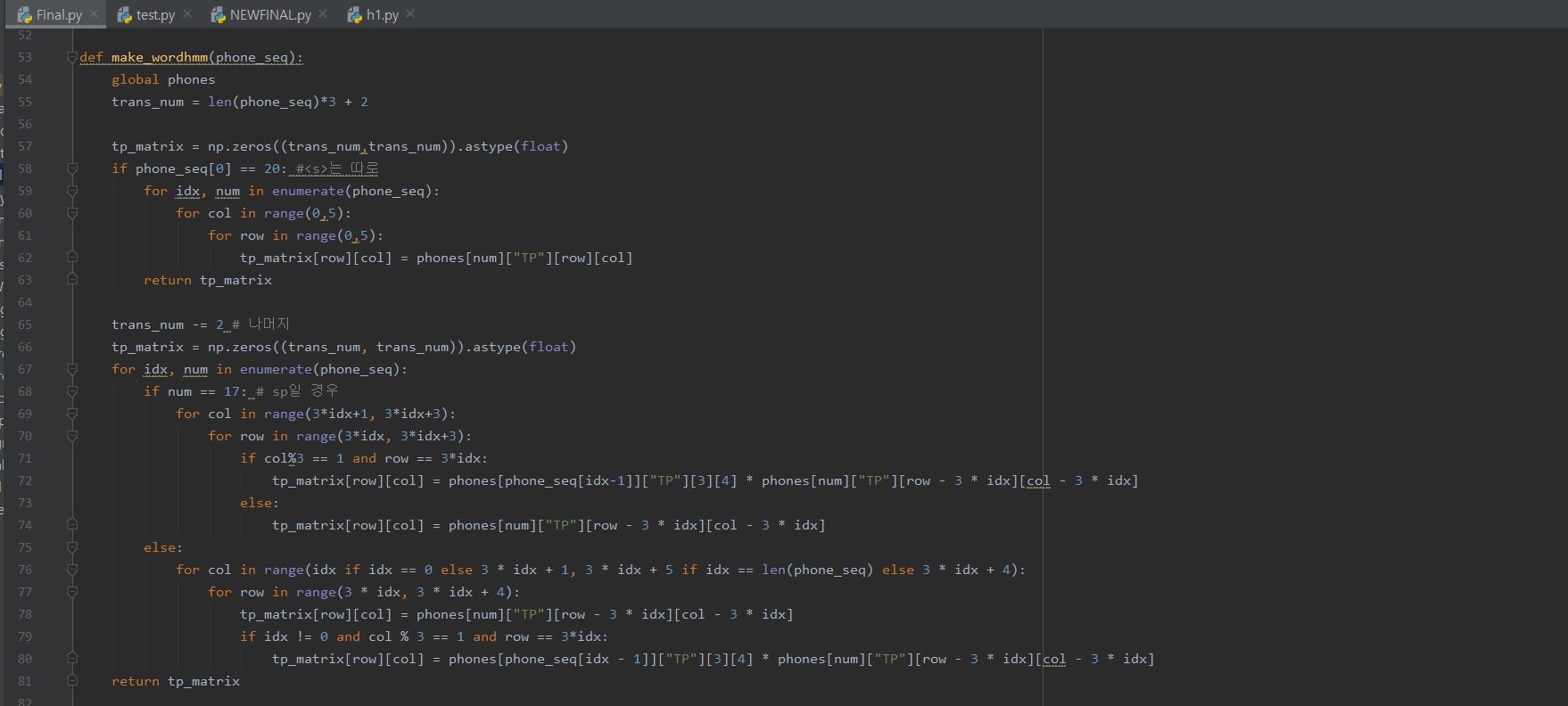
word\_hmm = {”word”:[phone의 sequence, 해당 word의 transition table, 단어의 Unigram 확률]}

같이 구성된다. Word transition table은 make\_wordhmm 함수를 통해 phone의 sequence를 넣어주면 해당 phone들의 transition table을 in과 out probability를 고려해 이어 붙여주도록 되어있다. 또한 단어의 unigram 확률은 unigram.txt 파일을 통해 불러와 주었다.

