Appendix

The following chapters include central scripts in executing the experiments presented in Section 4. The scripts assume a certain folder structure in order to make them run, but this can be easily edited in the top of each script. Lists of files are easily obtained, thus their generation are omitted.

Data organization

The following scripts provide a setup of the data, generation of IdMap, Key and Ndx.

task/generate_enroll.py

```
# This script generates the IdMap file that defines the enrollment set in a Sidekit
# speaker identification experiment on the RSR2015 database.
# Speaker models for t-normalization are also trained the same way as ordinary
# enrollment speaker models.
# In addition to the speaker models, a 'complementary' model w is trained on a larger
# amount of data to model all speakers outside the enrollment and t-norm set.
#
# The script is based on the script 'rsr2015_init.py'.
# The set is customly defined to fit to the context of the specific task, and uses
# data from session set A (sessions {1,4,7}). Sentences {001,002,003} from session 1
# plus command 032 from sessions {1,4,7} is used, in total 6 adaptation utterances
# for each target speaker model.
#
# Speakers with id m/f 001-050 is reserved for the enrollment/test speakers
# Speakers with id m/f 051-100 is reserved for z-norm and t-norm
# Speakers with id m/f 101-157/143 is reserved for complementary model adaptation
#
# Written by Joergen Antonsen, April 23 2017


import sidekit
import h5py
import numpy as np
import random
```

```
import os
random.seed(42) # to make the selected data predictable

print('- IdMap file generation -')

# ----------------Set parameters------------------
feat_path = '/home/studenter/jorgeja/Projects/master/data/feats/RSR2015/'

test_set_size = 10 # number of speakers in test set
num_unknown = 2 # number of unknown speakers in test set
enroll_set_size = test_set_size-num_unknown # number of known speakers in
    test set
num_sets = 100 # number of unique test sets

enroll_range = 50 # max index of speakers in enrollment set
norm_range = 100 # max index of speakers in normalization set
separate_gender = False # if true, gender separate IdMap files are
    generated

ww_adapt_num = 3 # number of wake words for model enrollment (IF > 3, YOU
    MUST ALSO EDIT generate_test.py)
stc_adapt_num = 3 # number of sentences for model enrollment
spkr_adapt_num = ww_adapt_num+stc_adapt_num # total number of enrollment
    utterances per speaker
world_adapt_num = 1 # total number of complementary model adaptation
    utterances (per speaker)

gender_name = ['male','female']
if ww_adapt_num <= 3:
    enroll_session_list = ['01','04','07']
else:
    enroll_session_list = ['01','04','07','02','05','08']
full_session_list = ['0'+str(i+1) for i in range(9)]
session_groups = [['01','04','07'],['02','05','08'],['03','06','09']]
sentence_id = ['001','002','003','004','005','006','007','008','009']
full_sentence_list = ['00'+str(i+1) for i in range(9)]+['0'+str(i+1) for i
     in range(9,73)]

ww_id = '032' # wake word id

assert norm_range >= 100, 'Script must be edited if norm_range is less
    than 100'
assert world_adapt_num <= 9, 'world_adapt_num is too big. Value or entire
    script must be edited.'
assert ww_adapt_num <= 6, 'ww_adapt_num is too big. Max size allowed is 6.
    '

# ----------Generate model and segment lists-------
# First we make a list of the speakers in the enrollment set
num_list = [str(i+1) for i in range(enroll_range)]
num_list[0:9] = ['00'+num for num in num_list[0:9]]
num_list[9:enroll_range] = ['0'+num for num in num_list[9:enroll_range]]
spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]

# Then we make a speaker list for the models for t-normalization
num_list = [str(i+1) for i in range(enroll_range,norm_range)]
for i in range(len(num_list)):
```

```
    if len(num_list[i]) == 2:
        num_list[i] = '0'+num_list[i]
if not separate_gender:
    spkr_norm_list = ['m'+num for num in num_list]+['f'+num for num in
    num_list]
else:
    spkr_norm_list = [['m'+num for num in num_list],['f'+num for num in
    num_list]]

# Finally, we make a speaker list for the complementary model training set
num_list_m = [str(i+1) for i in range(norm_range,157)]
num_list_f = [str(i+1) for i in range(norm_range,143)]
if not separate_gender:
    spkr_comp_list = ['m'+num for num in num_list_m]+['f'+num for num in
    num_list_f]
else:
    spkr_comp_list = [['m'+num for num in num_list_m],['f'+num for num in
    num_list_f]]


# ------------Generate IdMap file contents------------
if not separate_gender:
    # MODEL LIST
    models = []
    # Add enrollment speaker model ids to model list
    for spkr in spkr_list:
        models.extend([spkr for i in range(spkr_adapt_num)])
    # Add speaker models for t-normalization
    for spkr in spkr_norm_list:
        models.extend([spkr for i in range(spkr_adapt_num)])
    # Add complementary model id ('w')
    models.extend(['w' for i in range((300-norm_range*2)*world_adapt_num)
    ])


    # SEGMENT LIST
    segments = []
    # Add enrollment and t-norm speaker utterances to segment list
    for spkr in spkr_list+spkr_norm_list:
        # 1: Add wake word from a number (ww_adapt_num) of sessions
        for sess in enroll_session_list:
            if spkr[0] == 'm':
                file_path = os.path.join('male',spkr,spkr+'_'+sess+'_'+
    ww_id)
            elif spkr[0] == 'f':
                file_path = os.path.join('female',spkr,spkr+'_'+sess+'_'+
    ww_id)
            if os.path.exists(os.path.join(feat_path,file_path+'.h5')):
                segments.append(file_path)
            else:
                raise Exception(file_path+'.h5 does not exist.')

        # 2: Add a number (stc_adapt_num) of phrases from the sessions in
    enroll_session_list
        # In 'most' cases, these phrases will be chosen from session '01'
        sess_index = 0
        sess_list = []
```

```
        if spkr[0] == 'm':
            gender = 'male'
        else:
            gender = 'female'

        for i in range(stc_adapt_num):
            sess = random.choice(enroll_session_list)
            file_exists = False
            sess_idx = 0 # To be increased if file does not exist
            while not file_exists:
                sess = enroll_session_list[sess_idx]
                seg_id = sentence_id[i]
                file_path = os.path.join(gender,spkr,spkr+'_'+sess+'_'+
    seg_id)
                if os.path.exists(os.path.join(feat_path,file_path+'.h5'))
     and file_path not in segments:
                    segments.append(file_path)
                    file_exists = True
                    sess_idx = 0
                else:
                    sess_idx += 1

    # Add utterances for complementary model adaptation
    # Sessions and utterance id's are randomly selected
    for spkr in spkr_comp_list:
        files_exists = False
        while not files_exists:
            if spkr[0] == 'm':
                gender = 'male'
            else:
                gender = 'female'
            sess = random.sample(full_session_list,world_adapt_num)
            utt = random.sample(full_sentence_list[0:30],world_adapt_num)
    # selecting only phrases, not short commands or digits
            # First check for existence (if not, redo selection)
            for idx in range(world_adapt_num):
                file_path = os.path.join(gender,spkr,spkr+'_'+sess[idx]+'_
    '+utt[idx])
                files_exists = True
                if not os.path.exists(os.path.join(feat_path,file_path+'.
    h5')):
                    files_exists = False
                    break
            # Then add segments to segment list
            if files_exists:
                for idx in range(world_adapt_num):
                    file_path = os.path.join(gender,spkr,spkr+'_'+sess[idx
    ]+'_'+utt[idx])
                    segments.append(file_path)

else:
    # Write gender-dependent code when needed
    pass



# -------------Make and save IdMap-files------------
```

```
if separate_gender:
    # Make two IdMap-files
    for gender in [0,1]:
        enroll_idmap = sidekit.IdMap()
        enroll_idmap.leftids = np.asarray(models[gender])
        enroll_idmap.rightids = np.asarray(segments[gender])
        enroll_idmap.start = np.empty(enroll_idmap.rightids.shape, '|O')
        enroll_idmap.stop = np.empty(enroll_idmap.rightids.shape, '|O')
        valid = enroll_idmap.validate()
        if not valid:
            raise ValueException('Error in generating IdMap file.')
        enroll_idmap.write('enroll_'+gender_name[gender]+'_'+str(
    enroll_size)+'_ww_'+str(ww_adapt_num)+'_'+str(stc_adapt_num)+'.h5')
else:
    # Make one IdMap-file
    enroll_idmap = sidekit.IdMap()
    enroll_idmap.leftids = np.asarray(models)
    enroll_idmap.rightids = np.asarray(segments)
    enroll_idmap.start = np.empty(enroll_idmap.rightids.shape, '|O')
    enroll_idmap.stop = np.empty(enroll_idmap.rightids.shape, '|O')
    valid = enroll_idmap.validate()
    if not valid:
        raise ValueException('Error in generating IdMap file.')
    enroll_idmap.write('enroll_gidp_{}_t_norm_{}_ww_{}_stc_{}.h5'\
                           .format(str(enroll_range),str(norm_range-
    enroll_range),\
                            str(ww_adapt_num),str(stc_adapt_num)))

print('Success!')
```

