# TIMIT 数据集_声纹识别_kaldi

## 调整数据分布和格式

- 原始数据分布

TIMIT/DOC

/TEST/DR{1-8}/SPK/{wav}

/TRAIN/DR{1-8}/SPK/{wav}

README.DOC

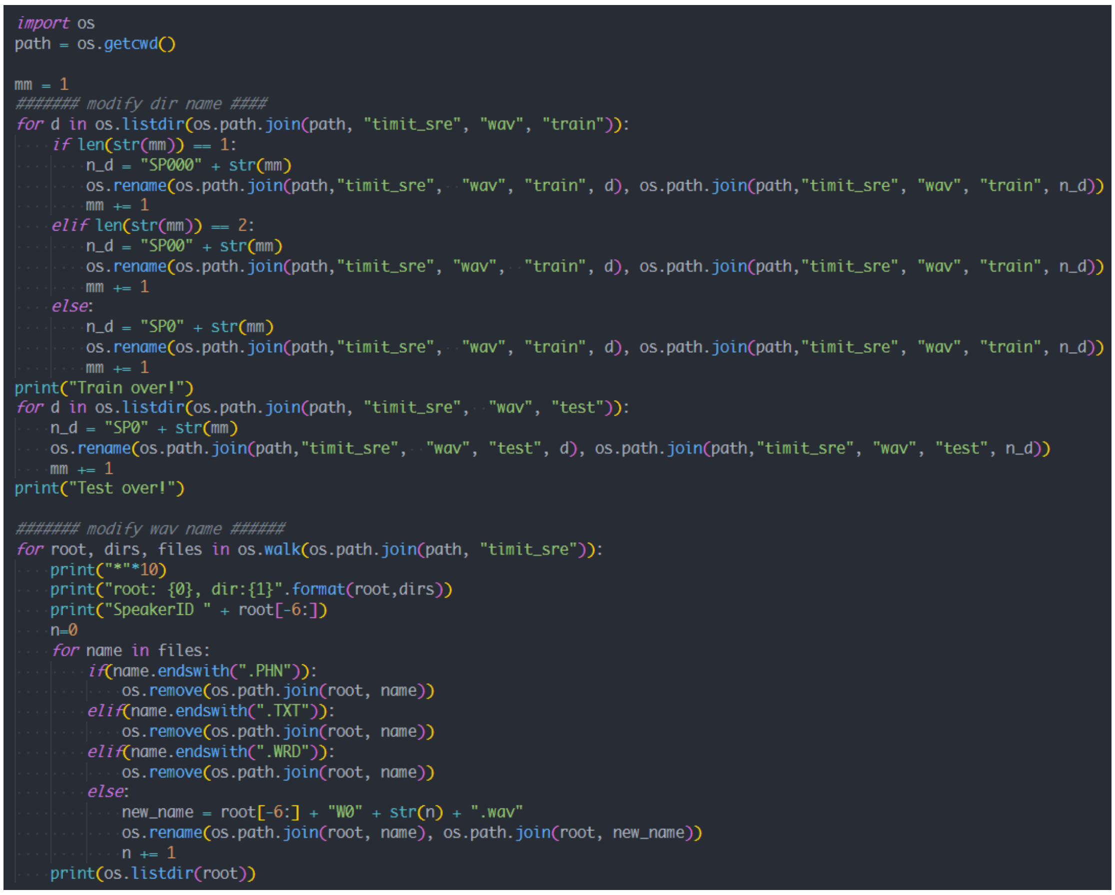
这里的 SPK 是类似“FAKS0”，“MDAB0”这样的名称。

- 借鉴 AISHELL.v1 中的文件分布，只需要获取 TRAIN 和 TEST 两个文件夹，分别把 DR 里的说话人文件都放在一个目录下（train 和 test 目录下），可以手动改为

