Kaldi 설치 및 튜토리얼 한국어 가이드.



**Kaldi 개요와 시스템 요구사양**

**Kaldi란?**

분야 연구를 위해 만들어진 C++기반 음성인식 툴킷 입니다.

처음부터 관련 연구원, 음성인식 분야 연구하는 졸업생이나 고학년학부생 위해 만들어졌습니다.

**이름의 유래**

커피나무를 발견한 에티오피아목동의 이름에서 따왔습니다.

**필요사양**

Linux 주요 배포판등 Unix계열 OS (이전에 Windows에서도 테스트가능했으나 더 이상 업데이트하지 않는다고 합니다. Linux를 권장합니다.)

NvidiaGPU(권장) –신경망 교육에 사용하기 때문에 거의 반 필수 입니다. 없을 시 속도 차이가 매우 많이 납니다.

Ram: 4-gram으로 설정 시 60G 이상

VGA: 최소 1060 6G 이상

HDD: 최소 500G 이상의 여유 공간

단순 테스트시에는 이보다는 낮은 사양으로도 구동 가능하긴 합니다.

**필요 소프트웨어 패키지**

Git, wget, 표준Unix유틸리티(bash,perl,awk,grep,make등 대부분 unix기반 os에는 기본적으로 설치 되어있습니다.)

그외에는 Kaldi 설치하다 문제있으면 하나씩 받으라고 공식문서에서 알려주네요.

**튜토리얼 시작하기**

**1. 칼디 다운로드하기.**

- git clone https://github.com/kaldi-asr/kaldi.git

깃허브에서 kaldi를 다운로드 받습니다.

cd kaldi/tools/; make; cd ../src; ./configure; make

\*이과정에서 오류가 발생할 수 있지만 치명적이지 않다고 합니다. 일단 무시해도 된다고 합니다.

Tools/ 디렉토리란? : kaldi에 필요한 프로그램들 설치하는 곳. (make명령으로 설치한것들이 여기있습니다.)

openFst용으로 해당 디렉토리에 핵심코드가 많으니 전문가 분들은 잘 살펴보시기

바랍니다.

\*실제 진행중 mkl 오류 발생으로 ./configure; 안됨. mkl 을 꼭 설치해야합니다.

**2. 예제 스크립트 실행**

cd kaldi/egs

들어가면 README.txt 파일 보일꺼에요.

cd rm

들어가서 README.txt 파일을 살펴보면 전체구조를 확인할 수 있습니다.

cd s5

s5는 해당 툴킷이 버전 5 라는 뜻 입니다.

RESULTS 파일을 유심히 보세요. 나중에 결과를 받았을 때 이 파일을 비교해 봐야합니다.

Run.sh 파일도 여기 위치하고 있습니다.

**3. 데이터 준비**

-그리드 엔진 사용하고 있지 않은 경우.

train\_cmd="run.pl"

decode\_cmd="run.pl"

-그리드 엔진 사용하고 있는 경우.

train\_cmd="queue.pl -q all.q@a\*.clsp.jhu.edu"

decode\_cmd="queue.pl -q all.q@[ah]\*.clsp.jhu.edu"

\*그리드 엔진이란? 여러 컴퓨터를 묶어 사용해서 많은 데이터를 처리할 수 있는 시스템.

-리소스 매니지먼트 에서 테스트 및 corpora 훈련 세트 만들기.

\*corpora (말뭉치): 자연언어 연구를 위해 특정한 목적을 가지고 언어의 표본을 추출한 집합.

local/rm\_data\_prep.sh /export/corpora5/LDC/LDC93S3A/rm\_comp

만약 위 명령어 다돌아가면 "RM\_data\_prep succeeded" 라고 나오고 ‘data’ 라는 새 디렉토리가 생성.

data 폴더.

1. local: 데이터를 위한 사전.
2. train: 말뭉치를 훈련시킬 목적으로 세분화된 데이터.
3. test\_\*: 말뭉치를 테스트할 목적으로 세분화된 데이터.

\*train 폴더는 test\_\* 폴더와 같은 구조이지만 데이터 크기가 훨씬 큽니다.

-다음 명령어로 확인 할 수 있습니다.

cd data (디렉터리로 이동)

wc train/text test\_\*/text

-생성된 data 폴더의 train로 이동합니다.

cd data/train

-다음 커맨드를 입력 하세요

head text

head spk2gender

head spk2utt

head utt2spk

head wav.scp

명령어 개요.

1. text: 발화 하는 음소와 글자간의 맵핑 파일. 단어를 컴퓨터가 이해할 수 있는 숫자 형태로 바꿈.
2. spk2gender: 화자의 성별 맵핑.
3. utt2spk: 화자와 text파일의 ID와의 1:1 맵핑.
4. wav.scp: kaldi 에서 직접 읽는 파일. 읽기 전용으로 key-value 쌍으로 되어있다.