task/generate_test.py

```
# This script generates a random test set for a speaker identification
    experiment in Sidekit.
# Its output consists of two files:
# - 'eval....txt', which can be read directly by sidekit.Key() using Key('
    eval....txt')
# - 'cat_list....sh', which lists files to be concatenated and stored in
    '../data/sph/cat/',
# and requires permission change (chmod +x <filename>) to be executeable.
#
# All test data is from the RSR2015 database session sets B (sessions
    {2,5,8}) and C (sessions {3,6,9})
# For each trial speaker, a concatenated phrase made by segments from the
    same session is generated.
# The phrase consists of command segment 032 (wake word) and one of the
    sentence segments 010-030.
#
# The test sets are open sets with a number of known and unknown speakers.
# The known speakers (enrollment speakers) have MAP-adapted speaker models
     (id 01-50).
# All test speakers are scored against these models and the adapted
    complementary model (trained on id 101-157/143).
# From the current parameters, there are 100 test sets, each consisting of
     10 speakers.
# In addition, all test speakers are scored against a number of models (id
     51-100) for t-normalization,
```

```
# and a number of utterances from the same set of speakers are scored against the enroll-set for z-normalization.
#
# NOTE: Subdirectories under cat_dir: male/ and female/, must exist in advance
#
# Written by Joergen Antonsen, April 24 2017


import os
import copy
import random
random.seed(42) # make random.shuffle() do the same every time this script is executed.

print('- Test set and concatenation script generation -')

# ----------------------Set parameters------------------------
base_dir = '/home/studenter/jorgeja/Projects/master/'
rsr2015path = base_dir+'data/sph/RSR2015/'
cat_dir = base_dir+'data/sph/cat/' # output location of concatenated files
data_path = base_dir+'data/'
feat_path = base_dir+'data/feats/RSR2015'

test_set_size = 10 # number of speakers in the test set
num_unknown = 2 # number of unknown speakers in test set
enroll_set_size = test_set_size-num_unknown # number of known speakers in test set
num_sets = 100 # number of unique test sets

enroll_range = 50 # max index of speakers in enrollment set
norm_range = 100 # max index of speakers in normalization set
separate_gender = False # if true, gender separate Key/Ndx files are generated

spkr_adapt_num = 6 # number of speaker model adaptation utterances (per speaker)
world_adapt_num = 1 # total number of complementary model adaptation utterances (per speaker)

gender_list = ['m','f']
gender_name = ['male/','female/'] # for path names
test_session_list = ['02','03','05','06','08','09']
full_session_list = ['0'+str(i+1) for i in range(9)]
session_groups = [['01','04','07'],['02','05','08'],['03','06','09']]
command_id = ['0'+str(i+1) for i in range(9,30)]
ww_id = '032' # wake word id

assert norm_range >= 100, 'Script must be edited if norm_range is less than 100'
assert world_adapt_num <= 9, 'world_adapt_num is too big. Value or entire script must be edited.'


# ----------Generate model and segment lists-------
# First we make a list of the speakers in the enrollment set
num_list = [str(i+1) for i in range(enroll_range)]
```

```
num_list[0:9] = ['00'+num for num in num_list[0:9]]
num_list[9:enroll_range] = ['0'+num for num in num_list[9:enroll_range]]
spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]

# We then do random selection of num_sets sets of speakers
# These test sets are stored in test_set with the following format for each test:
# [tg, tg, tg, tg, tg, tg, tg, tg, imp, imp], tg = target, imp = 'impostor'
if not separate_gender:
    test_set = []
    spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]
    for sets in range(num_sets):
        test_set.append(random.sample(spkr_list,test_set_size))
    with open('test_set_list_gidp_'+str(test_set_size)+'_spk_'+str(num_sets)+'_sets.txt','w') as fh:
        for sets in test_set:
            for spkr in sets:
                fh.write(spkr+' ')
            fh.write('\n')
else:
    test_set = [[],[]]
    spkr_list = [['m'+num for num in num_list],['f'+num for num in num_list]]
    for gender in [0,1]:
        for sets in range(num_sets):
            test_set[gender].append(random.sample(spkr_list[gender], test_set_size))
        with open('test_set_list_'+gender_name[gender]+'_'+str(test_set_size)+'_spk_'+str(num_sets)+'_sets.txt','w') as fh:
            for sets in test_set[gender]:
                for spkr in sets:
                    fh.write(spkr+' ')
                    fh.write('\n')


# ----------------Define tests and make test lists-----------------
# First, we define some speaker lists:
enroll_spkr_list = spkr_list
num_list = [str(i+1) for i in range(enroll_range,norm_range)]
for i in range(len(num_list)):
    if len(num_list[i]) == 2:
        num_list[i] = '0'+num_list[i]
if not separate_gender:
    norm_spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]
else:
    norm_spkr_list = [['m'+num for num in num_list],['f'+num for num in num_list]]

# Then, lists to hold the models and test segments, and identity labels plus concatenation list
if not separate_gender:
    model_list = []
    test_segment_list = []
    identity_lbl_list = []
```

```
    cat_list = [] # format: [[wake_word],[command],[concatenated segment]]

    # 1: For each test set, score all speakers against enrolled speaker
    models
    for tests in test_set:
        # Score against all speakers for each test
        for j in range(test_set_size): # j denotes both known and unknown
    speakers)
            # Select session and utterances for testing (wake word+command
    )
            test_spkr = tests[j]
            gender = 'male' if test_spkr[0] == 'm' else 'female'
            file_exists = False
            while not file_exists:
                sess = random.choice(test_session_list)
                com_id = random.choice(command_id)
                ww_path = os.path.join(gender,test_spkr,test_spkr+'_'+sess
    +'_'+ww_id)
                com_path = os.path.join(gender,test_spkr,test_spkr+'_'+
    sess+'_'+com_id)
                cat_path = os.path.join('cat',gender,test_spkr+'_'+sess+'_
    '+ww_id+'-'+com_id)
                if os.path.exists(os.path.join(rsr2015path,'sph',ww_path+'
    .sph')) and os.path.exists(os.path.join(rsr2015path,'sph',com_path+'.
    sph')) and cat_path not in test_segment_list:
                    for i in range(enroll_set_size): # i denotes speaker
    models to be tested
                        enroll_spkr = tests[i]
                        model_list.append(enroll_spkr)
                        test_segment_list.append(cat_path)
                        if i == j: # target trial
                            identity_lbl_list.append('target')
                        else: # nontarget trial
                            identity_lbl_list.append('nontarget')
                    cat_list.append([ww_path,com_path,cat_path])
                    file_exists = True


    # 2: t-normalization: Score each test speaker utterance against
    enrolled norm-nodels
    orig_test_segment_list = list(set(test_segment_list))

    for norm_spkr in norm_spkr_list:
        for segment in orig_test_segment_list:
            if norm_spkr[0] == segment.split('/')[2][0]: # NOTE:
    Normalization is gender-dependent
                # NOTE cont.: This can be avoided by removing the above if
    -condition
                # NOTE cont.: and using the commented code below instead
                model_list.append(norm_spkr)
                test_segment_list.append(segment)
                identity_lbl_list.append('nontarget')
        # The following commands can be used in case of gender-independent
         normalization
        # model_list.extend([norm_spkr for i in range(len(
    orig_test_segment_list))])
        # test_segment_list.extend(orig_test_segment_list)
```

```
        # identity_lbl_list.extend('nontarget' for i in range(len(
    orig_test_segment_list)))

    current_list_pos = len(test_segment_list) # current position in model
    and test segment lists


    # 3: z-normalization: Score z-/t-norm-speaker utterances against all
    test speaker models,
    # and the adapted complementary model
    for enroll_spkr in enroll_spkr_list:
        for spkr in norm_spkr_list:
            if enroll_spkr[0] == spkr[0]: # NOTE: Only utterances with the
     same gender as the model is scored
                # NOTE cont.: Simply remove the above if-condition for
    gender-independent normalization
                gender = 'male' if spkr[0] == 'm' else 'female'
                # Select session and utterances (wake word+command)
                file_exists = False
                while not file_exists:
                    sess = random.choice(full_session_list)
                    com_id = random.choice(command_id)
                    ww_path = os.path.join(gender,spkr,spkr+'_'+sess+'_'+
    ww_id)
                    com_path = os.path.join(gender,spkr,spkr+'_'+sess+'_'+
    com_id)
                    cat_path = os.path.join('cat',gender,spkr+'_'+sess+'_'
    +ww_id+'-'+com_id)
                    if os.path.exists(os.path.join(rsr2015path,'sph',
    ww_path+'.sph')) and os.path.exists(os.path.join(rsr2015path,'sph',
    com_path+'.sph')) and cat_path not in test_segment_list[
    current_list_pos:len(test_segment_list)]:
                        # Add scoring against enroll speaker
                        model_list.append(enroll_spkr)
                        test_segment_list.append(cat_path)
                        identity_lbl_list.append('nontarget')
                        # Add scoring against complementary model
                        model_list.append('w')
                        test_segment_list.append(cat_path)
                        identity_lbl_list.append('nontarget')
                        # Append to concatenation list
                        cat_list.append([ww_path,com_path,cat_path])
                        file_exists = True


    # 4: Score each test speaker utterance against adapted complementary
    model
    for segment in orig_test_segment_list:
        test_segment_list.append(segment)
        model_list.append('w')
        identity_lbl_list.append('nontarget')

    print('Total number of tests: '+str(len(model_list)))
    print('Total number of files to concatenate: '+str(len(cat_list)))
    print(' -about '+str(130*len(cat_list)/1000)+' MB space for audio (sph
    ) cat files')
    print(' -about '+str(50*len(cat_list)/1000)+' MB space for feature cat
```

```
     files (.h5) after extraction')

    if len(model_list) != len(test_segment_list) or len(test_segment_list)
     != len(identity_lbl_list):
        raise Exception('Length of model_list, test_segment_list and
    identity_lbl_list is not equal!')

else:
    # Write gender-dependent code when needed
    pass



# --------------Make sh-file to define concatenation--------------
if not separate_gender:
    cat_filename = 'concatenate_gidp_'+str(test_set_size)+'_spk_'+str(
    num_sets)+'_sets_more_data.sh'
else:
    cat_filename = 'concatenate_gdp_'+str(test_set_size)+'_spk_'+str(
    num_sets)+'_sets_more_data.sh'

with open(cat_filename,'w') as cat_file:
    cat_file.write('#!/bin/bash\n')
    # Write all files in cat_list to file
    for segment in cat_list:
        temp_ww_id = os.path.join(rsr2015path,'sph',segment[0]+'.sph')
        temp_utt_id = os.path.join(rsr2015path,'sph',segment[1]+'.sph')
        temp_cat_id = os.path.join(data_path,'sph',segment[2]+'.sph')
        cat_file.write('sox '+temp_ww_id+' '+temp_utt_id+' '+temp_cat_id+'
    \n')


# --------------Make key-txtfile to define trials---------------
if not separate_gender:
    key_filename = 'eval_gidp_'+str(test_set_size)+'_spk_'+str(num_sets)+'
    _sets_more_data.txt'
else:
    key_filename = 'eval_gdp_'+str(test_set_size)+'_spk_'+str(num_sets)+'
    _sets_more_data.txt'

with open(key_filename,'w') as key_file:
    # Add all test definitions
    for i in range(len(model_list)):
        key_file.write(model_list[i]+' '+test_segment_list[i]+' '+
    identity_lbl_list[i]+'\n')



# The following commands will not run on the sirkus8 server (location of
    Sidekit install),
# the reason of this being that 'sidekit/bosaris/key.py' does not support
    loading txt-files.
# 'key.py' can however be easily modified to support this, as Key.read_txt
    () already exists.
# See Key.__init__().