timit/wav/train/SPK{1-462}/{wav}

/test/SPK{1-168}/{wav}

- 为了方便观察中间过程和结果，最好修改一下文件名和 wav 名，可以再删去一些不用的文件。可以调用 modify_name.py



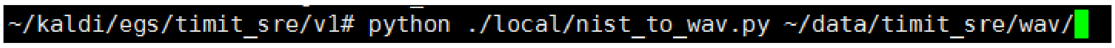
```python
import os
path = os.getcwd()

mm = 1
####### modify dir name ####
for d in os.listdir(os.path.join(path, "timit_sre", "wav", "train")):
    if len(str(mm)) == 1:
        n_d = "SP000" + str(mm)
        os.rename(os.path.join(path,"timit_sre",  "wav", "train", d), os.path.join(path,"timit_sre", "wav", "train", n_d))
        mm += 1
    elif len(str(mm)) == 2:
        n_d = "SP00" + str(mm)
        os.rename(os.path.join(path,"timit_sre", "wav",  "train", d), os.path.join(path,"timit_sre", "wav", "train", n_d))
        mm += 1
    else:
        n_d = "SP0" + str(mm)
        os.rename(os.path.join(path,"timit_sre",  "wav", "train", d), os.path.join(path,"timit_sre", "wav", "train", n_d))
        mm += 1
print("Train over!")
for d in os.listdir(os.path.join(path, "timit_sre",  "wav", "test")):
    n_d = "SP0" + str(mm)
    os.rename(os.path.join(path,"timit_sre",  "wav", "test", d), os.path.join(path,"timit_sre", "wav", "test", n_d))
    mm += 1
print("Test over!")

####### modify wav name ######
for root, dirs, files in os.walk(os.path.join(path, "timit_sre")):
    print("*"*10)
    print("root: {0}, dir:{1}".format(root,dirs))
    print("SpeakerID " + root[-6:])
    n=0
    for name in files:
        if(name.endswith(".PHN")):
            os.remove(os.path.join(root, name))
        elif(name.endswith(".TXT")):
            os.remove(os.path.join(root, name))
        elif(name.endswith(".WRD")):
            os.remove(os.path.join(root, name))
        else:
            new_name = root[-6:] + "W0" + str(n) + ".wav"
            os.rename(os.path.join(root, name), os.path.join(root, new_name))
            n += 1
    print(os.listdir(root))
```

最后的文件分布为

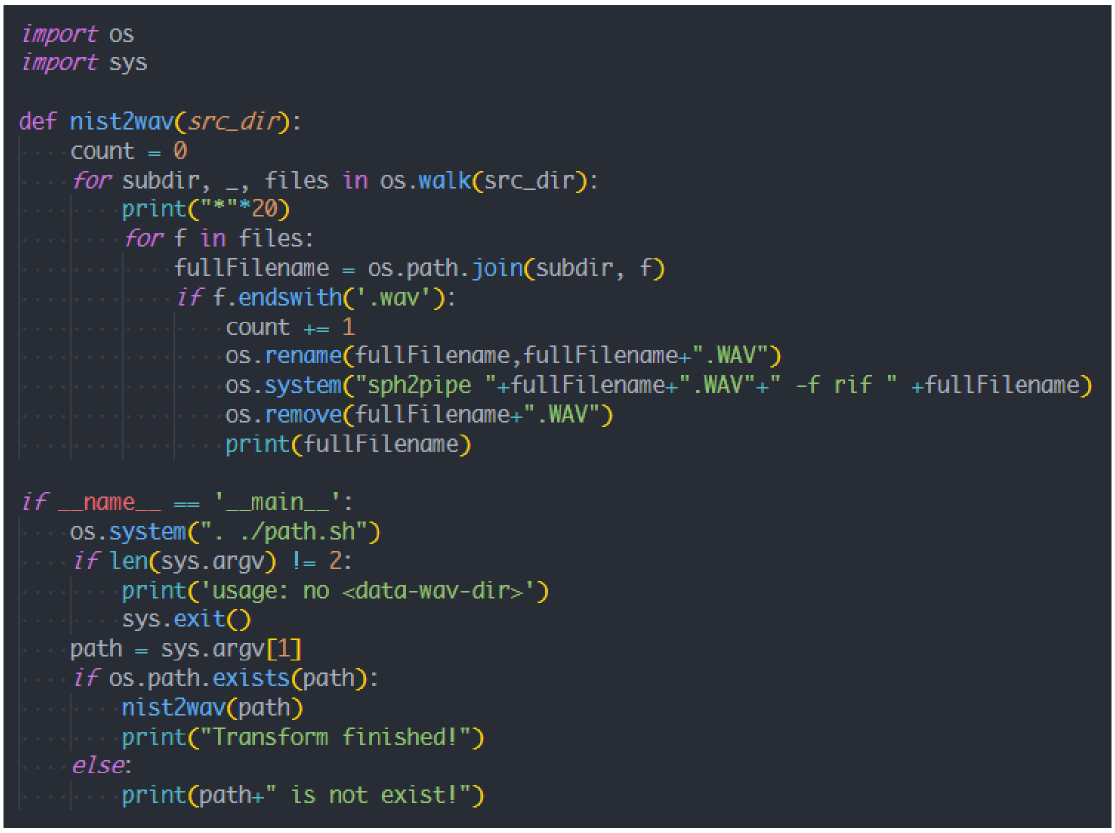
timit/wav/train/SP00{1-462}/{SP00W0{0-9}.wav}

/test/SPK00{463-630}/{SP00W0{0-9}.wav}
这个就是我们需要的数据文件分布

- 格式转换（这步很关键）
  因为 timit 中的 wav 不是 RIFF 格式，在提取 MFCC 时会报错，所以要用 kaldi 中自带的转换功能（sph2pipe）把 wav 都转一下

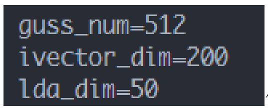


```
~/kaldi/egs/timit_sre/v1# python ./local/nist_to_wav.py ~/data/timit_sre/wav/
```