-kaldi에서 사용하는 원시데이터 만들기.

정수 형식의 텍스트 파일 만들어보자.

s5 디렉토리로 이동후 다음 명령어를 실행 합니다.

utils/prepare\_lang.sh data/local/dict '!SIL' data/local/lang data/lang

명령어를 실행하면 FST 포함된 로컬 폴더 안에 lang이라는 새 폴더가 생성됩니다.(/data/lang)

data/lang 안에는 words.txt, phones.txt 가 있을겁니다.

data/lang/phone 안에는 .csl 파일 있는데 무음상태 지정할 때 사용됩니다.

-L.fst 파일은 어휘목록 입니다. 안에잇는 정보 보려면 s5/ 디렉토리에서 다음을 실행하자

fstprint --isymbols=data/lang/phones.txt --osymbols=data/lang/words.txt data/lang/L.fst | head

만약 fstprint 명령어가 없다고 나오면 path에 추가해줘야 합니다.

-path.sh 를 실행하면 스크립트가 알아서 path에 추가해줍니다.

. ./path.sh

다음은 해당언어의 문법을 설명하는 FST를 만들어야 합니다.

-/s5 디렉토리로 이동하고 다음 명령을 입력하세요.

local/rm\_prepare\_grammar.sh

성공하면 "Succeeded preparing grammar for RM." 문구와 함께

/data/lang 폴더에 G.fst 파일 생성됩니다.

**Feature Extraction(특징 추출)**

다음단게는 훈련할 특징을 추출하는 것입니다. run.sh 를 열어 ‘mfcc’를 검색하고 해당하는 세 줄의

스크립트를 실행합니다. (특징 추출을 할 디렉토리를 지정해서 수정 해 줘야합니다.)

디렉토리에 많은 여유 공간이 필요할 수 있습니다.

예제에서는 /my/disk/rm\_mfccdir 로 설정했다고 가정합니다.

export featdir=/my/disk/rm\_mfccdir

# featdir이 존재하고 용량이 충분한지 확인 하세요.

# 원한다면 로컬에 실행할 수 있습니다.

mkdir $featdir

for x in test\_mar87 test\_oct87 test\_feb89 test\_oct89 test\_feb91 test\_sep92 train; do \

steps/make\_mfcc.sh --nj 8 --cmd "run.pl" data/$x exp/make\_mfcc/$x $featdir; \

steps/compute\_cmvn\_stats.sh data/$x exp/make\_mfcc/$x $featdir; \

done

여기서 --nj 8 구문은 cpu를 8개 사용할 것이라는 의미 입니다. cpu 수에 따라 조정할 수 있습니다.

로그파일의 위치는 exp/make\_mfcc/make\_mfcc.1.log 입니다.

steps/make\_mfcc.sh 스크립트에서 split\_scp.pl을 호출하는 라인을 보면 어떻게 동작하는지 짐작할

수 있습니다.

-다음을 입력하여 확인해 보세요.

wc $featdir/raw\_mfcc\_train.1.scp

wc data/train/wav.scp

다음으로 ‘scp’ 와 ‘ark’ 로 시작하는 파일을 살펴보겠습니다.

head data/train/wav.scp

head $featdir/raw\_mfcc\_train.1.scp

less $featdir/raw\_mfcc\_train.1.ark

.ark 파일은 바이너리 데이터가 포함되어 있으며 터미널에서 바로 작동하지 않으면 리셋 해야할

수 있습니다. 또한 .ark 파일은 상당히 클 수 있습니다.

다음을 입력해 이러한 파일을 더욱 편리하게 볼 수 있습니다.

-s5 디렉토리에서 다음을 실행하세요.

copy-feats ark:$featdir/raw\_mfcc\_train.1.ark ark,t:- | head

scp 파일도 동일하게 볼 수 있습니다.

copy-feats scp:$featdir/raw\_mfcc\_train.1.scp ark,t:- | head

-이 명령어를 통해 10번째 훈련 파일에서 일부 데이터를 출력해볼 수 있습니다.

head -10 $featdir/raw\_mfcc\_train.1.scp | tail -1 | copy-feats scp:- ark,t:- | head

\*kaldi는 스크립트파일 (.scp)과 아카이브파일 (.ark)을 똑같이 취급하며 두 파일은 모두 테이블로

구성되어 있습니다.

파이프 기호( | )를 활용해 아카이브와 스크립트 파일을 제어할 수 있습니다.

-다음 명령어를 입력해 보세요.

head -1 $featdir/raw\_mfcc\_train.1.scp | copy-feats scp:- ark:- | copy-feats ark:- ark,t:- | head

이제 편의를 위해 모든 테스트 데이터를 하나의 디렉토리로 병합 해 보겠습니다.

