

다중 화자 탐지 구현과 **STT** 기술 결합

저자 1 : 변상윤

저자 2 : 문성필

지도교수 권준호

목 차

1. 서론	3
1.1. 연구 배경	3
1.2. 기존 문제점	3
1.3. 연구 목표	3
2. 전체 구성 및 배경 지식	4
2.1. 전체 구성	4
2.2. 데이터 분석	5
2.3. API 선정	5
2.4. API 설명	6
2.4.1. UIS-RNN	6
2.4.2. Whisper	7
2.4.3. MFCC	9
3. 연구 내용	9
3.1. Diarization	10
3.1.1. preprocessing	10
3.1.2. Learning	13
3.1.3. Predict	14
4. 연구 결과 분석 및 평가	19
5. 결론 및 향후 연구 방향	21
6. 구성원별 역할 및 개발 일정	22
7. 참고 문헌	23

1. 서론

1.1. 연구 배경

최근 인공지능 기술의 발전으로 음성인식 기술인 **STT (Speech-To-Text)** 모델이 다양하게 개발되었다. **STT**란 사람이 말하는 음성 언어를 컴퓨터가 해석하여 텍스트로 변환하는 처리를 의미한다. 이는 회의록 작성, 유튜브 자막 생성, 상담 기록, 음성 명령어 처리, 청각 장애인들의 학습권 보장 등 다양한 분야에서 활용되고 있다.

우리는 이러한 최신 **STT**를 이용하여 다중화자 기능을 추가함으로써 위의 효과와 더불어 추가적인 효용을 기대할 수 있다. 예를 들어 다중화자 인식에 대한 알고리즘 학습과 그에 대한 심화를 생각 해 볼 수 있다.

1.2. 기존 문제점

실용성 있으면서 다중화자 탐지가 가능한 **stt** 기술이 적용된 애플리케이션이 현재 시중에 많지 않다. 네이버 클로바노트, 에이닷 정도가 많이 쓰이는 애플리케이션인데, 화자를 몇 명인지 선택하거나 2명까지 제한이 되거나하는 제한사항이 있었다. 이런 문제점을 보완해서 다중화자 탐지 기술을 직접 구현해본 다음 기존의 **stt** 기술을 결합하여 실용성 있는 애플리케이션을 만든다면 경쟁력이 있을 것이고, 해당 기술에 대한 이해도도 높아질 것으로 예상된다.

1.3. 연구 목표

다중화자 기술을 기존의 학습되어진 모델을 사용하지 않고 구현한다. 이 목표를 이루기 위해서 먼저 라벨링이 된 데이터셋을 받아 모델을 학습을 시키고, 이 학습된 모델을 이용해 화자 예측을 한다. 그리고 기존 **STT** 기술을 사용해 음성 파일을 대화록으로 바꾸어서 화자 정보와 대화 내용을 출력한다. 이 기술을 이용해 회의나 일상 대화의 자동 대화록 생성 기능을 하는 애플리케이션을 개발하고 모델 별 성능을 측정해본다.

2. 전체 구성 및 배경 지식

2.1. 전체 구성

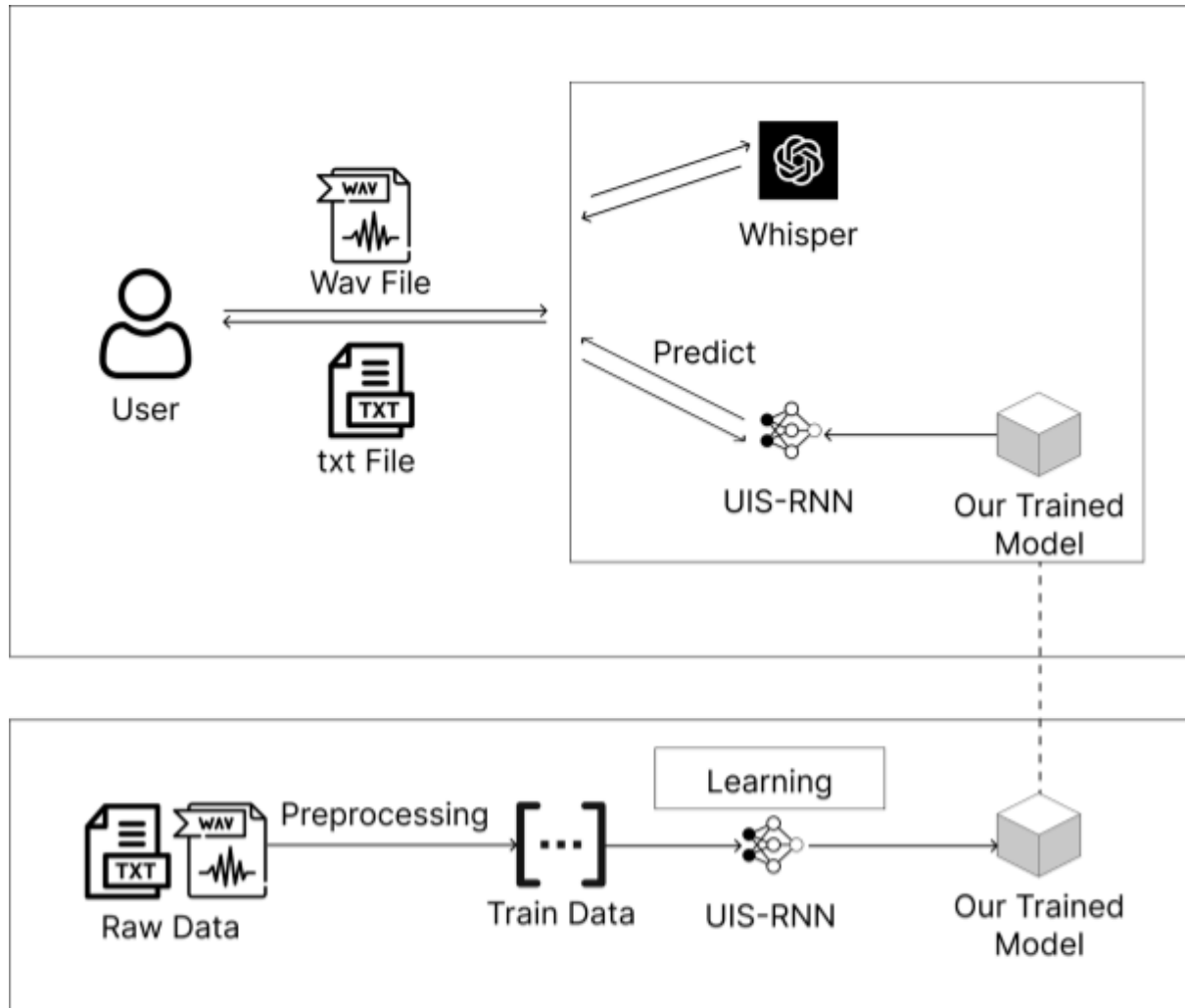


그림1. 전체 아키텍처 도식

그림1에 나와있듯이 사용자 인터페이스를 구성하고, 자세한 기능 구현은 whisper와 UIS-RNN을 이용한다.