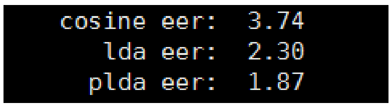
```python
import os
import sys

def nist2wav(src_dir):
    count = 0
    for subdir, _, files in os.walk(src_dir):
        print("*"*20)
        for f in files:
            fullFilename = os.path.join(subdir, f)
            if f.endswith('.wav'):
                count += 1
                os.rename(fullFilename,fullFilename+".WAV")
                os.system("sph2pipe "+fullFilename+".WAV"+" -f rif " +fullFilename)
                os.remove(fullFilename+".WAV")
                print(fullFilename)

if __name__ == '__main__':
    os.system(". ./path.sh")
    if len(sys.argv) != 2:
        print('usage: no <data-wav-dir>')
        sys.exit()
    path = sys.argv[1]
    if os.path.exists(path):
        nist2wav(path)
        print("Transform finished!")
    else:
        print(path+" is not exist!")
```

此刻，准备工作都做完了，接下来就可以开始训练模型了！
如果你比较了解 kaldi 训练和测试的流程，可以直接执行./run.sh（如果机器配置比较低，防止内存溢出，你还需要修改一些参数，比如 njob、thread、process 等）。因为 timit 数据量还算很少的，所以整体时间也很快。最终，可以看到基于 ivector 的余弦、lda 和 PLDA 三种评估结果，结果也同样放在 scores 文件下。
本人当时跑的结果：



```
guss_num=512
ivector_dim=200
lda_dim=50
```

使用了 512 个高斯分量，ivector 维度为 200，lda 测试时降维到 50（这些参数都可以在 run.sh 里直接设置，其他中间过程参数一般在 local、sid、utils 和 steps 目录下的文件中设置）
结果如下（EER 分数，百分比）：

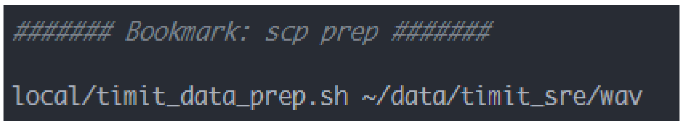


```
  cosine eer:   3.74
     lda eer:   2.30
    plda eer:   1.87
```

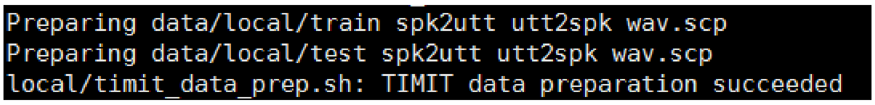
测试集中，3 个语音进行注册（enroll），7 个语音进行验证（eval），168 人

# 数据准备

参考了 aishell.v1 的方式



```
####### Bookmark: scp prep #######

local/timit_data_prep.sh ~/data/timit_sre/wav
```



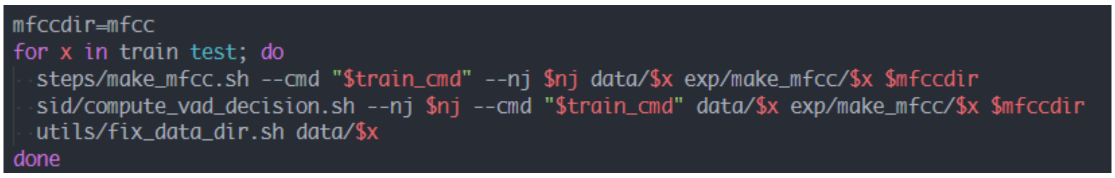
```
Preparing data/local/train spk2utt utt2spk wav.scp
Preparing data/local/test spk2utt utt2spk wav.scp
local/timit_data_prep.sh: TIMIT data preparation succeeded
```

声纹识别只需要 spk2utt、utt2spk 和 wav.scp 三种文件。这些文件表示了“说话人-语音文件集”、“语音文件-说话人”、“语音文件-文件路径”的映射。

# 模型训练

参考了 aishell.v1 的方式

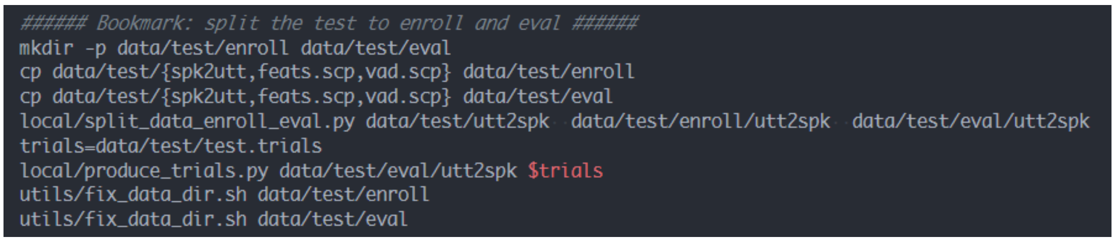
## 特征提取



```
mfccdir=mfcc
for x in train test; do
  steps/make_mfcc.sh --cmd "$train_cmd" --nj $nj data/$x exp/make_mfcc/$x $mfccdir
  sid/compute_vad_decision.sh --nj $nj --cmd "$train_cmd" data/$x exp/make_mfcc/$x $mfccdir
  utils/fix_data_dir.sh data/$x
done
```