```

```
# ---------------Convert key txt-file to hdf5----------------
"""
key = sidekit.Key(key_name+'.txt')
key.write(key_name+'_key.h5')

# Make ndx-file from key and store in hdf5-format
ndx = key.to_ndx()
ndx.write(key_name+'_ndx.h5')
"""
```

Feature extraction

The following script shows the general feature extraction setup for the concatenated files used in the experiments. The setup is similar to the feature extraction from the other datasets used in this thesis.

task/feat_ext_cat.py

```
# This script extracts MFCC features from the utterances in /sph/cat/{male,female}
# It is based on the svm-gmm tutorial from the SIDEKIT website

import numpy as np
import sidekit
import multiprocessing
import os
import sys
import time


print(time.strftime('%c'))
print('FEATURE EXTRACTION FROM ./sph/cat:')

base_dir = '/home/studenter/jorgeja/Projects/master/'


# Set the number of parallel process to run.
nbThread = max(multiprocessing.cpu_count()-1, 1)

# Make cat-dir file list
file_list = []
for gender_name in ['male/','female/']:
    temp_list = os.listdir(base_dir+'data/sph/cat/'+gender_name)
    temp_list = ['cat/'+gender_name+file.split('.')[0] for file in temp_list]
    file_list.extend(temp_list)


# PROCESS THE AUDIO TO SAVE MFCC ON DISK
print("Initialize FeaturesExtractor")
extractor = sidekit.FeaturesExtractor(audio_filename_structure=base_dir+"data/sph/{}.sph",
                                      feature_filename_structure=base_dir+"data/feats/RSR2015/more_data/{}.h5",
                                      sampling_frequency=16000,
```

```
                                        lower_frequency=300,
                                        higher_frequency=3400,
                                        filter_bank="log",
                                        filter_bank_size=24,
                                        window_size=0.02,
                                        shift=0.01,
                                        ceps_number=19,
                                        vad="snr",
                                        snr=40,
                                        pre_emphasis=0.97,
                                        save_param=["vad", "energy", "cep"],
                                        keep_all_features=False)

# Get the complete list of features to extract
show_list = np.unique(np.hstack([file_list]))
channel_list = np.zeros_like(show_list, dtype = int)


print("Extract features and save to disk")
extractor.save_list(show_list=show_list,
                    channel_list=channel_list,
                    num_thread=nbThread)

print('Success!')
```

UBM training

The following script trains the UBM used in all experiments.