aihub에서 받은 음성 파일과 labeling이 된 json 파일을 UIS-RNN에서 사용하는 np array 형식으로 변조한 후 이를 list로 묶어 model 학습을 위한 training data(npz)를 만든다. 그리고 이를 UIS-RNN에 학습시켜 모델을 만든다.

UIS-RNN에서는 우리가 학습시켜 만든 모델을 이용하여 발화별 화자를 예측하도록 한다.

whisper의 출력물인 **utterance**, 타임스탬프와 **UIS-RNN**에서 받은 예측값을 적절히 결합하여 하나의 **txt** 파일로 출력하도록 한다.

2.2. 데이터 분석

이번 연구에 사용된 데이터셋은 **AI-Hub**에서 제공한 주요 영역별 회의 음성인식 데이터이다. 실제 환경, 시사토론, 독서모임, 온라인회의, 방송에서의 자연스러운 환경과 잡음이 결합된 회의 형태의 발성, 발음을 확보하기 위해 실제 환경에서 대화하는 **TV**, 라디오의 고품질 방송 콘텐츠, 의회 녹취록, **UCC** 음성 및 주제를 정한 직접 녹음 음성데이터를 **wav** 파일 형식과 개인정보 및 차별화 혐오발언을 비식별화와 라벨링한 데이터를 **json**파일 형식으로 수집한 데이터이고, 이 중 경제 분야 데이터의 일부분을 가져와서 학습을 진행시켰다. 주요 데이터 분석 방법은 음성 파일의 특징벡터를 추출하고, **json** 파일의 발화 시작 시간과 끝 시간, 화자 번호를 사용해서 **uis-rnn** 모델을 학습시켰다.

2.3. API 선정

2.3.1. UIS-RNN

자료들을 찾아봤을 때 **UIS-RNN**이 화자 수를 알 수 없는 상황에서 다중 화자 구별에 특화된 **API**기도 했고, 참고할 수 있는 자료들도 많았기 때문에, **UIS-RNN**을 선정하기로 했다.

2.3.2. Pyannote

pyannote 같은 경우 단순히 **pretrained model**과 **token**으로 **diarization**을 간단히 구현할 수도 있고 모델을 학습시킬 수도 있었다. **pyannote**의 **training a model tutorial**을 살펴보니 **ami**에서 **db**를 불러왔다. **ami**의 **db**를 살펴보니 **rttm**으로, 이미 전처리된 파일이었다. 사실상 이미 전처리 된 데이터를 학습시키기만 하는 것이라 졸업과제의 의도와 벗어난 것 같아 보류하였다.

2.3.3. Kaldi

kaldi의 경우 음성 인식과 화자 분할을 위한 오픈 소스 툴킷이며 주로 음성 데이터를

처리하고 인식하는 데 사용된다. 컴포넌트 기반이고 확장성이 좋다. 그러나 문서에 대해 학습 곡선이 가파르고, 셀 파일로 구성되어 있어 익숙지 않은 환경이라 보류하기로 하였다.

2.3.4. Whisper

높은 정확도와 다국어 지원으로 한국어의 경우에도 인식률이 높고. 오픈 소스로 무료로 사용할 수 있어서 선정했다.

2.3.5. MFCC

MFCC는 음성을 짧은 시간 간격(10~20ms)단위로 쪼개 계산하기 때문에 음성 신호의 시간적 변화를 잘 포착할 수 있다. 이는 음성 인식 뿐 아니라 화자 인식 분야에서 활용될 수 있기 때문에 MFCC 알고리즘을 사용하기로 결정하였다.

2.4. API 설명

2.4.1. UIS-RNN

UIS-RNN은 unbounded interleaved-state rnn으로서, 교차로 배치된 rnn을 뜻한다. rnn이란 Recurrent Neural Network로, 순환 신경망이라고 하고 순차적 데이터 또는 시계열 데이터로 훈련된 심층 신경망으로, 순차적 입력을 기반으로 순차적 예측 또는 결론을 내릴 수 있는 머신 러닝 모델을 만들어 시간에 따라 변하는 데이터를 효과적으로 처리한다. 그림 2는 UIS-RNN의 시스템 구조도이다.

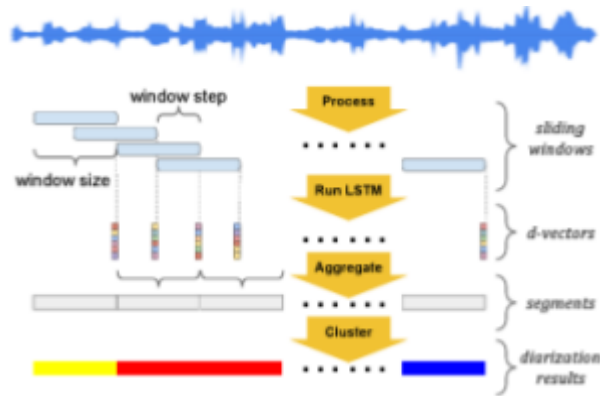


그림 2. UIS-RNN의 시스템 구조도. 참고 문헌 [1]

Training은 `fit()` 메소드로 할 수 있다. `fit()` 함수는 두 가지 매개변수를 받을 수 있는데, training sequence와 training cluster ID를 받을 수 있다. train sequence는 observation sequence이고 이는 2 dimension numpy array of type float64이다. first dimension은 sequence의 length이고 이는 각 sequence마다 다를 수 있다. second dimension은 각 observation의 size이다. 이는 모든 sequence사이에서 동일해야 한다. observation은 d-vector가 될 수 있다. 예측은 `predict()` 메소드를 사용해서 할 수 있다. `predict()` 함수는 두 가지 매개 변수를 받을 수 있는데 test sequence와 inference arguments다. inference arguments에는 beam size, look ahead, test iteration이 존재한다.