-다음 명령어를 입력하세요. (s5 디렉토리에서)

utils/combine\_data.sh data/test data/test\_{mar87,oct87,feb89,oct89,feb91,sep92}

steps/compute\_cmvn\_stats.sh data/test exp/make\_mfcc/test $featdir

또한 화자당 1000개의 발언만 유지하는 훈련 데이터(train.1k) 를 생성해 보겠습니다.

-다음 명령을 입력하세요.

utils/subset\_data\_dir.sh data/train 1000 data/train.1k

**Monophone training (모노폰 교육)**

다음 단게로 모노폰 모델을 훈련 시켜보겠습니다.

만약 kaldi를 설치한 디스크의 용량이 부족한 경우 다음 명령어로 다른 디스크에 exp/ 디렉토리를

생성 한 후 다음을 입력하세요.

nohup steps/train\_mono.sh --nj 4 --cmd "$train\_cmd" data/train.1k data/lang exp/mono &

다음을 입력 하면 가장 최근의 출력값을 볼 수 있습니다.

tail nohup.out

이 방법을 사용하면 연결이 끊어진 후에도 계속해서 실행을 완료할 수 있습니다.

하지만 더 좋은 방법은 역시 중간에 끊어지지 않고 쉘에서 끝까지 완료되는 것 입니다.

(exp/mono 디렉토리에 로그파일이 있습니다.)

.mdl 파일은 모델 파일 입니다.

모델 파일을 보겠습니다.

gmm-copy --binary=false exp/mono/0.mdl - | less

모델 파일을 열어보면 맨 위에 토폴로지 정보가 포함되어 있습니다.

그리고 모델 정보 앞에 다른 정보들도 있는 것을 볼 수 있습니다.

\*파일은 바이너리 또는 텍스트 형식이며 –binary=false 또는 true 로 제어할 수 있습니다.

이제 kaldi의 트리파일을 보겠습니다.

copy-tree --binary=false exp/mono/tree - | less

이 파일은 ‘tree’ 형식 으로 분할이 없이 매우 직관적입니다.

이제 exp/mono/ali.1.gz 파일을 봅시다.(이전에 충분히 훈련시켰다면 존재할 것입니다.)

copy-int-vector "ark:gunzip -c exp/mono/ali.1.gz|" ark,t:- | head -n 2

이것은 훈련 데이터의 비터비 정렬 입니다.

이 정렬을 좀더 읽기 쉽게 보려면 다음을 입력하세요.

show-alignments data/lang/phones.txt exp/mono/0.mdl "ark:gunzip -c exp/mono/ali.1.gz |" | less

다음으로 훈련이 어떻게 진행되고 있는지 살펴보겠습니다. (쉘이 bash라고 가정합니다.)

grep Overall exp/mono/log/acc.{?,??}.{?,??}.log

exp/mono/log/update.\*.log 파일중 하나를 보면 업데이트 로그에 어떤 정보가 있는지

확인할 수 있습니다.

모노폰 훈련이 긑나면 디코딩을 테스트 할 수 있는데요.

그전에 디코딩 그래프를 생성해 봅시다.

utils/mkgraph.sh --mono data/lang exp/mono exp/mono/graph

utils/mkgraph.sh 가 호출하는 프로그램에는 fst로 시작하는 프로그램이 많이 있습니다.

이것들중 대부분은 우리가 위의 과정들을 진행하면서 만든 파일들 입니다.

이러한 프로그램들의 위치는 다음과 같이 확인 가능 합니다.

utils/mkgraph.sh 에서 fstdeterminizestar 의 위치를 찾는다고 가정합니다.

which fstdeterminizestar

그래프 생성 프로세스 후 다음명령어로 디코딩을 시작할 수 있습니다.

steps/decode.sh --config conf/decode.config --nj 20 --cmd "$decode\_cmd" \

exp/mono/graph data/test exp/mono/decode

디코딩된 출력의 일부를 보는 방법은 다음 과 같습니다.

less exp/mono/decode/log/decode.2.log

이제 화면에 대본이 표시됩니다.(짝짝짝)

이 프로그램의 실제 출력은 exp/mono/decode/scoring/2.tra 에서 확인할 수 있습니다.

tra 파일에서 실제 디코딩된 단어 시퀀스를 보려면 다음을 입력합니다.