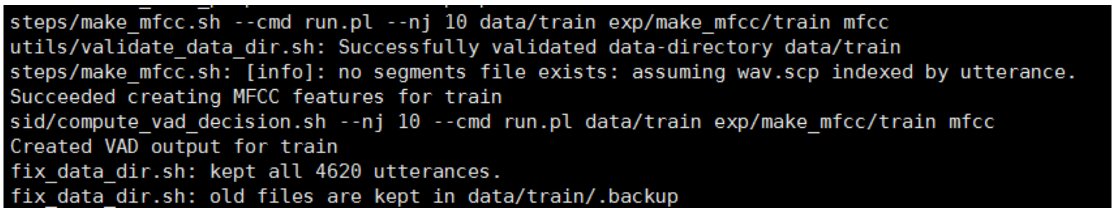
Mfcc 和 vad 参数都可以在 conf 文件中修改。

接下来把测试集中的特征数据，拆分成“注册集”和“验证集”：

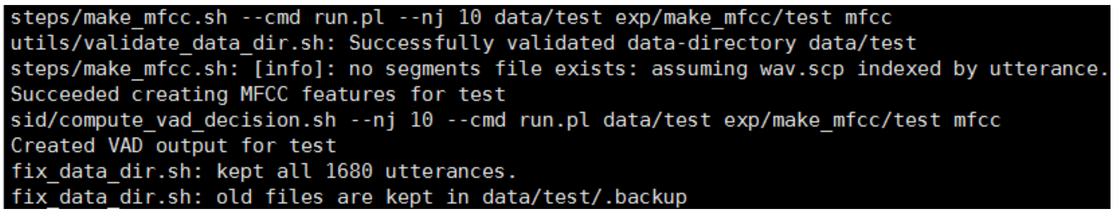


```
###### Bookmark: split the test to enroll and eval ######
mkdir -p data/test/enroll data/test/eval
cp data/test/{spk2utt,feats.scp,vad.scp} data/test/enroll
cp data/test/{spk2utt,feats.scp,vad.scp} data/test/eval
local/split_data_enroll_eval.py data/test/utt2spk  data/test/enroll/utt2spk  data/test/eval/utt2spk
trials=data/test/test.trials
local/produce_trials.py data/test/eval/utt2spk $trials
utils/fix_data_dir.sh data/test/enroll
utils/fix_data_dir.sh data/test/eval
```

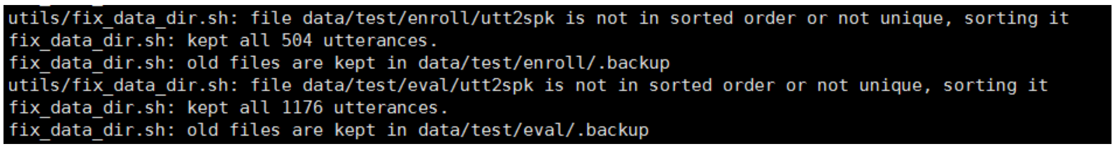
其中$trials 文件比较重要，它是关于“说话人-语音文件-是否匹配”的映射。如果其中一行的语音文件来自这个说话人，则是否匹配为“target”；如果语音文件不来自这个说话人，则为“nontarget”。



```
steps/make_mfcc.sh --cmd run.pl --nj 10 data/train exp/make_mfcc/train mfcc
utils/validate_data_dir.sh: Successfully validated data-directory data/train
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed by utterance.
Succeeded creating MFCC features for train
sid/compute_vad_decision.sh --nj 10 --cmd run.pl data/train exp/make_mfcc/train mfcc
Created VAD output for train
fix_data_dir.sh: kept all 4620 utterances.
fix_data_dir.sh: old files are kept in data/train/.backup
```