2.4.2. Whisper

Whisper는 OpenAI에서 개발한 음성 인식 모델이다. 많은 다양한 음성의 데이터셋을 이용해서 학습시켰고, 여러 언어의 발화 인식과, 발화 번역, 언어 식별, 음성 활동 감지가 가능하다. Whisper는 인코더 - 디코더 transformer 구조를 사용한다. 작동 방식은 오디오 파일을 불러와서, 이를 log-mel spectrogram으로 변환을 하고, 이 것을 인코더를 통해서 이 파일을 처리하고 디코더를 통해 언어 식별, 타임 스탬프 생성, 텍스트 생성을 해 출력을 한다. 이 후 우리가 다중 화자 분리를 구현한 부분을 결합해서 프로그램을 제작할 것이다. 그림 3은 Whisper의 구조도이다.

2.4.3. MFCC

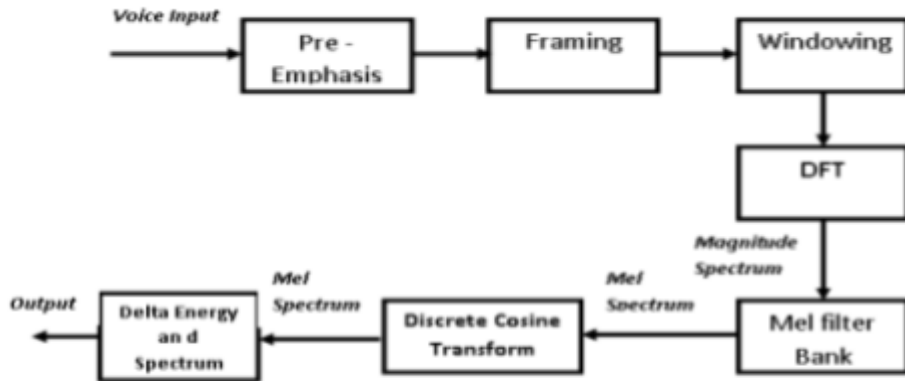


그림 4. MFCC block diagram. 참고문헌[3]

MFCC란 음성 데이터를 특징벡터화 해 주는 알고리즘이다. 사람의 달팽이관의 특성을 고려한 값으로 도식도는 그림 4와 같다.

framing 과정에서 음성데이터를 쪼개고 프레임을 겹치게 분할한다. 이때 프레임을 discrete하게 분할 하는 것이 아니라 50%정도 겹치게 분할하는 것이다. 샘플링 과정에서 프레임의 접합부에서 순간 변화율이 무한대가 되는 것을 막기 위함이다.

그 뒤 frame별로 hamming window를 적용해주고, DFT(Discrete Fourier Transform)를 적용하여 주파수 성분을 얻는다. Mel filter bank 부분에선 달팽이관 특징에 맞게 낮은 주파수에서는 작은 filter를 가지고 높은 주파수에서는 큰 filter를 가진다. 그다음 DCT(Discrete Cosine Transform)를 사용하여 시간 도메인으로 멜 스펙트럼을 변환한다. 각 발화는 음향 벡터로 변환된다. 음성 신호와 프레임이 변하면서 포먼트의 기울기가 변하고, 캡스트럼 특징의 변화에 대해 추가적인 feature를 추가 할 필요가 있다. 이를 보정하기 위한 것이 마지막 단계이다.

3. 연구 내용

연구 내용에 앞서 재현성을 위해 하드웨어 환경과 소프트웨어 환경에 대해 설명하겠다.

컴퓨팅 파워가 필요하여 데이터를 가공하기로 하였다.

```
# n_mfcc 값, observation dim 값
N_MFCC_VALUE = 13

FRAME_RATE = 16000

HOP_LENGTH = 160
# 기본 frame rate :16000
# hop time :0.01s

N_FFT = 320
# frame rate : 16000
# window 크기 0.02s
```

그림 6. MFCC 매개변수

음성 데이터를 MFCC 알고리즘으로 변형시켜주었다. 그림 6은 MFCC 변형에 필요한 파라미터들이다.

실행 성능을 고려해 단순히 음성 파일의 샘플링 속도를 낮출 수도 있었지만, 이는 음질 해상도를 저하시키고 학습에 부정적인 영향을 미칠 수 있어 보류하였다. 샘플링 속도는 16,000Hz로 고정하였고, 특징 벡터의 개수인 N_MFCC_VALUE는 처음에 64로 설정했다가 이후 13으로 낮추었다. 일반적인 상황에서 화자의 수가 그리 많지 않을 것이라 예상하여 낮추어도 성능면에 있어 큰 차이가 없을

것으로 판단했기 때문이다. N_FFT는 음성을 일정 구간으로 나누는 윈도우의 크기이며, HOP_LENGTH는 각 구간에서 건너뛰는 길이를 의미한다. HOP_LENGTH를 160으로 설정하면 160/16000으로 0.01초 간격으로 이동하게 된다. N_FFT/FRAME_RATE는 음성 벡터를 쪼개는 최소 단위인데 여기서 320/16000이므로 0.02초가 되며, 이 구간을 50%씩 겹치면서 0.01초씩(hop length) 이동한다. hop length가 0.01 이므로 음성 파일 1초당 100개의 리스트가 생성되고, MFCC 특징 벡터의 차원을 13으로 설정했기 때문에 1초당 1,300개의 요소가 생성된다. 초기에는 N_FFT와 HOP_LENGTH를 10배 크게 설정해 빠른 학습을 시도했으나, 예측 정확도가 0.07로(그림 7) 매우 저조했다. 이에 따라 정확도가 더 중요하다고 판단하여 N_FFT와 HOP_LENGTH를 각각 320과 160으로 조정하였다.

```
Performance:
  averaged accuracy: 0.074000
  accuracy numbers for all testing sequences:
    0.074000
=====

All predict done : 380.32299065589905s
○ zxc6147@zxc6147-MS-7889:/media/zxc6147/06E694BCE694AD85/Users/zxc61
```

그림 7. N_FFT와 HOP_LENGTH를 10배 크게 설정한 후 예측한 결과

MFCC 메소드를 호출하면 ndarray[shape=(N_MFCC_VALUE, 영상에 따른 길이)]가 반환된다. 그러나 UIS-RNN의 fit() 내의 fit_concatenated 설명을(그림 8) 살펴보면 train_sequence의 형태가 (summation of lengths of all utterances, observation dimension)이므로 mfcc를 transpose 해주어서 같은 형태로 만들어주었다.