(언어모델 척도에 따라 2~13 사이의 LM 스케일을 사용할 수 있습니다. test는 2.tra를 사용합니다.)

utils/int2sym.pl -f 2- data/lang/words.txt exp/mono/decode/scoring/2.tra

sym2int.pl 이라는 스크립트를 정수형으로 변환하려면 다음 명령어를 입력하세요.

utils/int2sym.pl -f 2- data/lang/words.txt exp/mono/decode/scoring/2.tra | \

utils/sym2int.pl -f 2- data/lang/words.txt

-f 2- 옵션은 id 를 정수로 변환하지 않도록 하는것입니다. 다음을 입력해 보세요.

tail exp/mono/decode/log/decode.2.log

이제 디코딩한 정보를 출력해 봅시다.

gmm-decode-faster

**Kaldi for Dummies 튜토리얼**

**소개**

이제 자신의 데이터 세트를 사용하여 kaldi에서 간단한 ASR 시스템을 만들어봅시다.

이 장에서는 Kaldi를 설치하고 작동하게하며 오디오 데이터를 사용하여 ASR시스템을

실행하는 방법을 배우게 됩니다.

음성 인식 및 ASR시스템에 관한 확실한 지식이 필요합니다.

또 스크립트 프로그래밍언어(bash, perl, python)의 기초를 아는 것도 도움이 됩니다.

**환경**

Linux Ubuntu 14.10 을 기준으로 설명합니다. 다음의 항목들을 설치하세요.

* atlas – 선형 대수학 계산 자동화 및 최적화
* autoconf – 다른 운영체제 에서 자동 소프트웨어 컴파일.
* automake – Makefile 파일 생성
* git – 버전 관리 시스템.
* libtool – 정적 및 동적 라이브러리 생성.
* svn – kaldi 다운로드 및 설치에 필요한 버전 관리 시스템.
* wget – HTTP,HTTPS 및 FTP 프로토콜을 사용한 데이터 전송.
* zlib – 데이터 압축

Unix 기반이라면 아마 설치되어있을 항목들. (없으면 설치하세요.)

-awk, bash, grep, make, perl

**Kaldi 다운로드**

페이지 3 을 참조하세요.

Kaldi 디렉토리에 다음 항목들이 있는지 확인해 봅시다.

* egs – 30개 이상의 말뭉치로 ASR 시스템을 빠르게 구축할 수 있는 예제 스크립트.
* misc – 추가적인 툴들입니다. 꼭 필요하지 않을 수 있습니다.
* src – Kaldi 소스코드가 들어있습니다.
* tools – 유용한 컴포넌트와 외부 툴들입니다.
* windows – Windows를 사용하여 Kaldi를 실행하기 위한 도구입니다. (무쓸모)

**필요한 자료**

여러 명의 다른 화자가 숫자만 말하는 오디오 데이터 (예: 하나, 둘, 셋)

**우리의 목적**

* 데이터를 훈련세트와 테스트 세트로 나눠봅니다.
* ASR 시스템을 설정합니다.
* ASR 시스템을 훈련시킵니다.
* 테스트 해봅니다.
* 디코딩하여 결과를 얻습니다.

**시작 하기**

kaldi/egs/ 에 digits 폴더를 만듭니다.

mkdir kaldi/egs/digits

**데이터 준비**

**오디오 데이터**

우리가 가지고 있는 오디오 데이터를 기반으로 ASR 시스템을 설정 해 봅시다.

예를 들어 100개의 파일이 있다고 가정해봅시다. 모든 파일 형식은 WAV 입니다.

각 파일에는 영어로 녹음된 3개의 음성 숫자가 녹음되어 있습니다.

파일의 이름을 인식 가능한 방식으로 세팅합니다.

(예: 일 삼 사 라고 말하는 오디오 파일이라면 1\_3\_4.wav)

또 같은 화자라도 다른환경이나 세팅에서 말했다면 폴더를 나눠 정렬 합니다.

에를들어 이런 식으로 파일들을 세팅할 수 있습니다.

* 10명의 다른 화자(화자가 많을수록 좋습니다.)
* 각 화자는 10개의 문장을 말합니다.
* 100개의 문장 파일(특정 화자와 관련된 10개의 폴더에 각 10개의 wav파일 배치)
* 0~9까지의 숫자를 말하는 300개의 단어 세트.
* 각 파일은 3개의 단어로 구성됩니다.

kaldi/egs/digits 디렉토리로 이동하여 digits\_audio 폴더를 만듭니다.

mkdir digits\_audio

그안에 또 train과 test 폴더를 만드세요.(disgits\_audio 폴더에 있다고 가정합니다.)

mkdir train

cd ../

mkdit test

먼저 10명의 화자별로 분류된 각각의 폴더가 있다고 가정합니다.

kaldi/egs/digits/digits\_audio/test 디렉토리안에 화자의 이름으로 폴더를 하나 만듧니다.

여기선 화자의 이름을 jisang 으로 가정합니다.

mkdir jisang

그후 그 안에 그 사람과 관련된 모든 오디오 파일을 넣으세요.

나머지 9명의 화자에 관한 데이터는 train 폴더에 넣습니다.

이 파일들은 훈련 데이터 세트로 활용 될 것입니다.