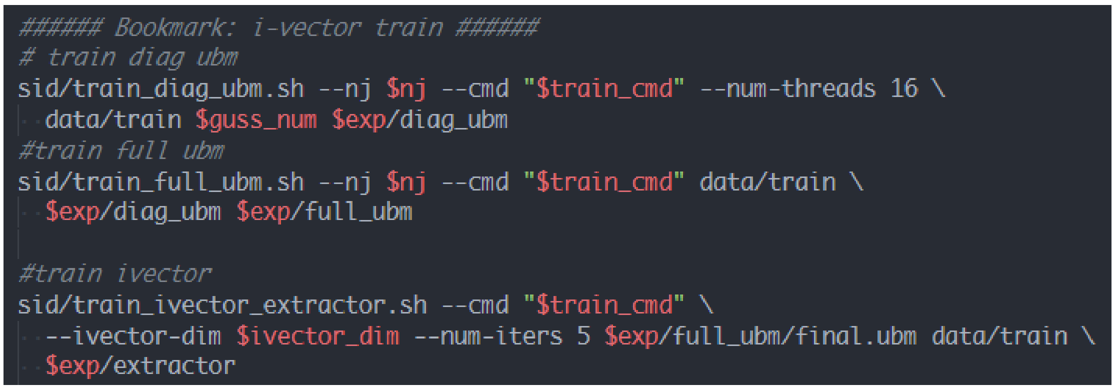


```
steps/make_mfcc.sh --cmd run.pl --nj 10 data/test exp/make_mfcc/test mfcc
utils/validate_data_dir.sh: Successfully validated data-directory data/test
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed by utterance.
Succeeded creating MFCC features for test
sid/compute_vad_decision.sh --nj 10 --cmd run.pl data/test exp/make_mfcc/test mfcc
Created VAD output for test
fix_data_dir.sh: kept all 1680 utterances.
fix_data_dir.sh: old files are kept in data/test/.backup
```



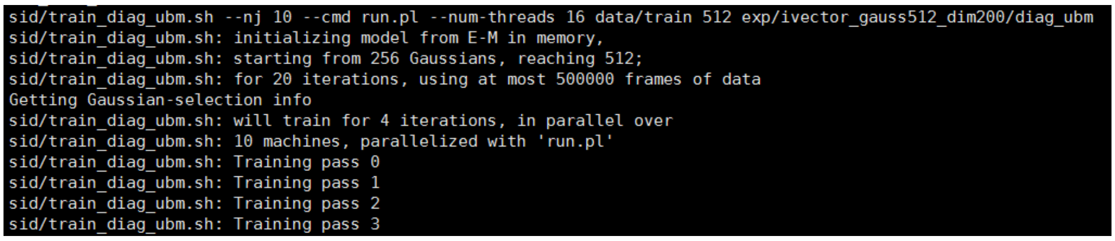
```
utils/fix_data_dir.sh: file data/test/enroll/utt2spk is not in sorted order or not unique, sorting it
fix_data_dir.sh: kept all 504 utterances.
fix_data_dir.sh: old files are kept in data/test/enroll/.backup
utils/fix_data_dir.sh: file data/test/eval/utt2spk is not in sorted order or not unique, sorting it
fix_data_dir.sh: kept all 1176 utterances.
fix_data_dir.sh: old files are kept in data/test/eval/.backup
```

## I-vector 训练

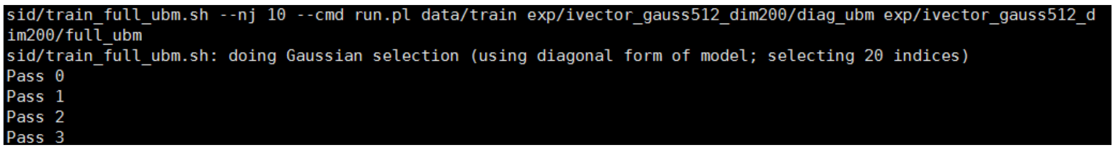


```
###### Bookmark: i-vector train ######
# train diag ubm
sid/train_diag_ubm.sh --nj $nj --cmd "$train_cmd" --num-threads 16 \
  data/train $guss_num $exp/diag_ubm
#train full ubm
sid/train_full_ubm.sh --nj $nj --cmd "$train_cmd" data/train \
  $exp/diag_ubm $exp/full_ubm

#train ivector
sid/train_ivector_extractor.sh --cmd "$train_cmd" \
  --ivector-dim $ivector_dim --num-iters 5 $exp/full_ubm/final.ubm data/train \
  $exp/extractor
```

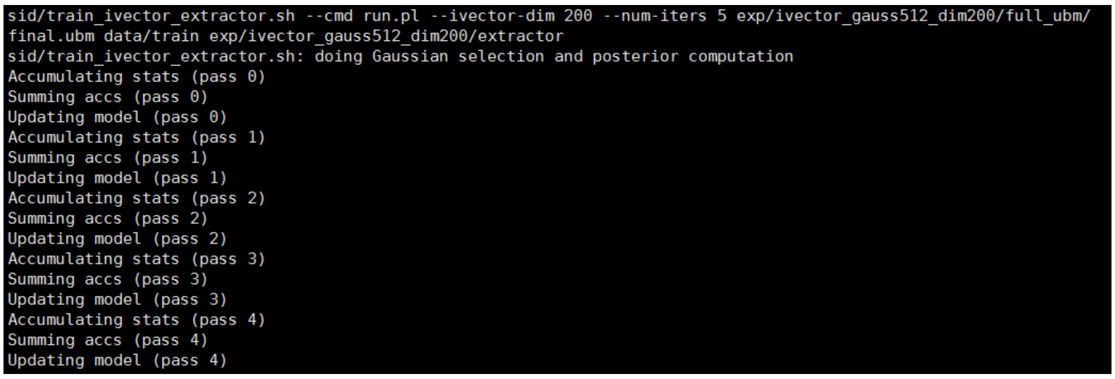
先训练一个对角协方差混合高斯，基于此，在训练一个全协方差混合高斯，它就是 UBM 模型；然后再训练一个 I-vector 模型。



```
sid/train_diag_ubm.sh --nj 10 --cmd run.pl --num-threads 16 data/train 512 exp/ivector_gauss512_dim200/diag_ubm
sid/train_diag_ubm.sh: initializing model from E-M in memory,
sid/train_diag_ubm.sh: starting from 256 Gaussians, reaching 512;
sid/train_diag_ubm.sh: for 20 iterations, using at most 500000 frames of data
Getting Gaussian-selection info
sid/train_diag_ubm.sh: will train for 4 iterations, in parallel over
sid/train_diag_ubm.sh: 10 machines, parallelized with 'run.pl'
sid/train_diag_ubm.sh: Training pass 0
sid/train_diag_ubm.sh: Training pass 1
sid/train_diag_ubm.sh: Training pass 2
sid/train_diag_ubm.sh: Training pass 3
```