UBM/train_ubm_ldc.py

```
# This script trains a gender-independent ubm and stores it to gmm/
    gender_id_wo_fisher_ubm.h5
# It also trains gender-dependent ubms, stored to  gmm/{male,female}
    _wo_fisher_ubm.h5
# It is based on the example scripts of the Sidekit documentation
# Written by Joergen Antonsen, April 2 2017
# Modified by Joergen Antonsen, April 4 2017

import sidekit
import os
import sys
import multiprocessing
import logging
import numpy as np
import time
import random

logging.basicConfig(filename='log/ldc_ubm.log',level=logging.DEBUG)

print(time.strftime('%c'))
start = time.time()
print('UBM TRAINING')
print('Prepare for training..')

```

```
# Set parameters
train_gender_models = True # if false, train gender-independent model
#gender_to_train = 'male' # 'male' or 'female'
distribNb = 1024  # number og GMM components
feat_dir = "/home/studenter/jorgeja/Projects/master/data/feats/LDC/new/"
selection_size = 1000 # size of ubm lists after data selection
random.seed(1) # to make the selected data predictable

# Automatically set the number of parallel process to run.
nbThread = max(multiprocessing.cpu_count()-1, 1)

"""
# Read ubm-lists
if not train_gender_models:
    with open('task/ubm_list_all.txt') as inputFile: # example line:
    female/kcro
        ubmList = inputFile.read().split('\n')
        while '' in ubmList:
            ubmList.remove('')
else:
    with open('task/ubm_list_m.txt') as inputFile:
        ubmList_m = inputFile.read().split('\n')
    while '' in ubmList_m:
        ubmList_m.remove('')
    with open('task/ubm_list_f.txt') as inputFile:
        ubmList_f = inputFile.read().split('\n')
    while '' in ubmList_f:
        ubmList_f.remove('')
"""

# Make ubm-lists from random selections of features in ../data/feats
ubmList_m = os.listdir(feat_dir+'male')
ubmList_m = ['male/'+files.split('.')[0] for files in ubmList_m]
ubmList_f = os.listdir(feat_dir+'female')
ubmList_f = ['female/'+files.split('.')[0] for files in ubmList_f]

#ubmList = ubmList_m + ubmList_f

ubmList_m = random.sample(ubmList_m, selection_size)
ubmList_f = random.sample(ubmList_f, selection_size)
ubmList = ubmList_m[0:selection_size/2] + ubmList_f[0:selection_size/2]
"""
with open('/home/studenter/jorgeja/Projects/master/ubm/lists/ubm_list_m_'+
    str(selection_size)+'.txt','w') as fh:
    for files in ubmList_m:
        fh.write(files+'\n')
with open('/home/studenter/jorgeja/Projects/master/ubm/lists/ubm_list_f_'+
    str(selection_size)+'.txt','w') as fh:
    for files in ubmList_f:
        fh.write(files+'\n')
with open('/home/studenter/jorgeja/Projects/master/ubm/lists/ubm_list_all_
    '+str(selection_size)+'.txt','w') as fh:
    for files in ubmList:
        fh.write(files+'\n')
"""

# Check size of file lists
```

```
temp_size_m = 0
for files in ubmList_m:
    temp_size_m += os.stat(feat_dir+files+'.h5').st_size
temp_size_m /= 1000000
print('The size of the male set is '+str(temp_size_m)+'MB')

temp_size_f = 0
for files in ubmList_f:
    temp_size_f += os.stat(feat_dir+files+'.h5').st_size
temp_size_f /= 1000000
print('The size of the female set is '+str(temp_size_f)+'MB')

temp_size = 0
for files in ubmList:
    temp_size += os.stat(feat_dir+files+'.h5').st_size
temp_size /= 1000000
print('The size of the mixed set is '+str(temp_size)+'MB')

#if temp_size_m > 7000 or temp_size_f > 7000 or temp_size > 7000:
#    raise ValueError('Training set is too large.')


# Create a FeaturesServer to load features and feed the other methods
features_server = sidekit.FeaturesServer(features_extractor=None,
                                         feature_filename_structure=
    feat_dir+"{}.h5",
                                         sources=None,
                                         dataset_list=["vad","cep"], #
    NOTE: vad not in feature vector, only for frame selection
                                         mask=None,
                                         feat_norm="cmvn",
                                         global_cmvn=None,
                                         dct_pca=False,
                                         dct_pca_config=None,
                                         sdc=False,
                                         sdc_config=None,
                                         delta=True,
                                         double_delta=True,
                                         delta_filter=None,
                                         context=None,
                                         traps_dct_nb=None,
                                         rasta=True,
                                         keep_all_features=False)



# Load and preprocess features, and train ubm

# ----------- GENDER INDEPENDENT -----------
if not train_gender_models:
    print('Train gender-independent UBM')
    ubm = sidekit.Mixture()
    llk = ubm.EM_split(features_server, ubmList, distribNb, num_thread=
    nbThread, save_partial=True)
    ubm.write('gmm/gender_id_diag_ubm_'+str(distribNb)+'.h5')

    print('Training done!')
```

```
    print(time.strftime('%c'))
    end = time.time()
    m, s = divmod(end-start, 60)
    h, m = divmod(m, 60)
    print("Time elapsed: "+str(round(h))+'H '+str(round(m))+'M '+str(round
    (s))+'S')

    """
    print('Train full covariance UBM')
    ubm_full = sidekit.Mixture()
    ubm_full.EM_convert_full(diagonal_mixture=ubm,
                             features_server=features_server,
                             featureList=ubmList,
                             distrib_nb=distribNb,
                             iterations=2,
                             num_thread=nbThread
                             )
    """

else: # gender_to_train == 'male':
    # -------------- MALE --------------
    print('Train male UBM')
    start = time.time()
    ubm_m = sidekit.Mixture()
    llk_m = ubm_m.EM_split(features_server, ubmList_m, distribNb,
    num_thread=nbThread, save_partial=False)
    ubm_m.write('gmm/male_diag_ubm_'+str(distribNb)+'.h5')
    print('Training done!')
    print(time.strftime('%c'))
    end = time.time()
    m, s = divmod(end-start, 60)
    h, m = divmod(m, 60)
    print("Time elapsed: "+str(round(h))+'H '+str(round(m))+'M '+str(round
    (s))+'S')

    # ------------- FEMALE -------------
    print('Train female UBM')
    start = time.time()
    ubm_f = sidekit.Mixture()
    llk_f = ubm_f.EM_split(features_server, ubmList_f, distribNb,
    num_thread=nbThread, save_partial=False)
    ubm_f.write('gmm/female_diag_ubm_'+str(distribNb)+'.h5')
    print('Training done!')
    print(time.strftime('%c'))
    end = time.time()
    m, s = divmod(end-start, 60)
    h, m = divmod(m, 60)
    print("Time elapsed: "+str(round(h))+'H '+str(round(m))+'M '+str(round
    (s))+'S')

#else:
#    raise ValueError("Error in settings. Please change settings by
    editing this script.")
```

GMM-UBM system

The following scripts define and run the GMM-UBM systems.