`def fit_concatenated(self, train_sequence, train_cluster_id, args):`[▶ View Source](#)

Fit UISRNN model to concatenated sequence and cluster_id.

Args: train_sequence: the training observation sequence, which is a 2-dim numpy array of real numbers, of size `N * D`.

- 'N': summation of lengths of all utterances.
- 'D': observation dimension.

For example,

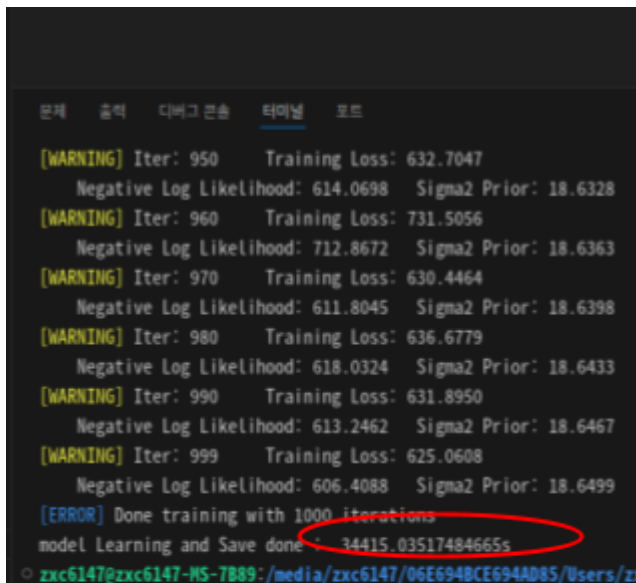
```
train_sequence =
[[1.2 3.0 -4.1 6.0] --> an entry of speaker #0 from utterance 'iaaa'
 [0.8 -1.1 0.4 0.5] --> an entry of speaker #1 from utterance 'iaaa'
 [-0.2 1.0 3.8 5.7] --> an entry of speaker #0 from utterance 'iaaa'
 [3.8 -0.1 1.5 2.3] --> an entry of speaker #0 from utterance 'ibbb'
 [1.2 1.4 3.6 -2.7]] --> an entry of speaker #0 from utterance 'ibbb'
```

Here 'N=5', 'D=4'.

We concatenate all training utterances into this single sequence.

그림.8 fit_concatenated 설명. 참고 문헌 [4]

이때까지 한 전처리에도 불구하고 CUDA(HIP) out of memory 에러가 계속 일어났다. 이 에러의 경우 학습 데이터의 크기가 너무 커 한번에 메모리에 올리기 힘든 경우에도 일어나고 batch size가 너무 큰 경우에도 일어난다. 따라서 영상을 10등분 정도로 쪼개서 학습시켜 이를 방지토록 하였다.



A terminal window showing the output of a training process. The output includes several lines of training metrics for iterations 950 through 999, such as Training Loss, Negative Log Likelihood, and Sigma2 Prior. The final line shows an error message: "[ERROR] Done training with 1000 iterations" followed by "model Learning and Save done" and a timestamp "34415.03517484665s". The timestamp is circled in red. The terminal window has tabs at the top labeled "문제", "출력", "디버그 콘솔", "터미널", and "포트".

```
[WARNING] Iter: 950    Training Loss: 632.7047
Negative Log Likelihood: 614.0698    Sigma2 Prior: 18.6328
[WARNING] Iter: 960    Training Loss: 731.5056
Negative Log Likelihood: 712.8672    Sigma2 Prior: 18.6363
[WARNING] Iter: 970    Training Loss: 630.4464
Negative Log Likelihood: 611.8045    Sigma2 Prior: 18.6398
[WARNING] Iter: 980    Training Loss: 636.6779
Negative Log Likelihood: 618.0324    Sigma2 Prior: 18.6433
[WARNING] Iter: 990    Training Loss: 631.8950
Negative Log Likelihood: 613.2462    Sigma2 Prior: 18.6467
[WARNING] Iter: 999    Training Loss: 625.0608
Negative Log Likelihood: 606.4088    Sigma2 Prior: 18.6499
[ERROR] Done training with 1000 iterations
model Learning and Save done 34415.03517484665s
zxc6147@zxc6147-WS-7B89: /media/zxc6147/06E6948CE694AD85/Users/zx
```

그림 9. 학습 1000회

3.1.2. Learning

반복 학습 1000회를 모두 완료하는데에 34000초 가량(그림 9)이 걸렸다. 학습 시킨 음성 파일이 35분 24초짜리임을 감안해도 긴 시간이다. 첫 번째 파일 뿐 아니라 다른 파일도 마찬가지로 10시간 정도 걸려 시간상의 이유로 3개 정도만 학습시켰다.

```

[WARNING] Iter: 0      Training Loss: -14.1906
      Negative Log Likelihood: 6.9631      Sigma2 Prior: -21.1572      Regularization: 0.0036
[WARNING] Iter: 10     Training Loss: -14.2753
      Negative Log Likelihood: 6.0932      Sigma2 Prior: -20.3720      Regularization: 0.0035
[WARNING] Iter: 20     Training Loss: -14.4913
      Negative Log Likelihood: 6.0315      Sigma2 Prior: -20.5264      Regularization: 0.0036
[WARNING] Iter: 30     Training Loss: -14.5031
      Negative Log Likelihood: 6.3192      Sigma2 Prior: -20.8258      Regularization: 0.0036
[WARNING] Iter: 40     Training Loss: -14.0478
      Negative Log Likelihood: 6.8473      Sigma2 Prior: -20.8986      Regularization: 0.0036
[WARNING] Iter: 49     Training Loss: -14.5412
      Negative Log Likelihood: 6.3787      Sigma2 Prior: -20.9234      Regularization: 0.0036
[ERROR] Done training with 50 iterations
model Learning and Save done : 53104.36481785774s