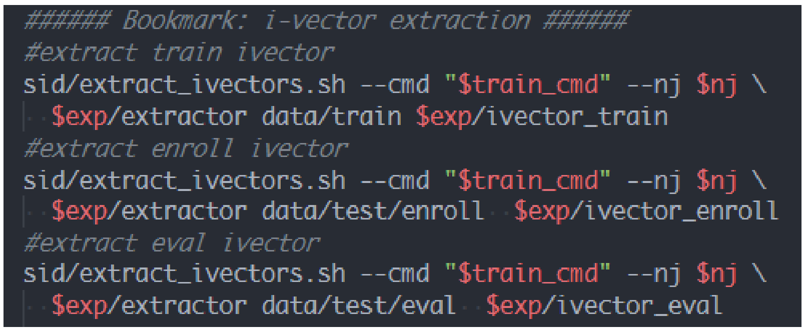
```
sid/train_full_ubm.sh --nj 10 --cmd run.pl data/train exp/ivector_gauss512_dim200/diag_ubm exp/ivector_gauss512_d
im200/full_ubm
sid/train_full_ubm.sh: doing Gaussian selection (using diagonal form of model; selecting 20 indices)
Pass 0
Pass 1
Pass 2
Pass 3
```



```
sid/train_ivector_extractor.sh --cmd run.pl --ivector-dim 200 --num-iters 5 exp/ivector_gauss512_dim200/full_ubm/
final.ubm data/train exp/ivector_gauss512_dim200/extractor
sid/train_ivector_extractor.sh: doing Gaussian selection and posterior computation
Accumulating stats (pass 0)
Summing accs (pass 0)
Updating model (pass 0)
Accumulating stats (pass 1)
Summing accs (pass 1)
Updating model (pass 1)
Accumulating stats (pass 2)
Summing accs (pass 2)
Updating model (pass 2)
Accumulating stats (pass 3)
Summing accs (pass 3)
Updating model (pass 3)
Accumulating stats (pass 4)
Summing accs (pass 4)
Updating model (pass 4)
```

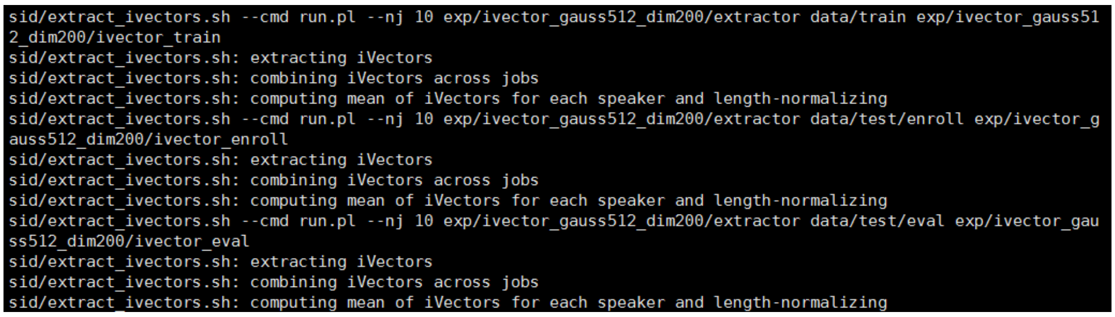
# 测试

## I-vector 特征提取



```
###### Bookmark: i-vector extraction ######
#extract train ivector
sid/extract_ivectors.sh --cmd "$train_cmd" --nj $nj \
    $exp/extractor data/train $exp/ivector_train
#extract enroll ivector
sid/extract_ivectors.sh --cmd "$train_cmd" --nj $nj \
    $exp/extractor data/test/enroll  $exp/ivector_enroll
#extract eval ivector
sid/extract_ivectors.sh --cmd "$train_cmd" --nj $nj \
    $exp/extractor data/test/eval  $exp/ivector_eval
```