**음향 데이터**

이제 kaldi가 오디오 데이터와 통신할 수 있도록 몇가지 텍스트 파일을 만들어야 합니다.

kaldi/egs/digits 디렉토리로 들어가서 data 폴더를 만듧니다.

mkdir data

그 후 data 내부에 test 및 train 폴더를 만듧니다. (이전에 오디오 데이터 세트와 이름이 같습니다.)

cd data

mkdir test

cd ../

mkdir train

**spk2gender**

-화자의 성별을 알려줍니다.

-예를들어 다음과 같이 정의 할 수 있습니다.

패턴: <speakerID><성별>

cristine f

dad m

josh m

july f

# and so on...

\*f 는 female m 은 male 을 의미 합니다.

**wav.scp**

이 파일은 특정 녹음 세션동안 한 사람이 말한 문장에 대한 모든 발화를 관련된 오디오 파일과

연결합니다. 예를들면 다음과 같습니다.

패턴: <발화ID><전사내용>

dad\_4\_4\_2 four four two

july\_1\_2\_5 one two five

july\_6\_8\_3 six eight three

# and so on...

**corpus.txt**

이 파일은 kaldi/egs/digits/data 폴더 안에 local 폴더를 만든 후 만들어야 합니다.

ASR 시스템에서 발생할 수 있는 모든 단일 발화 전사를 포함해야 합니다.

패턴:<전사내용>

one two five

six eight three

four four two

# and so on...

**언어 데이터**

**lexicon.txt**

이 파일에는 사전 정의한 모든 단어들이 발음 기호와 함께 들어있습니다.

(/egs/voxforge 에서 가져오세요.)

패턴:<단어> <발음1><발음2>…

!SIL sil

<UNK> spn

eight ey t

five f ay v

four f ao r

nine n ay n

one hh w ah n

one w ah n

seven s eh v ah n

six s ih k s

three th r iy

two t uw

zero z ih r ow

zero z iy r ow

**nonsilence\_phones.txt**

이 파일은 프로젝트에 있는 발음들을 나타냅니다.

패턴: <발음>

ah

ao

ay

eh

ey

f

hh

ih

iy

k

n

ow

r

s

t

th

uw

w

v

z

**silence\_phone.txt**

이 파일은 발음에서의 묵음을 나타냅니다.

패턴:<발음>

sil

**Tool 추가하기.**

kaldi/egs/swj/s5 에서 utils와 steps두개의 폴더를 우리가 만든kaldi/egs/digits 에 복사하는 것 과

같이 tool들을 추가할 수 있습니다.

**스코어링 스크립트**

이 스크립트는 디코딩 결과를 얻는데 도움이 됩니다.

kaldi/egs/voxforge/s5/local 에서 score.sh를 복사한 후 우리가 만든 프로젝트의 같은 위치에

넣을 수 있습니다. (예:kaldi/egs/digits/local)

**SRILM설치**

언어 모델 툴킷인 SRILM을 설치해 봅시다.

자세한 내용은 kaldi/tools/install\_srilm.sh 를 참조하세요.

**설정 파일 생성.**

kaldi/egs/digits 안에 conf 폴더를 만듧니다. 그안에 decode.config 파일과 mfcc.conf 파일을 만듧니다.

decode.config

first\_beam=10.0

beam=13.0

lattice\_beam=6.0

mfcc.conf

--use-energy=false

**스크립트 생성 및 실행**

이제 kaldi를 활용해 ASR시스템을 거의 구축했습니다.

마지막 작업은 스크립트를 준비하는 것입니다.

이 스크립트는 /egs/voxforge 디렉토리에서 사용되는 솔루션을 기반으로 합니다.

여기서는 두가지의 훈련방법을 사용합니다.

* MONO – 모노폰 트레이닝
* TRI1 – 간단한 트라이폰 트레이닝

작업

kaldi/egs/digits 디렉토리에서 3개의 스크립트를 생성합니다.

-cmd.sh

#로컬 시스템 작업 설정

export train\_cmd=run.pl

export decode\_cmd=run.pl

-path.sh

#kaldi 루트 디렉토리 정의

export KALDI\_ROOT = ‘pwd’ /../..