```

그림 10. 75개 파일 50번 iteration

학습이 몇 개의 음성 파일에만 집중되어 있는 것 같아 반복 횟수를 줄이고 파일의 숫자를 늘린 모델을 하나 더 만들었다. 75개 정도의 파일을 50번씩만 학습했음에도 총 53,104초가 걸렸다. (그림 10)

3.1.3. Predict

```

422, 422, 422, 422, 422, 444, 444, 444, 444, 4
4, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94,
Config:
  sigma_alpha: 1.0
  sigma_beta: 1.0
  crp_alpha: 1.0
  learning rate: 0.001
  regularization: 1e-05
  batch size: 8

Performance:
  averaged accuracy: 0.379659
  accuracy numbers for all testing sequences:
    0.379659
=====
All predict done : 36340.04804420471s
○ zxc6147@zxc6147-MS-7B89:/media/zxc6147/06E694B

```

그림 11. wav파일
1/50만큼 잘라서 학습

aihub의 음성 파일중 하나를 예측해보았다. 30~40분 가량의 음성 파일 모두를 예측하기에는 시간이 너무 많이 걸릴 것이라 생각하여 1/50 정도로 잘라서 하였다. 화자 번호를 clustering과 indexing을 하여 요소별로 비교한 정확도는 0.38정도로 나왔다. 예측 시간은 36000초 가량으로 나왔다. 예측의 결과 값인 화자 번호가 화자 수와는 괴리가 있는 큰 값이 나왔는데 이 값들은 선형적으로 있는 값이 아니라 특징 표현을 위한 값이다. 즉, 그림 11에서 화자 번호가 422가 나왔다고해서 그것이 1부터 422까지 모두 있다는 것은 아니다. 우리는 이를 사용자 관점에서 이해하기 쉽도록 하기 위해 매핑을 도입했다.

그림 11의 Config 위쪽 숫자들과 같은 비선형적인 화자 번호를 0부터 시작하며 매핑을 해주었다. 가장 많이 나온 id인 화자 번호의 값을 key로 받는 dictionary를 만들고, value의 값을 0부터 시작하여 새로운 key가 추가 될 때마다 1씩 늘려 0부터 선형적으로 증가하는 화자 번호로 매핑 하는 것이다. 이 key들의 value가 바로 speaker number가 되는 것이다.

예측과 정확도 판단은 0.01초 단위로 쪼개어 했지만 우리는 최종 결과물로 0.01초 단위의 화자 번호가 필요한 것이 아니다. whisper가 판단해낸 문장 구별별과 타임스탬프에 맞게 우리가 예측한 화자 번호를 매칭시켜줘야한다. whisper의 타임스탬프시간에 맞춰 화자 번호 리스트를 자른 후 잘린 구간 내 가장 많은 화자 번호를 대표 화자 번호로 선택하는 것이다. 그렇게 하면 몇 몇 튀는 값이 있어도 가장 정답일 가능성이 높은 화자 번호를 반환할 것이라 생각한다. 따라서, 예측의 결과물을 타임스탬프에 따라 매핑 해주고, 잘린 구간 내 가장 많은 화자 번호를 택하도록 하였다. 의사 코드는 아래와 같다.(그림 12)

```
set start_timestamp, end_timestamp from whisper

set start_index with start_timestamp
set end_index with end_timestamp

for speaker_id from speaker_id_list[start_index] to speaker_id_list[end_index]:
    if speaker_id not in dictionary then
        insert speaker_id in dictionary as key
        set dictionary value as 1
    else
        set dictionary[key=speaker_id].value plus 1

set selected_speaker_id to max of dictionary.value
```

그림 12. 의사 코드

```

predict start
/home/zxc6147/.local/lib/python3.10/site-packages/torch/nn/modules/linear.py:125: UserWarning: Attempting to
    return F.linear(input, self.weight, self.bias)
predict done : 4218.950757741928s
All predict done : 4218.950776576996s
/home/zxc6147/.local/lib/python3.10/site-packages/whisper/__init__.py:150: FutureWarning: You are using 'torch
    kle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main
    nctions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded vi
    rue' for any use case where you don't have full control of the loaded file. Please open an issue on GitHub fo
    checkpoint = torch.load(fp, map_location=device)
0.0$ 6.1$ 0 $ 안녕하세요 밥은 먹었나요
6.1$ 11.6$ 1 $ 날씨가 참 좋네요
11.6$ 16.8$ 1 $ 아니요 곧 먹을 예정이에요
16.8$ 23.9$ 0 $ 그렇다면 같이 먹지 않겠어요
23.9$ 29.9$ 0 $ 저는 선약이 있어서 먼저 가볼게요
zxc6147@zxc6147-MS-7B89: /mnt/mainStorage/Users/zxc61/Desktop/졸업과제/졸업과제용vscode/diarization$

```

그림 13. 녹음한 wav 파일(모델 A)
예측 결과

그림 13을 보자. 우리가 직접 음성 파일을 녹음하여 매핑에 대한 결과를 예측시켜 보았다. 이 음성 파일의 ground truth label은 0 0 1 0 1로 문장 5개 중 3개를 맞추었다. 이전의 mfcc window 한 개 한 개 비교한 것보다는 정확도가 향상되었지만 아직 상용 API에 비해 정확도가 부족하다. 또한 문장이 몇 개 없어 하나의 오차가 성능의 큰 부분을 좌우하기 때문에 신뢰도가 높지 않다고 여겨서 좀 더 긴 음성 파일을 녹음해서 예측시켜 보았다.


```
checkpoint = torch.load(fp, map_location=device)
0.0$ 8.3$ 0 $ 메이, 당신이 영화를 좋아하는 건 아는데, 그러면 제일 안 좋아하는 영화는 어떤 종류의 영화야?
10.2$ 16.5$ 1 $ 글썸, 더맨더머 같은 코미디 영화라고 말할 수 있겠지. 정말 잘 만들지 않고는 별로 쉽지 않거든.
17.7$ 20.8$ 1 $ 완다라는 이름의 물고기 같은 좋은 코미디 영화도 있어.
21.6$ 26.8$ 2 $ 오, 나도 그 영화 전에 봤는데 정말 멋진 영화였어.
26.8$ 30.9$ 3 $ 그래, 뛰어난 영화지. 정말 재밌었어.
32.5$ 37.0$ 4 $ 글썸, 그러면 영화 음악 중에서 가장 좋아하는 건 뭐야?
38.9$ 40.6$ 5 $ 이거 뭐야? 스모기 하는 거야?
42.1$ 45.3$ 4 $ 아냐, 그저 갑자기 궁금해서 그래.
46.8$ 51.1$ 6 $ 좋아, 영화 피아노의 주제 음악이 내가 좋아하는 영화 음악이야.
zxc6147@zxc6147-M5-7B89:/mnt/mainStorage/Users/zxc61/Desktop/졸업과제/졸업과제용vscode/diarization$
```

그림 14. 녹음한 wav 파일 예측 결과 2.