Running and writing scores: GMM-UBM/run_gmm_ubm.sh

```
# Runs a gmm-ubm system
# This system serves as a baseline together with the i-vector/PLDA script
# Differing from run_ubm_gmm_sid.py, this version of the file takes
    arguments

import sidekit
import os
import sys
import multiprocessing
import matplotlib.pyplot as mpl
import logging
import numpy as np
import time

logging.basicConfig(filename='log/ubm_gmm.log',level=logging.DEBUG)

if len(sys.argv) != 4:
    print('Usage: '+sys.argv[0]+' <use more test data {0,1}> <number of
    enrollment wake words>'\
              +'<number of enrollment sentences>')
    exit()

print('RUN GMM-UBM SPEAKER IDENTIFICATION TESTS')
print(time.strftime('%c'))

# Set parameters
gender = 'gidp' # 'gdp' 'gidp'
distribNb = 1024  # number of Gaussian distributions for each GMM
base_dir = '/home/studenter/jorgeja/Projects/master/'
feat_dir = base_dir+'data/feats/RSR2015/'
ubm_dir = base_dir+'ubm/gmm/gender_id_diag_ubm.h5'
selection_size = 1000 # number of utterances in ubm training set

ww_adapt_num = int(sys.argv[2]) # number of wakewords per speaker for
    enrollment data
stc_adapt_num = int(sys.argv[3]) # number of sentences per speaker for
    enrollment data
more_test_data = sys.argv[1] # decides length of test data (0: wakeword+
    command, 1: wakeword+sentence)

save_stats = False # if True, save sufficient statistics

print('- ww_adapt_num = '+str(ww_adapt_num))
print('- stc_adapt_num = '+str(stc_adapt_num))
print('- more_test_data = '+more_test_data)

# Automatically set the number of parallel process to run.
nbThread = max(multiprocessing.cpu_count()-1, 1)


# Load task definition
if stc_adapt_num == 3:
    # Adaptation data: 3 wake-words and 3 sentences
    enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/enroll_'+gender+'
    _50_t_norm_50.h5')
```

```
elif stc_adapt_num == 6:
    # Adaptation data: 3 wake-words and 6 sentences
    enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/more_data/enroll_'
    +gender+'_50_t_norm_50_ww_3_stc_6.h5')
elif stc_adapt_num == 9:
    # Adaptation data: 3 wake-words and 9 sentences
    enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/more_data/enroll_'
    +gender+'_50_t_norm_50_ww_3_stc_9.h5')
else:
    raise ValueError('Enrollment IdMap file not found with stc_adapt_num =
     '+str(stc_adapt_num))

if more_test_data == '0':
    test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/eval_'+gender+'
    _10_spk_100_sets_ndx.h5')
    key = sidekit.Key(base_dir+'task/gmm_ubm/eval_'+gender+'
    _10_spk_100_sets_key.h5')
elif more_test_data == '1':
    test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/more_data/eval_'+gender+
    '_10_spk_100_sets_more_data_ndx.h5')
    key = sidekit.Key(base_dir+'task/gmm_ubm/more_data/eval_'+gender+'
    _10_spk_100_sets_more_data_key.h5')
else:
    raise ValueError('Invalid value of parameter <use more test data
    {0,1}>: '+more_test_data)


# Create a FeaturesServer to load features and feed the other methods
features_server = sidekit.FeaturesServer(features_extractor=None,
                                         feature_filename_structure=
    feat_dir+"{}.h5",
                                         sources=None,
                                         dataset_list=["cep", "vad"],
                                         mask=None,
                                         feat_norm="cmvn",
                                         global_cmvn=None,
                                         dct_pca=False,
                                         dct_pca_config=None,
                                         sdc=False,
                                         sdc_config=None,
                                         delta=True,
                                         double_delta=True,
                                         delta_filter=None,
                                         context=None,
                                         traps_dct_nb=None,
                                         rasta=True,
                                         keep_all_features=False)

print('Load the UBM')
# Load ubm
ubm = sidekit.Mixture()
ubm.read(ubm_dir)


print('Compute the sufficient statistics')
# Create a StatServer for the enrollment data and compute the statistics
enroll_stat = sidekit.StatServer(enroll_idmap,
```

```
                                    distrib_nb=distribNb,
                                    feature_size=57)
enroll_stat.accumulate_stat(ubm=ubm,
                            feature_server=features_server,
                            seg_indices=range(enroll_stat.segset.shape[0])
    ,
                            num_thread=nbThread)
if not os.path.exists('data/stat_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5'):
    enroll_stat.write('data/stat_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')

print('MAP adaptation of the speaker models')
regulation_factor = 3  # MAP regulation factor
enroll_sv = enroll_stat.adapt_mean_map_multisession(ubm, regulation_factor
    )
if not os.path.exists('data/sv_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5'):
    enroll_sv.write('data/sv_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')

print('Compute trial scores')
scores_gmm_ubm = sidekit.gmm_scoring(ubm,
                                     enroll_sv,
                                     test_ndx,
                                     features_server,
                                     num_thread=nbThread)
if more_test_data == '0':
    scores_gmm_ubm.write('scores/scores_gmm_ubm_enroll_'+gender+'_100_ww_'
    +str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
else:
    scores_gmm_ubm.write('scores/more_test_data/scores_gmm_ubm_enroll_'+
    gender+'_100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')

print('Completed scoring')
print('')

# Analyzation of scores, including z-/t-norm, will be done in a separate
    script
```