#유용한 툴들의 경로.

export PATH=$PWD/utils/:$KALDI\_ROOT/src/bin:$KALDI\_ROOT/tools/openfst/bin:$KALDI\_ROOT/src/fstbin/:$KALDI\_ROOT/src/gmmbin/:$KALDI\_ROOT/src/featbin/:$KALDI\_ROOT/src/lmbin/:$KALDI\_ROOT/src/sgmm2bin/:$KALDI\_ROOT/src/fgmmbin/:$KALDI\_ROOT/src/latbin/:$PWD:$PATH

#오디오 데이터 디렉토리 정의

export DATA\_ROOT="/home/{user}/kaldi/egs/digits/digits\_audio"

#SRILM 활성화

. $KALDI\_ROOT/tools/env.sh

#데이터 정렬을 위해 필요한 변수.

export LC\_ALL=C

-run.sh

#!/bin/bash

. ./path.sh || exit 1

. ./cmd.sh || exit 1

nj=1 # 병렬 작업 수 데이터 많아지면 숫자 올려야 함.

lm\_order=1 # n-gram 수량 숫자 문법은 1이면 충분. 보통 3~4추천

# 안전 메커니즘.

. utils/parse\_options.sh || exit 1

[[ $# -ge 1 ]] && { echo "Wrong arguments!"; exit 1; }

# 이전에 생성된 데이터 제거

rm -rf exp mfcc data/train/spk2utt data/train/cmvn.scp data/train/feats.scp data/train/split1 data/test/spk2utt data/test/cmvn.scp data/test/feats.scp data/test/split1 data/local/lang data/lang data/local/tmp data/local/dict/lexiconp.txt

echo

echo "===== PREPARING ACOUSTIC DATA ====="

echo

# Needs to be prepared by hand (or using self written scripts):

#

# spk2gender [<speaker-id> <gender>]

# wav.scp [<uterranceID> <full\_path\_to\_audio\_file>]

# text [<uterranceID> <text\_transcription>]

# utt2spk [<uterranceID> <speakerID>]

# corpus.txt [<text\_transcription>]

# spk2utt 파일 만들기

utils/utt2spk\_to\_spk2utt.pl data/train/utt2spk > data/train/spk2utt

utils/utt2spk\_to\_spk2utt.pl data/test/utt2spk > data/test/spk2utt

echo

echo "===== FEATURES EXTRACTION ====="

echo

# feats.scp 파일 만들기

mfccdir=mfcc

# 데이터 정렬에 문제가 있는경우 아래 스크립트의 주석 처리를 제거하고 수정하세요.

# utils/validate\_data\_dir.sh data/train # 준비된 데이터 확인하는 명령- here: for data/train directory

# utils/fix\_data\_dir.sh data/train # 필요하다면 적절하게 정렬시키는 명령- here: for data/train directory

steps/make\_mfcc.sh --nj $nj --cmd "$train\_cmd" data/train exp/make\_mfcc/train $mfccdir

steps/make\_mfcc.sh --nj $nj --cmd "$train\_cmd" data/test exp/make\_mfcc/test $mfccdir

# cmvn.scp 파일 만들기

steps/compute\_cmvn\_stats.sh data/train exp/make\_mfcc/train $mfccdir

steps/compute\_cmvn\_stats.sh data/test exp/make\_mfcc/test $mfccdir

echo

echo "===== PREPARING LANGUAGE DATA ====="

echo

# Needs to be prepared by hand (or using self written scripts):

#

# lexicon.txt [<word> <phone 1> <phone 2> ...]

# nonsilence\_phones.txt [<phone>]

# silence\_phones.txt [<phone>]

# optional\_silence.txt [<phone>]

# 언어 데이터 준비

utils/prepare\_lang.sh data/local/dict "<UNK>" data/local/lang data/lang

echo

echo "===== LANGUAGE MODEL CREATION ====="

echo "===== MAKING lm.arpa ====="

echo

loc=`which ngram-count`;

if [ -z $loc ]; then

if uname -a | grep 64 >/dev/null; then

sdir=$KALDI\_ROOT/tools/srilm/bin/i686-m64

else

sdir=$KALDI\_ROOT/tools/srilm/bin/i686

fi

if [ -f $sdir/ngram-count ]; then

echo "Using SRILM language modelling tool from $sdir"

export PATH=$PATH:$sdir

else

echo "SRILM toolkit is probably not installed.

Instructions: tools/install\_srilm.sh"

exit 1

fi

fi

local=data/local

mkdir $local/tmp

ngram-count -order $lm\_order -write-vocab $local/tmp/vocab-full.txt -wbdiscount -text $local/corpus.txt -lm $local/tmp/lm.arpa

echo

echo "===== MAKING G.fst ====="

echo

lang=data/lang

arpa2fst --disambig-symbol=#0 --read-symbol-table=$lang/words.txt $local/tmp/lm.arpa $lang/G.fst

echo

echo "===== MONO TRAINING ====="

echo

steps/train\_mono.sh --nj $nj --cmd "$train\_cmd" data/train data/lang exp/mono || exit 1

echo

echo "===== MONO DECODING ====="

echo

utils/mkgraph.sh --mono data/lang exp/mono exp/mono/graph || exit 1

steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode\_cmd" exp/mono/graph data/test exp/mono/decode

echo

echo "===== MONO ALIGNMENT ====="