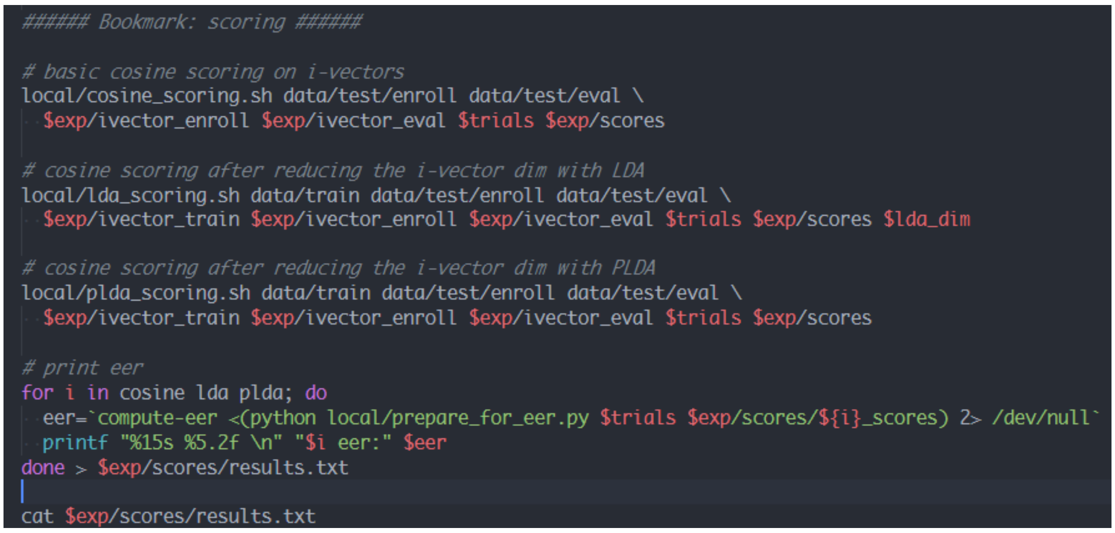
分别对训练集、注册集和验证集提取各自的 I-vector。



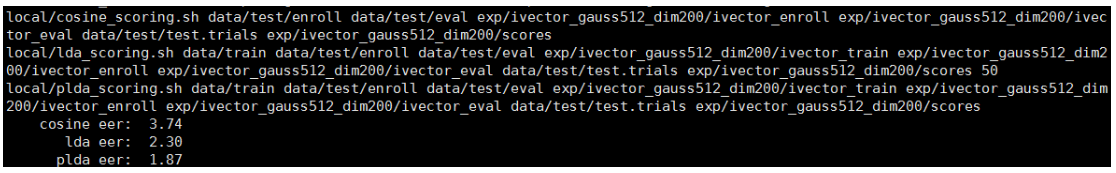
```
sid/extract_ivectors.sh --cmd run.pl --nj 10 exp/ivector_gauss512_dim200/extractor data/train exp/ivector_gauss51
2_dim200/ivector_train
sid/extract_ivectors.sh: extracting iVectors
sid/extract_ivectors.sh: combining iVectors across jobs
sid/extract_ivectors.sh: computing mean of iVectors for each speaker and length-normalizing
sid/extract_ivectors.sh --cmd run.pl --nj 10 exp/ivector_gauss512_dim200/extractor data/test/enroll exp/ivector_g
auss512_dim200/ivector_enroll
sid/extract_ivectors.sh: extracting iVectors
sid/extract_ivectors.sh: combining iVectors across jobs
sid/extract_ivectors.sh: computing mean of iVectors for each speaker and length-normalizing
sid/extract_ivectors.sh --cmd run.pl --nj 10 exp/ivector_gauss512_dim200/extractor data/test/eval exp/ivector_gau
ss512_dim200/ivector_eval
sid/extract_ivectors.sh: extracting iVectors
sid/extract_ivectors.sh: combining iVectors across jobs
sid/extract_ivectors.sh: computing mean of iVectors for each speaker and length-normalizing
```