Library for score evaluation: GMM-UBM/normalize.py

```
# This script contains definitions of scoring and normalization
# functions to be used in GMM-UBM system evaluations

import sidekit
import numpy as np
import copy


def get_second_largest(mylist):
    # Returns the second largest element in a numpy.ndarray
    newlist = np.delete(mylist, mylist.argmax())
    return newlist.max()

def get_global_minmax(x_val, y_val, max_or_min='', log_y=False):
    # Returns global max or global min of a numpy.ndarray
```

```
    # x_val :       numpy.ndarray. The x-axis
    # y_val :       numpy.ndarray. The y-axis
    # max_or_min : string. Defines if global min or max should be returned
    # log_y :       bool. If true, return logarithmic y-value
    assert max_or_min == 'max' or max_or_min == 'min', 'max_or_min must be
     either \'max\' or \'min\''

    if max_or_min == 'max':
        x_max, y_max = [x_val[y_val.argmax()], y_val.max()]
    else:
        x_max, y_max = [x_val[y_val.argmin()], y_val.min()]

    if log_y:
        return x_max, np.log(y_val)
    else:
        return x_max, y_max

def normalization(orig_score, norm_mean, norm_std):
    # Returns z- or t- (depending on input) normalized log-likelihood
    score
    # orig_score : single llr-score from speaker verification test
    # norm_mean :  estimated mean for normalization
    # norm_std :   estimated standard deviation for normalization

    # Return normalized score
    return (orig_score-norm_mean)/norm_std

def get_mean_std(norm_scores):
    # Returns estimated mean and standard deviation of elements in
    norm_scores
    # mean: estimated mean of elements in norm_scores
    # std: estimated standard deviation of elements in norm_scores

    norm_scores_np = np.array(norm_scores)

    return [norm_scores_np.mean(),norm_scores_np.std()]

def z_normalization(scoremat, z_norm_est):
    # Returns a z-normalized matrix of scores
    # scoremat:   matrix of scores from the Sidekit.bosaris.Scores object
    # z_norm_est: an array of estimated zero normalization variables

    assert len(z_norm_est) == scoremat.shape[0], 'scoremat and z_norm_est
    dimension mismatch'

    scoremat_norm = copy.deepcopy(scoremat)
    for i in range(scoremat.shape[0]):
        if z_norm_est[i] != []:
            for j in range(scoremat.shape[1]):
                scoremat_norm[i,j] = normalization(scoremat_norm[i,j],
    z_norm_est[i][0],z_norm_est[i][1])
    return scoremat_norm

def t_normalization(scoremat, t_norm_est, enroll_range, norm_range):
    # Returns a t-normalized matrix of scores
    # scoremat :     matrix of scores from the Sidekit.bosaris.Scores
    object
```

```
    # t_norm_est :   an array of estimated t-normalization variables
    # enroll_range : number of enrollment speakers (assumed equal for m/f)
    # norm_range :   number of normalization speakers (assumed equal for m/f)

    assert len(t_norm_est) == scoremat.shape[1], 'scoremat and t_norm_est dimension mismatch'

    scoremat_norm = copy.deepcopy(scoremat)
    for i in range(scoremat.shape[1]):
        for j in [k for k in range(enroll_range)]+[k for k in range(norm_range,norm_range+enroll_range)]+[scoremat.shape[0]-1]:
            scoremat_norm[j,i] = normalization(scoremat_norm[j,i], t_norm_est[i][0],t_norm_est[i][1])
    return scoremat_norm


def estimate_z_norm_variables(enroll_range, norm_range, modelset, scoremat, norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m):
    # Get z-norm variables for each model
    z_norm_est = []
    num_models, num_segments = scoremat.shape

    for i in range(num_models):
        if i < enroll_range or (i >= norm_range and i < norm_range+enroll_range): # z-norm is only applied to enrolled models
            if modelset[i][0] == 'f':
                z_norm_est.append(get_mean_std(scoremat[i,norm_seg_idx_f:enroll_seg_idx_m]))
            elif modelset[i][0] == 'm':
                z_norm_est.append(get_mean_std(scoremat[i,norm_seg_idx_m:num_segments]))
            else:
                raise Exception('Unknown model ID in scores.modelset: '+modelset[i])
        else:
            z_norm_est.append([])
    # finally, add the the complementary model's z-norm variables
    z_norm_est[num_models-1] = get_mean_std(scoremat[num_models-1,norm_seg_idx_f:enroll_seg_idx_m]+scoremat[num_models-1,norm_seg_idx_m:num_segments])
    return z_norm_est

def estimate_t_norm_variables(segset, scoremat, enroll_range, norm_range, modelset):
    # Get t-norm variables for each enrollment speaker
    # All trial segments are scored against a number of speaker models outside the enrollment set (the norm models),
    # then the mean and standard deviation of these scores are calculated and used for score normalization
    t_norm_est = []
    num_models, num_segments = scoremat.shape

    for i in range(num_segments):
        # t-norm is applied to all trial segments (excluding normalization segments)
        if segset[i].split('/')[2][0] == 'f':
```

```
109             t_norm_est.append(get_mean_std(scoremat[enroll_range:
    norm_range,i]))
110         elif segset[i].split('/')[2][0] == 'm':
111             t_norm_est.append(get_mean_std(scoremat[norm_range+
    enroll_range:num_models-1,i]))
112         else:
113             raise Exception('Unknown model ID in scores.modelset: '+
    modelset[i])
114     return t_norm_est
115 
116 def get_raw_scores(scoremat, target_mask, nontarget_mask, enroll_range,
    norm_range):
117     # Returns raw scores from a speaker identification experiment
118     # In:
119     # scoremat :            matrix of scores
120     # target_mask :         boolean mask defining target trials in
    scoremat
121     # nontarget_mask :      boolean mask defining nontarget trials in
    scoremat
122     # enroll_range :        number of enrollment speakers (assumed equal
    for m/f)
123     # norm_range :          number of normalization speakers (assumed
    equal for m/f)
124     # Out:
125     # score_target_raw :    target speaker scores
126     # score_nontarget_raw : nontarget speaker scores
127 
128     score_target_raw = []
129     score_nontarget_raw = []
130     num_models, num_segments = scoremat.shape
131 
132     for i in range(num_segments): # i denotes the index of the current
    trial
133         score_tar_f = scoremat[0:enroll_range,i][target_mask[0:
    enroll_range,i]]
134         score_tar_m = scoremat[norm_range:norm_range+enroll_range,i][
    target_mask[norm_range:norm_range+enroll_range,i]]
135         score_target_raw.append(np.concatenate([score_tar_f,score_tar_m]))
136         score_non_f = scoremat[0:enroll_range,i][nontarget_mask[0:
    enroll_range,i]]
137         score_non_m = scoremat[norm_range:norm_range+enroll_range,i][
    nontarget_mask[norm_range:norm_range+enroll_range,i]]
138         score_nontarget_raw.append(np.concatenate([score_non_f,score_non_m
    ,[scoremat[num_models-1,i]]]))
139 
140     return score_target_raw, score_nontarget_raw
141 
142 def estimate_diff_to_next(score_target, score_nontarget, norm_seg_idx_f,
    enroll_seg_idx_m, norm_seg_idx_m):
143     # Returns an estimate of the mean distance between target and second
    best nontarget scores
144     # The estimate, diff_to_next, will be estimated from a subset of the
    smallest distances
145     # The relative size of this subset is determined by the value of
    subset_ratio
146     # In:
147     # score_target :        array of target scores
```

```
148     # score_nontarget :     array of nontarget scores
149     # norm_seg_idx_f :      index of female normalization segments in the
        score arrays
150     # enroll_seg_idx_m :    index of male enrollment segments
151     # norm_seg_idx_m :      index of male normalization segments
152     # Out:
153     # diff_to_next :        estimated mean distance between target and
        nontarget scores
154
155     subset_ratio = 5 # 1/subset_ration is the relative size of the subset
        of the smallest distances in diff_vec
156     num_segments = len(score_target)
157
158     # Make array of differences between the two highest scores
159     diff_vec_tg = np.array([])
160     diff_vec_ntg = np.array([])
161     for i in [j for j in range(0,norm_seg_idx_f)]+[j for j in range(
        enroll_seg_idx_m,norm_seg_idx_m)]:
162         # Target trials:
163         # Go through all scores, and pick only the cases where the target
        score is the largest one
164         if score_target[i].size != 0 and score_target[i].max() >
        score_nontarget[i].max():
165             diff_vec_tg = np.append(diff_vec_tg, [score_target[i].max()-
        score_nontarget[i].max()])
166         # Nontarget trials:
167         # Add all differences between best and second best score
168         if score_target[i].size == 0:
169             diff_vec_ntg = np.append(diff_vec_ntg, [score_nontarget[i].max
        ()-get_second_largest(score_nontarget[i])])
170
171     # Estimate mean difference from subset
172     diff_vec_tg = np.sort(diff_vec_tg)
173
174     #print(diff_vec_tg[0:100])
175     #print(diff_vec_ntg[0:100])
176     #print('Total number of highest target scores: '+str(len(diff_vec_tg))
        )
177     #print('Scores included in estimation: '+str(len(diff_vec_tg)/
        subset_ratio))
178
179     diff_mean = diff_vec_tg[0:len(diff_vec_tg)/subset_ratio].mean()
180
181     #fr_num = 0
182     #for diff in diff_vec_tg:
183     #    if diff < diff_mean:
184     #        fr_num += 1
185     #print('Number of false rejections: '+str(fr_num))
186
187     return diff_mean
188
189 def naive_decision(norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m,
        score_target, score_nontarget):
190     # Performs naive decision, i.e. chooses the models that gives the
191     # highest score after a trial, and assigns the segment the identity
192     # of that model.
193     # In:
```

```
    # norm_seg_idx_f :   index of female normalization segments in the original score objects
    # enroll_seg_idx_m : index of male enrollment segments
    # norm_seg_idx_m :   index of male normalization segments
    # score_target :     array of target scores
    # score_nontarget :  array of nontarget scores
    # Out:
    # rr :               overall recognition rate
    # far :              false acceptance ratio
    # frr :              false rejection ratio
    # ear :              erroneous acceptance ratio

    correct_trials = 0
    correct_target_trials = 0
    correct_nontarget_trials = 0
    num_target_trials = 0 # total number of target trials
    num_fa = 0 # number of false acceptions
    num_fr = 0 # number of false rejections
    num_ea = 0 # erroneous acceptance (speaker is classified as the wrong enrollement speaker)
    num_trials = norm_seg_idx_f+norm_seg_idx_m-enroll_seg_idx_m

    for i in [j for j in range(0,norm_seg_idx_f)]+[j for j in range(enroll_seg_idx_m,norm_seg_idx_m)]:
        # Unknown test speaker: Correct trial if complementary model has the highest score
        if score_target[i].size == 0:
            if score_nontarget[i].argmax() == len(score_nontarget[i])-1:
                correct_trials += 1 # unknown speaker is rejected as it scores highest for comp. model
                correct_nontarget_trials += 1
            else:
                num_fa += 1 # unknown speaker is falsely accepted as one of the enrollment speakers
        # Known (enrolled) test speaker: Correct trial if the speakers' enrolled model
        # has the highest score
        else: # enrolled test speakers