echo

steps/align\_si.sh --nj $nj --cmd "$train\_cmd" data/train data/lang exp/mono exp/mono\_ali || exit 1

echo

echo "===== TRI1 (first triphone pass) TRAINING ====="

echo

steps/train\_deltas.sh --cmd "$train\_cmd" 2000 11000 data/train data/lang exp/mono\_ali exp/tri1 || exit 1

echo

echo "===== TRI1 (first triphone pass) DECODING ====="

echo

utils/mkgraph.sh data/lang exp/tri1 exp/tri1/graph || exit 1

steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode\_cmd" exp/tri1/graph data/test exp/tri1/decode

echo

echo "===== run.sh script is finished ====="

echo

**결과**

이제 run.sh 스크립트를 실행하기만 하면 됩니다. 이 튜토리얼에서 실수를 한 경우

터미널의 로그가 이를 해결하는 방법을 제시합니다.

**데이터 준비**

예제 스크립트를 실행한 후 자신이 가진 데이터로 kaldi를 실행해봅시다.

이 섹션에서는 데이터를 준비하는 방법에 대해 설명합니다.

s5내에 /local 디렉토리에는 데이터 베이스가 있습니다.

예를 들어 리소스매니저(RM)의 설정은 local/rm\_data\_prep.sh 를 보면 됩니다.

RM에 관한 커맨드를 이렇게 사용할 수 있습니다.

local/rm\_data\_prep.sh /export/corpora5/LDC/LDC93S3A/rm\_comp || exit 1;

utils/prepare\_lang.sh data/local/dict '!SIL' data/local/lang data/lang || exit 1;

local/rm\_prepare\_grammar.sh || exit 1;

WSJ의 경우에는 이렇게 사용할 수 있습니다.

wsj0=/export/corpora5/LDC/LDC93S6B

wsj1=/export/corpora5/LDC/LDC94S13B

local/wsj\_data\_prep.sh $wsj0/??-{?,??}.? $wsj1/??-{?,??}.? || exit 1;

local/wsj\_prepare\_dict.sh || exit 1;

utils/prepare\_lang.sh data/local/dict "<SPOKEN\_NOISE>" data/local/lang\_tmp data/lang || exit 1;

local/wsj\_format\_data.sh || exit 1;

데이터를 준비하는 것은 크게 데이터와 언어로 나눌 수 있습니다.

데이터는 data/train 디렉토리에 저장되며 우리가 가진 녹음 파일들과 연관되어 있습니다.

언어는 data/lang 디렉토리에 저장되며 어휘, 발화와 같이 언어 자체와 관련된 것들이 담겨 있습니다.

이미 존재하는 시스템과 이미 존재하는 언어모델로 디코딩 할 데이터를 준비한다면 그것들을 불러온

후 데이터 부분만 추가할 수 있습니다.

**데이터 부분 준비**

데이터 파트에 관한 내용은data/train 부분에 담겨있습니다. (스크립트 실행 후 생성 됩니다.)

예를들어 egs/swbd/s5 폴더를 살펴보겠습니다.

s5# ls data/train

cmvn.scp feats.scp reco2file\_and\_channel segments spk2utt text utt2spk wav.scp

모든 파일이 위와 같을 필요는 없습니다. 만약 ‘utt2spk’, ‘text’, ‘wav.scp’ 파일이 없다면 만들어야 할 수도

있습니다. (segments, reco2file\_and\_channel 파일도 필수는 아니지만 가능하다면 만들 수 있습니다.)

**직접 생성해야하는 파일**

**‘text’ 파일**

s5# head -3 data/train/text

sw02001-A\_000098-001156 HI UM YEAH I'D LIKE TO TALK ABOUT HOW YOU DRESS FOR WORK AND

sw02001-A\_001980-002131 UM-HUM

sw02001-A\_002736-002893 AND IS

각 줄의 첫번째 요소는 임의의 텍스트 문자열인 utterance-id 이지만 화자 정보가 있는경우 ID를

utterance-id의 접두사로 만들어야 합니다. 나머지 줄은 각 문장의 필사본 입니다.

모든 단어가 어휘에 정확하게 맞는지 확인할 필요는 없으며, 단어는 data/lang/oov.txt 파일에 지정된

단어로 맵핑된 것입니다.

utt2spk 파일과 spk2utt 파일은 순서가 동일해야 합니다.

예를들어 speaker-ids 리스트는 문자열의 정렬 순서와 동일해야 합니다.

그렇게 하려면 speaker-ids를 utter의 식별번호로 만들면 됩니다.

위의 예에서는 speaker와 utterane 부분을 구분하기 위해 밑줄을 사용했지만 일반적으로는

‘-‘ 기호를 사용하는 것이 더 나을 수 있습니다. (ASCII 코드값이 더 낮음)

**wav.scp 파일**

이 파일의 형식은 다음과 같습니다.