그림 14는 우리가 녹음한 또 다른 음성 파일 파일을 가지고 예측한 결과다. 위의 ground truth label은 0 1 1 0 1 0 1 이다. 같은 최적 적합(optimal matching)을 기준으로 9가지 중 4가지를 맞추었다. 4번 화자에서는 화자가 동일하다는 것을 측정해냈지만 이전의 화자(그림14의 0번, 2번)와 다르다고 판별해냈다.

```
a = scaled_dot_product_attention(
0.0$ 8.3$ 0 $ 메이, 당신이 영화를 좋아하는 건 아는데, 그러면 제일 안 좋아하는 영화는 어떤 종류의 영화야?
10.2$ 16.5$ 1 $ 글썸, 더맨더머 같은 코미디 영화라고 말할 수 있겠지. 정말 잘 만들지 않고는 별로 쉽지 않거든.
17.7$ 20.8$ 2 $ 완다라는 이름의 물고기 같은 좋은 코미디 영화도 있어.
21.6$ 26.8$ 3 $ 오, 나도 그 영화 전에 봤는데 정말 멋진 영화였어.
26.8$ 30.9$ 3 $ 그래, 뛰어난 영화지. 정말 재밌었어.
32.5$ 37.0$ 4 $ 글썸, 그러면 영화 음악 중에서 가장 좋아하는 건 뭐야?
38.9$ 40.6$ 5 $ 이거 뭐야? 스모기 하는 거야?
42.1$ 45.3$ 6 $ 아냐, 그저 갑자기 궁금해서 그래.
46.8$ 51.1$ 7 $ 좋아, 영화 피아노의 주제 음악이 내가 좋아하는 영화 음악이야.
```

그림 15. 75개의 파일을 개당 50회 반복하여 학습한 모델

정확도를 모델별 비교해보려는 목적으로 3개의 파일을 개당 1000회 반복하여 학습한 모델(그림 13, 14)(이하 모델 A)과 별개로 75개의 파일을 개당 50회 반복하여 학습한 모델(그림 15)(이하 모델 B)을 만들어보았다.

우리는 모델 A와 모델 B를 객관적으로 비교하기 위해 두 모델을 사용하여 동일한 데이터 3가지로 결과를 내보았다. 또한 한 모델에서의 음성 파일 길이별 예측 시간(단위 : 초)과 정확도, 문장별 정확도를 비교해보았다.(표 1)

여기서 정확도란 화자 번호 리스트에서 요소별로 최적 적합을 사용하여 비교한 정확도이고, 문장별 정확도는 whisper의 타임스탬프에 맞춰 구간별 가장 많은 화자 번호를 대표

화자번호로 하여 문장별로 최적 적합을 사용하여 비교한 정확도이다. 문장별 정확도 아래의 분수는 “맞춘 문장 개수 / 전체 문장 개수” 이다.

	모델 A			모델 B		
파일 길이	음성 파일 1	음성 파일 2	음성 파일 3	음성파일 1	음성 파일 2	음성 파일 3
30초	소요 시간:1479 정확도:0.145 문장별 정확도:0.600 (3/5)	소요 시간:1248 정확도:0.279 문장별 정확도:0.833 (5/6)	소요 시간:1203 정확도:0.197 문장별 정확도:0.800 (4/5)	소요 시간:2102 정확도:0.131 문장별 정확도: 0.400 (2/5)	소요 시간:1599 정확도:0.287 문장별 정확도:0.500 (3/6)	소요 시간:2009 정확도:0.115 문장별 정확도:0.400 (2/5)
45초	소요 시간:2606 정확도:0.153 문장별 정확도:0.500 (4/8)	소요 시간:2117 정확도:0.213 문장별 정확도:0.400 (4/10)	소요 시간:2354 정확도:0.204 문장별 정확도:0.625 (5/8)	소요 시간:3044 정확도:0.194 문장별 정확도: 0.500 (4/8)	소요 시간:2206 정확도:0.243 문장별 정확도: 0.400 (4/10)	소요 시간:4258 정확도:0.160 문장별 정확도:0.625 (5/8)
60초	소요 시간:3999 정확도:0.144 문장별 정확도:0.467 (7/15)	소요 시간:3258 정확도:0.203 문장별 정확도:0.385 (5/13)	소요 시간:3903 정확도:0.188 문장별 정확도:0.545 (6/11)	소요 시간: 5395 정확도: 0.115 문장별 정확도: 0.400 (6/15)	소요 시간: 3961 정확도: 0.166 문장별 정확도:0.308 (4/13)	소요 시간: 7096 정확도: 0.113 문장별 정확도:0.364 (4/11)