            num_target_trials += 1
            if score_target[i].max() > score_nontarget[i].max():
                correct_trials += 1
                correct_target_trials += 1
            elif score_nontarget[i].argmax() == len(score_nontarget[i])-1:
                num_fr += 1 # known speaker is falsely rejected
            else:
                num_ea += 1 # known speaker is identified as the wrong known speaker

    rr = correct_trials/float(num_trials)
    far = num_fa/float(num_trials)
    frr = num_fr/float(num_trials)
    ear = num_ea/float(num_trials)

    print('Number of correct target trials: '+str(correct_target_trials) \
              +' of '+str(num_target_trials)+' target trials' \
              + '; ' + str(round(correct_target_trials/float(num_target_trials)*100,2))+'%')
```

```
242     print('Number of correct nontarget trials: '+str(
        correct_nontarget_trials) \
243               +' of '+str(num_trials-num_target_trials)+' nontarget trials
        ' \
244               + '; ' + str(round(correct_nontarget_trials/float(num_trials
        -num_target_trials)*100,2))+'%')
245 
246     return rr, far, frr, ear
247 
248 
249 def threshold_decision_strict(norm_seg_idx_f, enroll_seg_idx_m,
        norm_seg_idx_m, score_target, score_nontarget, threshold):
250     # Performs threshold decision, i.e. chooses the models that gives the
251     # highest score after a trial, and assigns the segment the identity
252     # of that model, if the score is above a set threshold value.
253     # In:
254     # norm_seg_idx_f :   index of female normalization segments in the
255     #                    original score object
256     # enroll_seg_idx_m : index of male enrollment segments
257     # norm_seg_idx_m :   index of male normalization segments
258     # score_target :     array of target scores
259     # score_nontarget :  array of nontarget scores
260     # treshold :         threshold value
261     # Out:
262     # rr :               overall recognition rate
263     # far :              false acceptance ratio
264     # frr :              false rejection ratio
265     # ear :              erroneous acceptance ratio
266 
267     correct_trials = 0
268     num_fa = 0 # number of false acceptions
269     num_fr = 0 # number of false rejections
270     num_ea = 0 # erroneous acceptance (speaker is classified as the wrong
        enrollment speaker)
271     num_trials = norm_seg_idx_f+norm_seg_idx_m-enroll_seg_idx_m
272 
273     for i in [j for j in range(0,norm_seg_idx_f)]+[j for j in range(
        enroll_seg_idx_m,norm_seg_idx_m)]:
274         # Unknown test speaker: Correct trial if complementary model has
        the highest score
275         if score_target[i].size == 0:
276             if score_nontarget[i].argmax() == len(score_nontarget[i])-1:
277                 correct_trials += 1 # unknown speaker is assigned to comp.
         models, i.e. rejected
278             elif score_nontarget[i].max() < threshold:
279                 correct_trials += 1 # unknown speaker scores lower than
        threshold for all models, rejection
280             elif get_second_largest(score_nontarget[i]) > threshold:
281                 correct_trials += 1 # unknown speaker scores higher for
        two or more models, rejection
282             else:
283                 num_fa += 1 # unknown speaker is falsely accepted as one
        of the enrollment speakers
284         # Known (enrolled) test speaker: Correct trial if the speakers'
        enrolled model
285         # has the highest score
286         else:
```

```
            if score_target[i].max() > threshold and score_nontarget[i].max() < threshold:
                correct_trials += 1 # known speaker is assigned to its own model
            elif score_nontarget[i].argmax() != len(score_nontarget[i])-1 and score_nontarget[i].max() > threshold:
                if get_second_largest(score_nontarget[i]) < threshold:
                    num_ea += 1 # known speaker is assigned to the wrong speaker model
                else:
                    num_fr += 1 # known speaker is falsely rejected
            else:
                num_fr += 1 # known speaker is falsely rejected

    rr = correct_trials/float(num_trials)
    far = num_fa/float(num_trials)
    frr = num_fr/float(num_trials)
    ear = num_ea/float(num_trials)

    return rr, far, frr, ear

def threshold_decision(norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m, score_target, score_nontarget, threshold, diff_to_next):
    # Performs threshold decision, i.e. chooses the models that gives the
    # highest score after a trial, and assigns the segment the identity
    # of that model, if the score is above a set threshold value.
    # It differs from threshold_decision_strict in the sense that trial scores are not rejected if
    # there are more than one score above the threshold value. A score is rejected if the difference
    # to the second best value is larger than diff_to_next
    # In:
    # norm_seg_idx_f :   index of female normalization segments in the
    #                    original score object
    # enroll_seg_idx_m : index of male enrollment segments
    # norm_seg_idx_m :   index of male normalization segments
    # score_target :     array of target scores
    # score_nontarget :  array of nontarget scores
    # treshold :         threshold value
    # diff_to_next :     minimum acceptable difference between best and second best score
    # Out:
    # rr :               overall recognition rate
    # far :              false acceptance ratio
    # frr :              false rejection ratio
    # ear :              erroneous acceptance ratio

    correct_trials = 0
    correct_target_trials = 0
    correct_nontarget_trials = 0
    num_fa = 0 # number of false acceptions
    num_fr = 0 # number of false rejections
    num_ea = 0 # erroneous acceptance (speaker is classified as the wrong enrollment speaker)
    num_trials = norm_seg_idx_f+norm_seg_idx_m-enroll_seg_idx_m

    for i in [j for j in range(0,norm_seg_idx_f)]+[j for j in range(
```

```
    enroll_seg_idx_m,norm_seg_idx_m)]:
        # Unknown test speaker: Correct trial if complementary model has
    the highest score
        if score_target[i].size == 0:
            if score_nontarget[i].argmax() == len(score_nontarget[i])-1:
                correct_trials += 1 # unknown speaker is assigned to comp.
     models, i.e. rejected
                correct_nontarget_trials += 1
            elif score_nontarget[i].max() < threshold:
                correct_trials += 1 # unknown speaker scores lower than
    threshold for all models, rejection
                correct_nontarget_trials += 1
            elif score_nontarget[i].max() - get_second_largest(
    score_nontarget[i]) < diff_to_next:
                correct_trials += 1 # difference between best and second
    best score too small, rejection
                correct_nontarget_trials += 1
            else:
                num_fa += 1 # unknown speaker is falsely accepted as one
    of the enrollment speakers
        # Known (enrolled) test speaker: Correct trial if the speakers'
    enrolled model
        # has the highest score
        else:
            if score_target[i].max() > threshold and score_target[i].max()
     - score_nontarget[i].max() > diff_to_next:
                correct_trials += 1 # known speaker is assigned to its own
     model
                correct_target_trials += 1
            elif score_nontarget[i].argmax() != len(score_nontarget[i])-1
    and score_nontarget[i].max() > threshold:
                if score_nontarget[i].max() - get_second_largest(
    score_nontarget[i]) > diff_to_next:
                    num_ea += 1 # known speaker is assigned to the wrong
    speaker model
                else:
                    num_fr += 1 # known speaker is falsely rejected
            else:
                num_fr += 1 # known speaker is falsely rejected


    rr = correct_trials/float(num_trials)
    far = num_fa/float(num_trials)
    frr = num_fr/float(num_trials)
    ear = num_ea/float(num_trials)

    return rr, far, frr, ear, correct_target_trials,
    correct_nontarget_trials


def moving_threshold_decision(norm_seg_idx_f, enroll_seg_idx_m,
    norm_seg_idx_m, score_target, score_nontarget, scoremat, num_points
    =100, additional_threshold=1):
    # Performs moving threshold decision
    # In:
    # norm_seg_idx_f :       index of female normalization segments in the
    #                        original score object
```

```
    # enroll_seg_idx_m :     index of male enrollment segments
    # norm_seg_idx_m :       index of male normalization segments
    # score_target :         array of target scores
    # score_nontarget :      array of nontarget scores
    # scoremat :             scoremat from a Sidekit.bosaris.Scores object
    # num_points :           resolution of output arrays
    # additional_threshold : additional threshold so that all threshold
    values are included
    # Out:
    # rr :                   array of overall recognition rate
    # far :                  array of false acceptance ratio
    # frr :                  array of false rejection ratio
    # ear :                  array of erroneous acceptance ratio
    # threshold_array :      threshold array for plots
    # NOTE: if function get_raw_scores is edited to return numpy arrays,
    # scoremat is not needed for finding max and min values of scores

    # Deciding min and max values of threshold
    threshold_min = min(scoremat[:,:norm_seg_idx_f].min(), scoremat[:,
    enroll_seg_idx_m:norm_seg_idx_m].min())
    threshold_max = max(scoremat[:,:norm_seg_idx_f].max(), scoremat[:,
    enroll_seg_idx_m:norm_seg_idx_m].max()) + additional_threshold
    # Initialize lists:
    rr, far, frr, ear, tar_correct, nontar_correct = np.array([]), np.
    array([]), np.array([]), np.array([]), np.array([]), np.array([])
    threshold_array = np.linspace(threshold_min, threshold_max, num_points
    )

    # Estimate diff_to_next from score
    diff_to_next = estimate_diff_to_next(score_target, score_nontarget,
    norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m)

    # Do decision for moving threshold
    for threshold in threshold_array:
        tmp_rr, tmp_far, tmp_frr, tmp_ear, correct_target_trials,
    correct_nontarget_trials = threshold_decision(norm_seg_idx_f,
                                          enroll_seg_idx_m,
                                          norm_seg_idx_m,
                                          score_target,
                                          score_nontarget,
                                          threshold,
                                          diff_to_next)
        rr = np.append(rr,tmp_rr)
        far = np.append(far,tmp_far)
        frr = np.append(frr,tmp_frr)
        ear = np.append(ear,tmp_ear)
        tar_correct = np.append(tar_correct,correct_target_trials)
        nontar_correct = np.append(nontar_correct,correct_nontarget_trials
    )
```

```
    # Display error rates for target and nontarget trials
    print('Maximum number of correct target trials: '+str(int(tar_correct.max()))\
              +' of 800 target trials; '+str(round(tar_correct.max()/800*100,2))+'%')
    tar_max_idx = int(tar_correct.argmax())
    print('Corresponding number of correct nontarget trials: '+str(int(nontar_correct[tar_max_idx]))\
              +' of 200 nontarget trials; '+str(round(nontar_correct[tar_max_idx]/200*100,2))+'%')


    return rr, far, frr, ear, threshold_array
```