<recording-id> <extended-filename>

예들들어 이런 식으로 사용합니다.

s5# head -3 data/train/wav.scp

sw02001-A /home/dpovey/kaldi-trunk/tools/sph2pipe\_v2.5/sph2pipe -f wav -p -c 1 /export/corpora3/LDC/LDC97S62/swb1/sw02001.sph |

sw02001-B /home/dpovey/kaldi-trunk/tools/sph2pipe\_v2.5/sph2pipe -f wav -p -c 2 /export/corpora3/LDC/LDC97S62/swb1/sw02001.sph |

첫번째 부분은 recording-id 입니다. 화자의 id를 넣으면됩니다.

<extended-filename> 부분은 실제 파일의 이름이거나 wav형식 파일을 추출하는 명령일 수 있습니다.

wav.scp 파일은 단일 채널(mono) 이어야합니다. wav파일이 여러 채널로 되어있는 경우 sox 명령을

사용해 특정 채널을 추출 해야 합니다.

**segments 파일**

segments 파일 형식은 다음과 같습니다.

<utterance-id> <recording-id> <segment-begin> <segment-end>

이런식으로 사용할 수 있습니다.

s5# head -3 data/train/segments

sw02001-A\_000098-001156 sw02001-A 0.98 11.56

sw02001-A\_001980-002131 sw02001-A 19.8 21.31

sw02001-A\_002736-002893 sw02001-A 27.36 28.93

<utterance-id>의 접두사는 위의 wav.scp 파일의 <recording-id>와 동일합니다.

그다음 숫자들은 세그먼트의 시작과 종료 시간을 의미합니다.

**reco2file\_and\_channel 파일**

이 파일은 NIST의 sclite의 도구로 오류율을 측정할때만 사용됩니다.

다음과 같은 형식을 가지며

<recording-id> <filename> <recording-side (A or B)>

다음과 같이 사용할 수 있습니다.

s5# head -3 data/train/reco2file\_and\_channel

sw02001-A sw02001 A

sw02001-B sw02001 B

sw02005-A sw02005 A

<recording-side>는 2개의 채널이 있는 경우가 아니라면 A를 사용합니다.

stm 파일이 없거나 이것이 무엇인지 모른다면 reco2file\_and\_channel 파일은 필요없습니다.

**utt2spk 파일**

직접 생성해야하는 마지막 파일 입니다.

다음의 형식을 가집니다.

<utterance-id> <speaker-id>

이렇게 사용할 수 있습니다.

s5# head -3 data/train/utt2spk

sw02001-A\_000098-001156 2001-A

sw02001-A\_001980-002131 2001-A

sw02001-A\_002736-002893 2001-A

<speaker-id> 는 반드시 정확하게 매칭될 필요는 없습니다.

대화가 조금 섞여 있어도 됩니다. (예를 들어 두 명이 말했지만 한명이 말한것으로 지정해도 됩니다.)

<speaker-id>를 가지고 있지 않다면 대신 <utterance-id>를 사용할 수 있습니다.

<utterance-id> <utterance-id>

이렇게요.

kaldi시스템이 빌드하는데는 사용되지 않지만 spk2gender과 같은 파일도 있습니다.

s5# head -3 ../../rm/s5/data/train/spk2gender

adg0 f

ahh0 m

ajp0 m

단순히 화자의 성별을 알려주는 (f: female,m: male) 파일 입니다.

파일들을 모두 준비했다면 우리는 이 파일들을 잘 정렬해야합니다. 그렇지 않으면 스크립트를 실행할 때

오류가 발생합니다.

**직접 만들 필요가 없는 파일**

**spk2utt 파일**

이 파일은 egs/rm/s5/local/rm\_data\_prep.sh 에서 추출됩니다.

utils/utt2spk\_to\_spk2utt.pl data/train/utt2spk > data/train/spk2utt

**feats.scp 파일**

steps/make\_mfcc.sh --nj 20 --cmd "$train\_cmd" data/train exp/make\_mfcc/train $mfccdir

**cmvn.scp**

화자가 인덱싱한 cepstral 평균 및 분산 정규화에 대한 통계 입니다.

steps/compute\_cmvn\_stats.sh data/train exp/make\_mfcc/train $mfccdir

이렇게 데이터를 준비한 후 디렉터리에 올바르게 배치 되었는지 확인해봅시다.

utils/validate\_data\_dir.sh data/train

문제가 있다면 다음 명령을 사용해 봅시다.

utils/fix\_data\_dir.sh data/train

이제 우리는 kaldi 를 설치하고 테스트하고 다른 사람이 만들어 놓은 모델을 활용하고 내가 가진 음성 데이터를

kaldi가 학습할 수 있도록 준비하는 방법을 배웠습니다.