word\_hmm을 만드는 부분 캡쳐:



word\_hmm에서 쓰이는 word transition\_table을 만들어주는 함수 캡쳐:



Unigram probabilty를 word\_hmm에 append 해주는 부분 캡쳐:

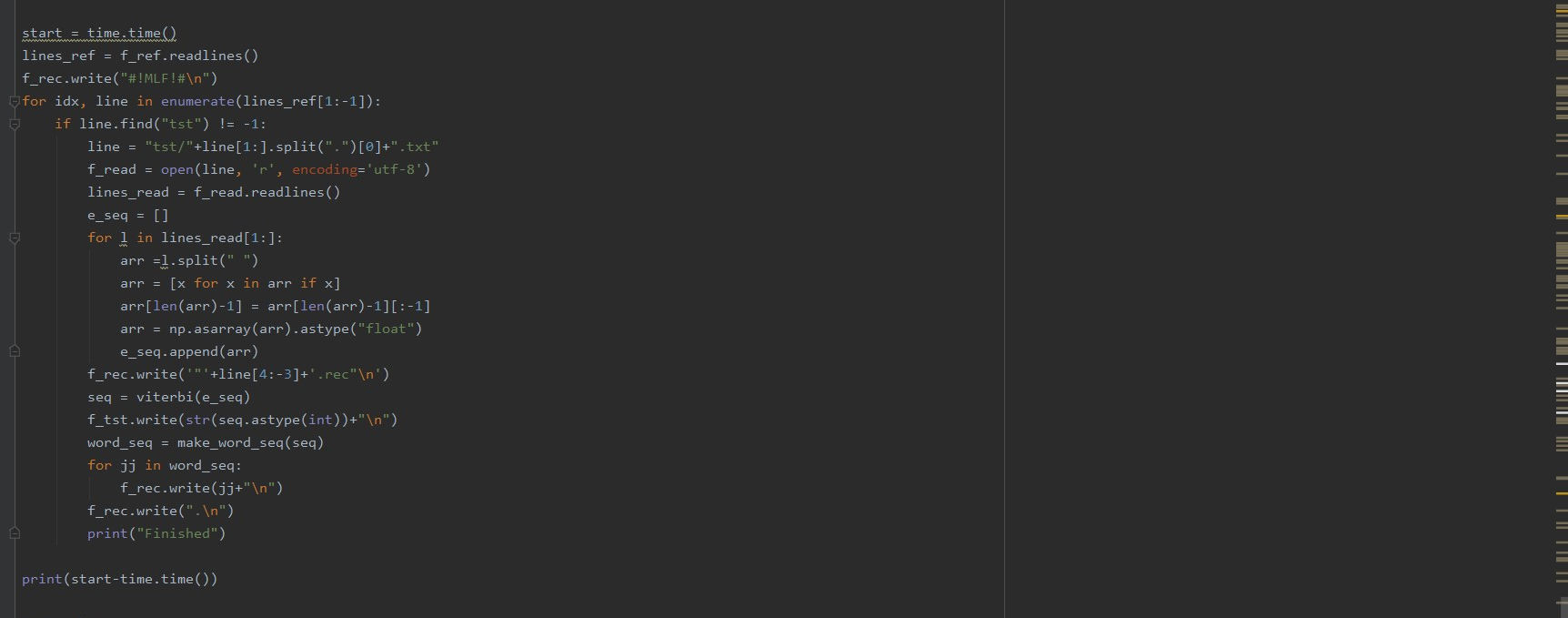


다음으로 bigram.txt 파일을 읽어 bigram\_list에 [첫단어,두 번째단어, 확률] 형식의 리스트로 저장해준다.

이렇게 구성을 마치고 main loop에 들어간다.