# 评估

使用了清华大学的方式



```
###### Bookmark: scoring ######

# basic cosine scoring on i-vectors
local/cosine_scoring.sh data/test/enroll data/test/eval \
  $exp/ivector_enroll $exp/ivector_eval $trials $exp/scores

# cosine scoring after reducing the i-vector dim with LDA
local/lda_scoring.sh data/train data/test/enroll data/test/eval \
  $exp/ivector_train $exp/ivector_enroll $exp/ivector_eval $trials $exp/scores $lda_dim

# cosine scoring after reducing the i-vector dim with PLDA
local/plda_scoring.sh data/train data/test/enroll data/test/eval \
  $exp/ivector_train $exp/ivector_enroll $exp/ivector_eval $trials $exp/scores

# print eer
for i in cosine lda plda; do
  eer=`compute-eer <(python local/prepare_for_eer.py $trials $exp/scores/${i}_scores) 2> /dev/null`
  printf "%15s %5.2f \n" "$i eer:" $eer
done > $exp/scores/results.txt

cat $exp/scores/results.txt
```



```
local/cosine_scoring.sh data/test/enroll data/test/eval exp/ivector_gauss512_dim200/ivector_enroll exp/ivector_gauss512_dim200/ivec
tor_eval data/test/test.trials exp/ivector_gauss512_dim200/scores
local/lda_scoring.sh data/train data/test/enroll data/test/eval exp/ivector_gauss512_dim200/ivector_train exp/ivector_gauss512_dim2
00/ivector_enroll exp/ivector_gauss512_dim200/ivector_eval data/test/test.trials exp/ivector_gauss512_dim200/scores 50
local/plda_scoring.sh data/train data/test/enroll data/test/eval exp/ivector_gauss512_dim200/ivector_train exp/ivector_gauss512_dim
200/ivector_enroll exp/ivector_gauss512_dim200/ivector_eval data/test/test.trials exp/ivector_gauss512_dim200/scores
    cosine eer:  3.74
       lda eer:  2.30
      plda eer:  1.87
```

## 可视化

最后使用 matlab 工具包{DETware_v2.1}进行可视化

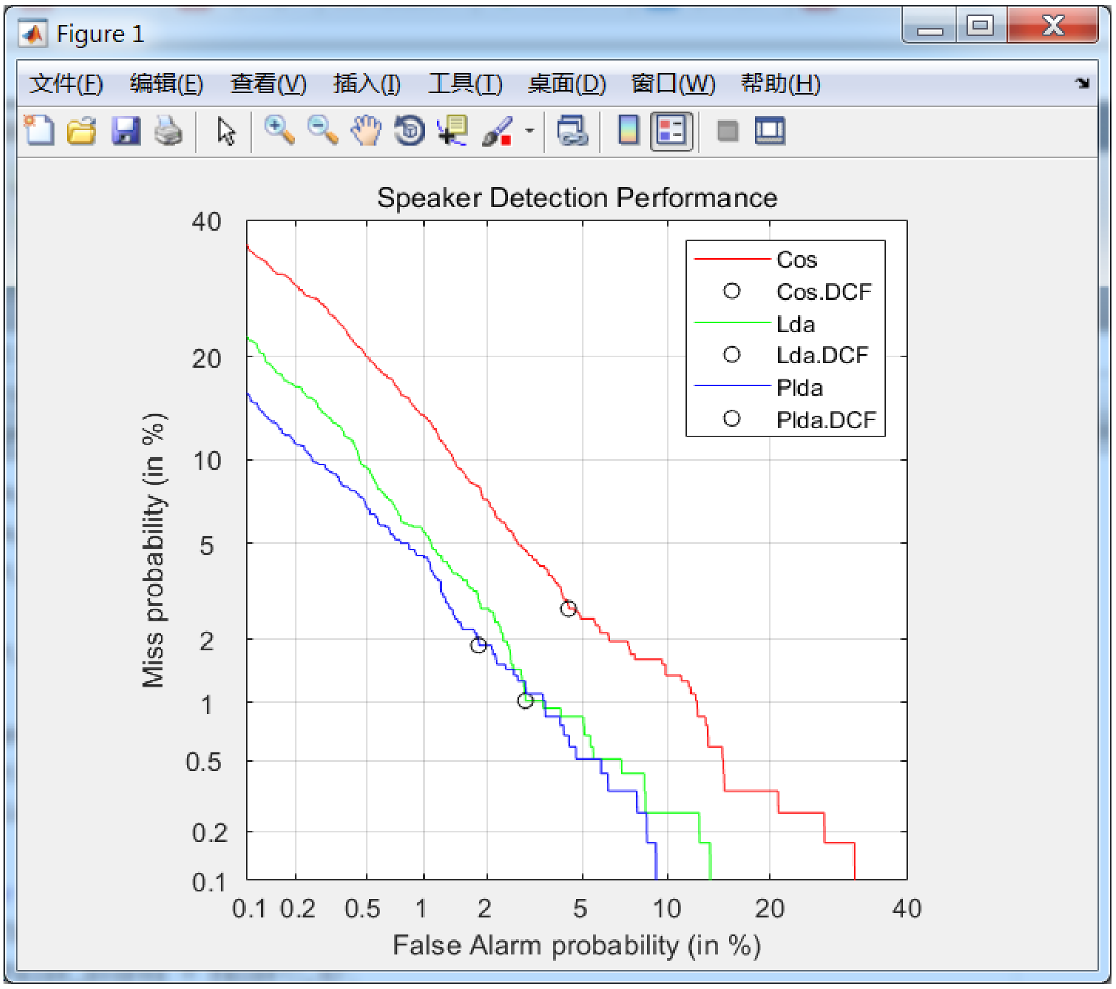


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Speaker Detection Performance
Cos
Cos.DCF
Lda
Lda.DCF
Plda
Plda.DCF
Miss probability (in %)
False Alarm probability (in %)