표 1. 모델A, 모델B의 결과 예측 소요시간, 정확도, 문장별 정확도에 따른 비교

4. 연구 결과 분석 및 평가

그래프 1. 파일 길이에 따른 예측 소요 시간

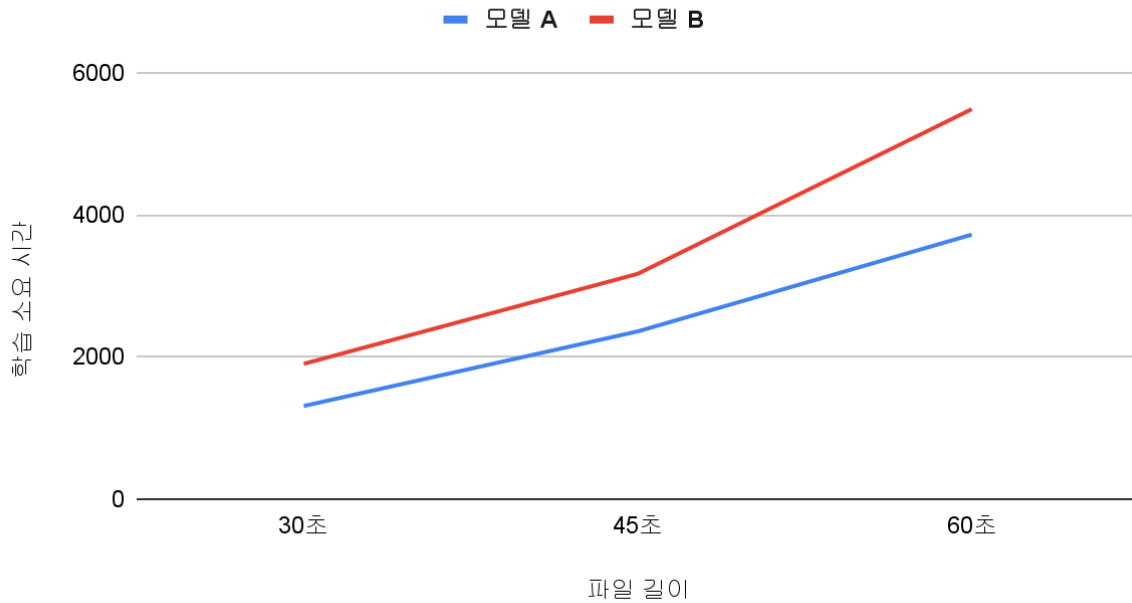
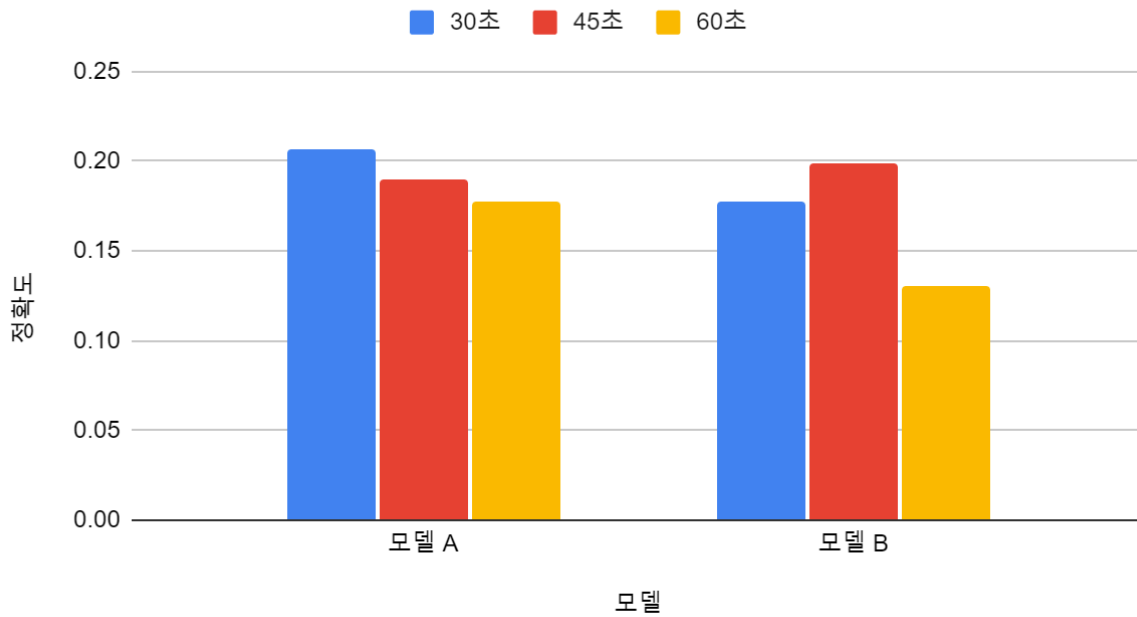


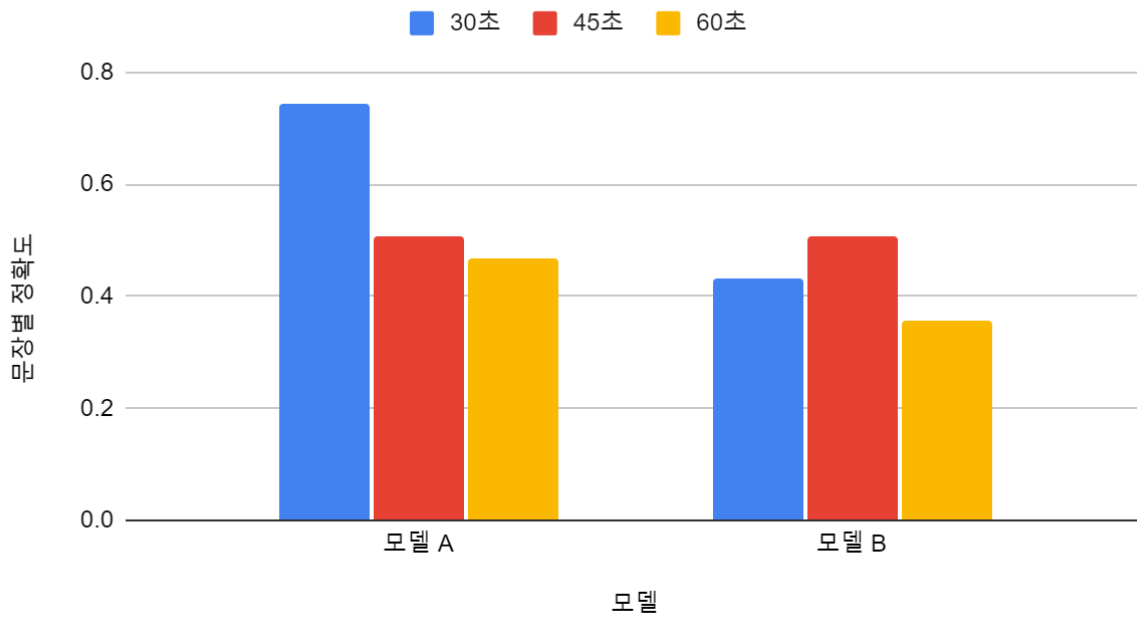
표1로부터 그래프 1을 그려보았다. 그래프 1을 보면 모델A, B 마찬가지로 파일 길이에 비례하여 학습 소요 시간이 증가함을 볼 수 있다.

모델 A가 모델 B보다 결과 예측 소요 시간이 짧았는데 30, 45, 60초 파일 길이별로 각각 31%, 26%, 32퍼센트 가량 짧았다.