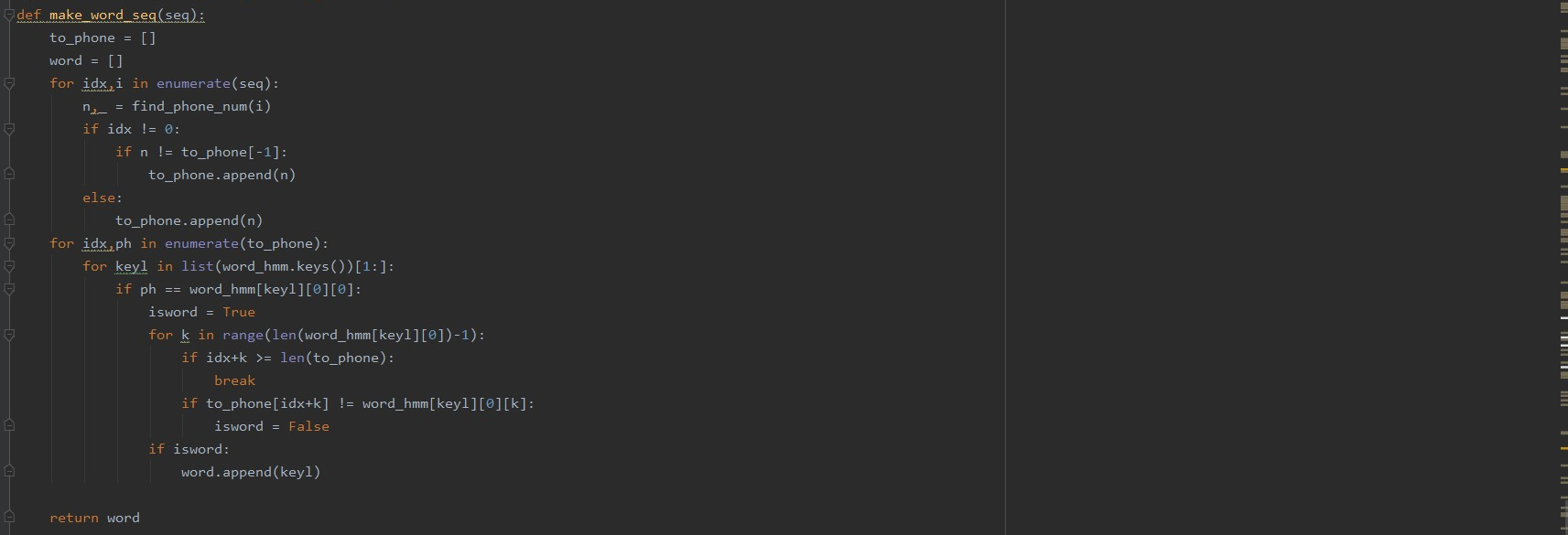
Main loop는 reference.txt를 읽어서 “tst”가 들어가는 라인에서 주소를 따와 lines\_read에 저장을 해주고, lines\_read에서 입력값 시퀀스를 따와 viterbi 함수를 돌려주는 것이다. 그 후 make\_word\_seq(seq) 함수를 통해 phone 시퀀스를 단어 시퀀스로 바꿔준 후 recognized.txt에 쓴다.

Main loop:



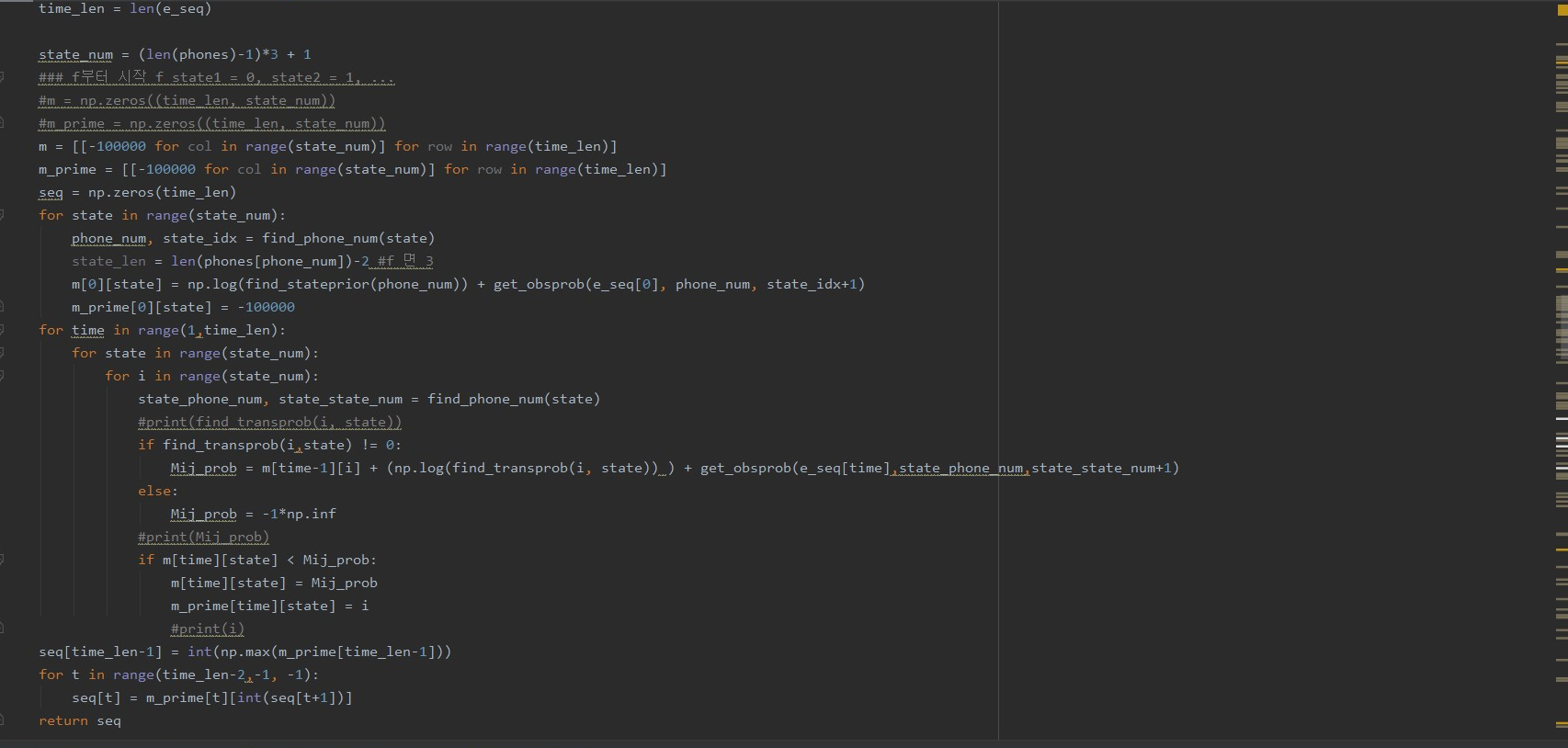
make\_word\_seq(seq) 함수:

phone seq의 중복 부분을 모두 제거한 후 phone들의 list에서 단어 시퀀스가 존재하는지 검사



다음은 가장 중요한 viterbi 함수이다.

viterbi 함수:

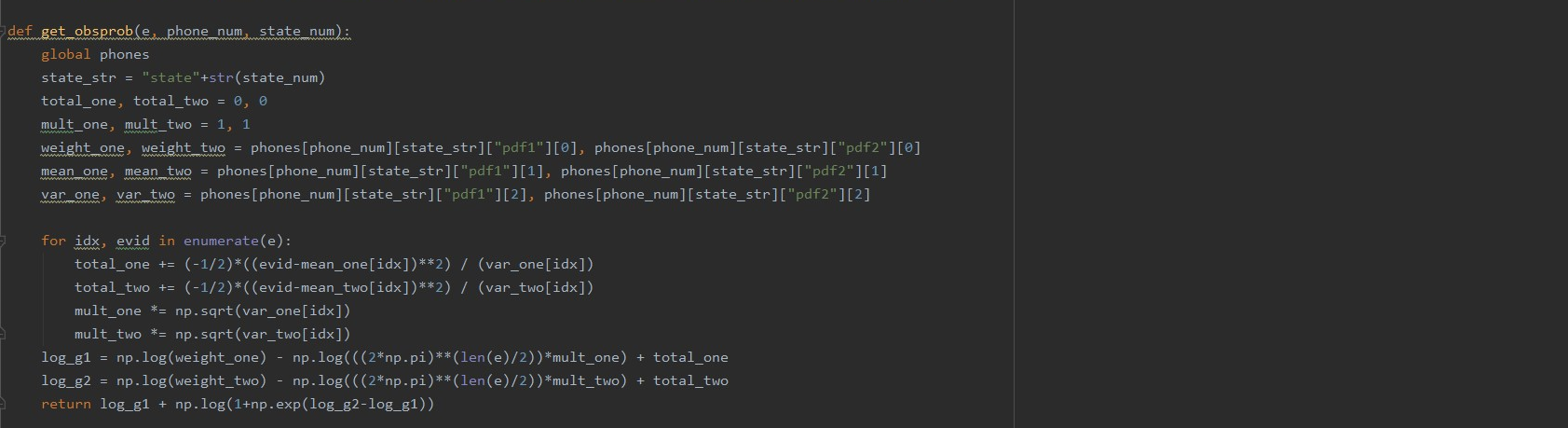


state는 각 phone으로 모두 61개 만들었고, 먼저 수도 코드와 같이 m[0][state]에 prior과 P(e|s)의 곱을 넣어주고 m’을 작은 수로 초기화 시킨다. find\_phone\_num(state)의 함수를 통해 해당 state의 phone 숫자와 몇 번째 state인지 구해주고, prior의 경우 find\_stateprior(phone\_num)의 함수를 통해 word\_hmm에 있는 unigram probability 값을 불러와준다. P(e|s)의 경우 get\_obsprob(e\_seq, phone\_num, state\_idx+1)을 통해 해당 time의 sequence를 phone의 weight, mean, variance를 통해 log(bs(e))를 계산해준다. 그리고 나온 prior값에 log를 취하여 더해준다.

find\_stateprior 함수:



get\_obsprob 함수:

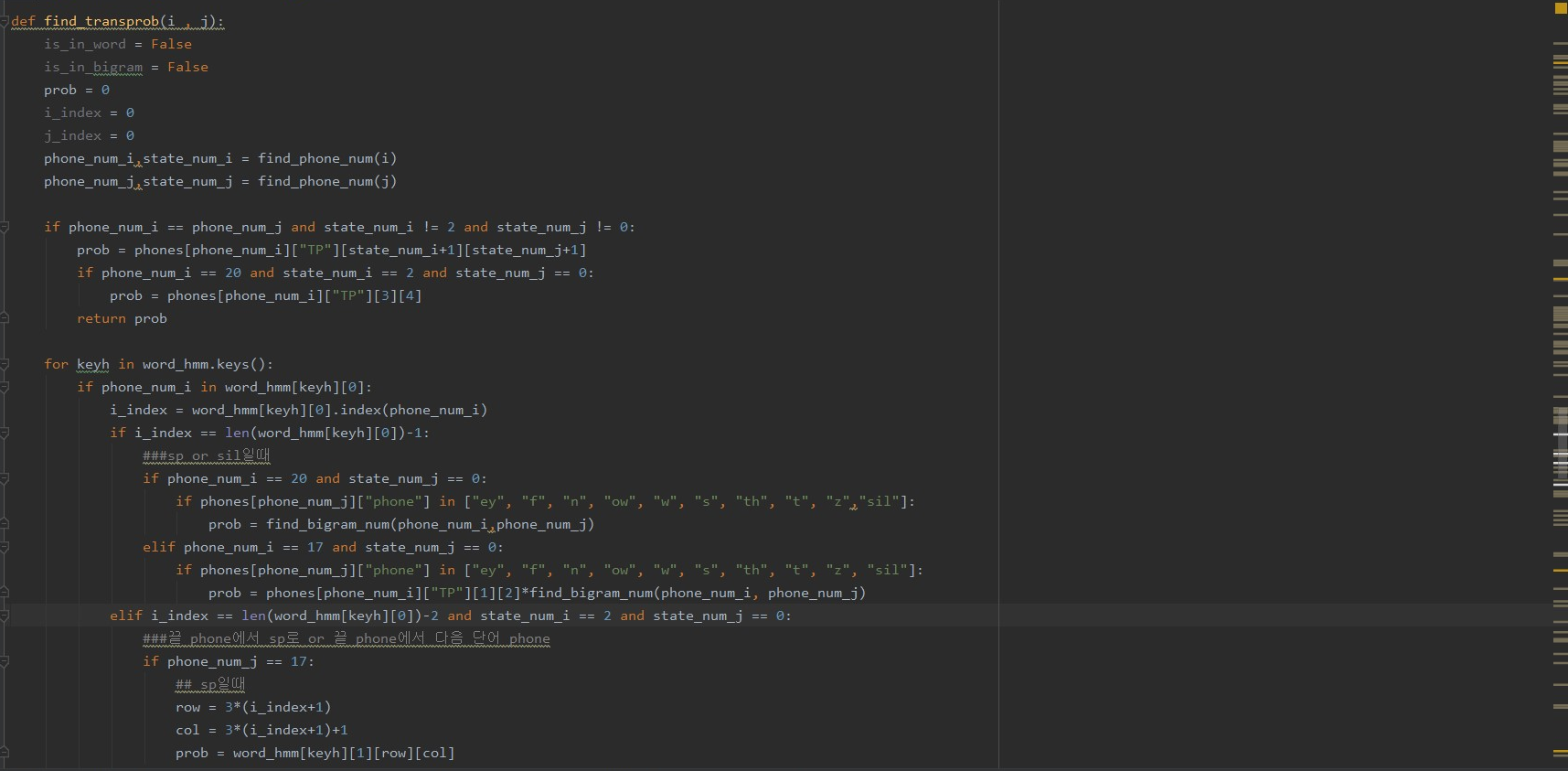


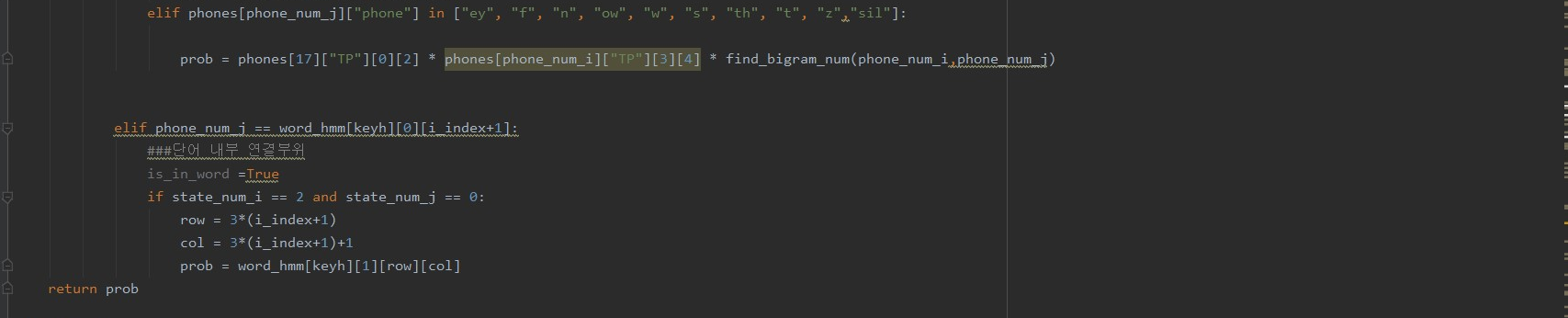
다음으로 각 time에 대하여 모든 state마다 확률의 최댓값을 구해준다. 여기서 확률을 구할 때 현재 state을 state으로 두고, time-1의 state들인 i에 대해서

m[time-1][i] + log(transition\_probability) + P(e|s)의 최댓값을 구해준다.

이때, transition\_probability는 find\_transprob(i,j) 함수를 통해 i state에서 j로 넘어갈 때 Tij가 얼마인지 각 phone들의 state을 고려해 계산해주었다.

find\_transprob 함수:





각 state에서의 최고 i 값을 m’에 저장해 최종적으로 최적의 state sequence를 찾게된다.

**3. 개선점 및 결과 화면**

우선 단어마다 중복되는 phone들을 서로 다른 state로 놓고 했어야 되지만 각 phone의 state만을 state로 놓고 계산해 accuracy가 낮게 나왔다. 그래서 NEWFINAL.py에 단어마다의 각 phone들의 state를 모두 state로 놓고 새로 구현을 해보았는데 급히 하느라 오류가 있을 듯 하고, 코드가 무거워서 그런지 실행이 너무 오래걸려 결과를 확인해보지 못하였다.

다음은 결과화면의 캡쳐이다.