그래프 2. 파일 길이에 따른 정확도

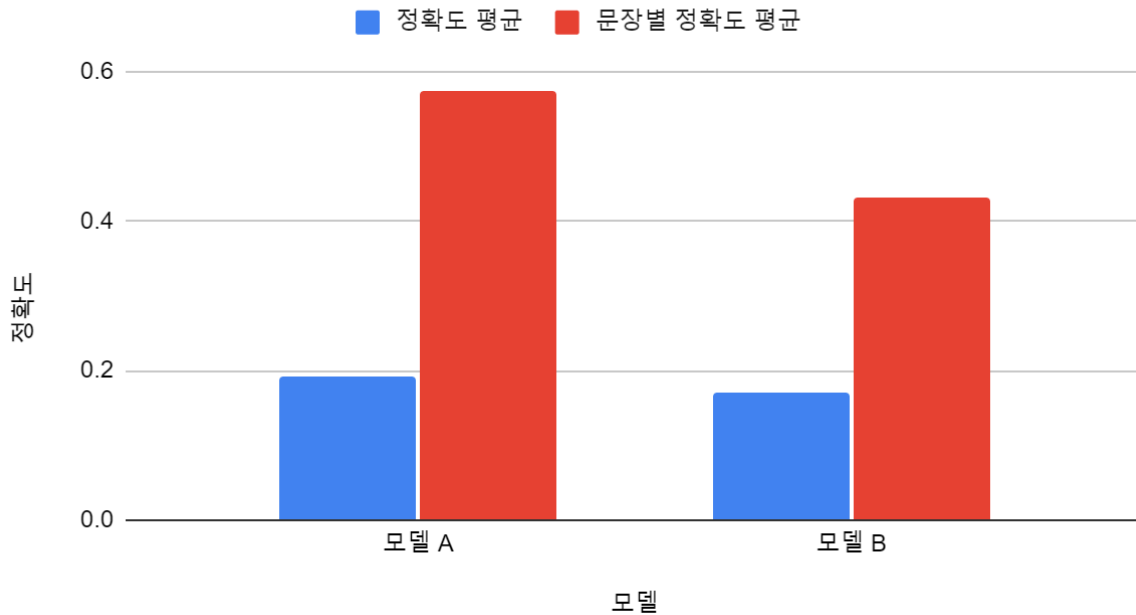


그래프 3. 파일 길이에 따른 문장별 정확도



그래프 2와 3을 보았을 때 정확도와 문장별 정확도는 어느정도 비례한다고 볼 수 있다.

그래프 4. 정확도 평균과 문장별 정확도 평균



그래프 4를 보았을 때, 정확도 평균보다 문장별 정확도 평균이 현저하게 높다는 것을 알 수 있다. 모델 A는 정확도 평균에 비해 문장별 정확도 평균이 2.99배였고, 모델 B는 2.55배였다.

이는 데이터 매핑 - 후처리가 중요하다는 것을 보여준다.

5. 결론 및 향후 연구 방향

이번 연구에서 **diarization api**들을 비교해보고 선정했다. 그리고 선정된 **api**를 통해 원천 음성 파일로부터 모델을 학습시켜서 성능을 비교했다.

아쉬운 점은 연구 내용에서 나왔듯이 학습이나 예측의 속도가 상용 애플리케이션에 비해 빠른 것도 아니었고 정확도가 높은 것도 아니었다. 그러나 단순히 속도나 정확도가 높은 **API**를 사용했으면 좋은 결과는 낼 수 있었겠지만, 다중화자 구별 전처리 및 학습 자체를 해 보며 이에 대한 이해도를 높인다는 경험을 해 보지는 못했을 것이다.

training 최적화 과정에서 **iteration** 수나 **learning rate**, 특징 벡터의 수 등 변인을 바꿔가며 최적화를 해야했지만 **training** 과 예측 과정에서 너무 오래 걸려 여러 변인을 바꾸어

실험하는데에 장애가 있었다.

특히 **N_MFCC** 같은 경우 특징 벡터 수에 관련된 직접적인 변수인 만큼 충분히 늘렸으면 정확도가 더 높았을 것으로 예상된다. 정확도가 낮은 것을 보아 데이터 전처리 같은 경우도 단순히 **librosa.load**나 **mfcc** 이외에 추가적인 처리가 필요했던 것으로 보인다.

향후 연구를 할 때에는 데이터 전처리에서 **i-vector**나 **d-vector embedding**과 같은 과정을 추가하여 연구를 이어나간다면, 정확도를 더 높일 수 있을 것으로 예상된다. 또한 화자의 수를 제한을 하는 등의 후처리 과정을 보강한다면, 정확도를 더 높일 수 있을 것으로 생각한다.

6. 구성원별 역할 및 개발 일정

6.1. 구성원별 역할

이름	역할
변상윤	배경지식 학습, 상용 API 비교 및 선정, 데이터 전처리, 다중화자 뼈대 구현, UIS-RNN 을 이용한 모델 학습, 보고서 작성, 발표 및 시연 준비
문성필	배경지식 학습, 상용 API 비교 및 선정, 다중화자 구현 방법 탐구, UIS-RNN 을 이용한 모델 학습, Flutter 애플리케이션 제작, 보고서 작성, 발표 및 시연 준비

6.2. 개발 일정

월	6					7					8					9					10	
주	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2
배경지식 학습																						
상용 API 비교 및 선정																						
다중화자 뼈대구현																						
데이터 전처리																						
UIS-RNN을 이용한 모델 학습																						
Flutter 애플리케이션 제작																						

7. 참고 문헌

- [1] UIS-RNN - <https://arxiv.org/pdf/1810.04719>, Aonan Zhang, Quan Wang, Zhenyao Zhu, John Paisley, Chong Wang, "FULLY SUPERVISED SPEAKER DIARIZATION"
- [2] Whisper - <https://arxiv.org/pdf/2212.04356>, Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever, "Robust Speech Recognition via Large-Scale Weak Supervision"
- [3]MFCC - <https://arxiv.org/pdf/1003.4083>, Lindasalwa Muda et al, Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping

(DTW) Techniques, JOURNAL OF COMPUTING, VOLUME 2, ISSUE 3, MARCH 2010, ISSN 2151-9617

[4]UISRNN API Documentation -

https://google.github.io/uis-rnn/uisrnn.html#UISRNN_fit_concatenated