GMM-UBM score evaluation: GMM-UBM/evaluate_GMM_UBM.py

```
# This script evaluates scores from a GMM-UBM speaker identification experiment
# z-/t-/zt-normalization is applied to scores before evaluation
# It is also possible to exclude the complementary model from the evaluation
#
# The system performance is measured and displayed in two ways:
# - Maximum recognition rate. The maximum recognition rate, for a given value
#   of the decision value
# - A plot of recognition rate vs. decision value
#
# Further:
# - score_file contains the actual scores against enrollment models, normalization
#   speaker models and the complementary model (w)
# - key_file contains a mapping between speaker models and test segments scored
#   on them, with boolean matrices indicating whether the trials are target/nontarget
#
# Written by Joergen Antonsen, April 26 2017

import sidekit
import numpy
import os
import sys
import normalize_gmm_ubm
from normalize_gmm_ubm import get_mean_std
from normalize_gmm_ubm import naive_decision
from normalize_gmm_ubm import get_raw_scores
from normalize_gmm_ubm import estimate_z_norm_variables
from normalize_gmm_ubm import estimate_t_norm_variables
from normalize_gmm_ubm import z_normalization
from normalize_gmm_ubm import t_normalization
from normalize_gmm_ubm import moving_threshold_decision
from normalize_gmm_ubm import get_global_minmax
import matplotlib.pyplot as plt
import matplotlib.lines as mlines # for legend

assert len(sys.argv) == 5, 'Usage: '+sys.argv[0]+' <score_file_path>'+\
```

```
    +' <number of enrollment wakewords>'\
    +' <number of enrollment sentences>\n'\
    +' <use more test data {0,1}>'\
    +'Example: '+sys.argv[0]+' scores/ww_3_stc_3/
    scores_gmm_ubm_gidp_100_ww_3_sct_3.h5 3 3 1'

print('GMM-UBM SYSTEM EVALUATION')
print('- score file: '+sys.argv[1])

# -----------------Set parameters------------------
test_set_size = 10 # number of speakers in the test set
num_unknown = 2 # number of unknown speakers in the test set
enroll_set_size = test_set_size-num_unknown # number of known speakers in
    test set
num_set_size = 100 # number of unique test sets

enroll_range = 50
norm_range = 100

ww_adapt_num = int(sys.argv[2]) # number of wake words for model
    enrollment
stc_adapt_num = int(sys.argv[3]) # number of sentences for model
    enrollment

score_file = sys.argv[1]

plot_fig = True # saves recognition rate plots as png if True
save_results = True # saves recognition rates as numpy arrays if True
more_test_data = sys.argv[4]

if more_test_data == '0':
    result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num
    )
    norm_score_dir = 'scores/normalized/ww_'+str(ww_adapt_num)+'_stc_'+str
    (stc_adapt_num)
    fig_dir = 'fig/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
elif more_test_data == '1':
    result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+
    str(stc_adapt_num)
    norm_score_dir = 'scores/more_test_data/normalized/ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)
    fig_dir = 'fig/more_test_data/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(
    stc_adapt_num)
else:
    raise ValueError('Invalid value of parameter <use more test data
    {0,1}>: '+more_test_data)
if not os.path.exists(result_dir):
    os.makedirs(result_dir)
if not os.path.exists(norm_score_dir):
    os.makedirs(norm_score_dir)
if not os.path.exists(fig_dir):
    os.makedirs(fig_dir)

# -----------Read scores and test files------------
scores = sidekit.bosaris.Scores(score_file)
if more_test_data == '0':
    key = sidekit.Key('../task/gmm_ubm/eval_gidp_10_spk_100_sets_key.h5')
```

```
elif more_test_data == '1':
    key = sidekit.Key('../task/gmm_ubm/more_data/
    eval_gidp_10_spk_100_sets_more_data_key.h5')


# -----Manage variables and sort segment indices----
modelset = scores.modelset # set of models (f001, f002, ..., m099, m100, w
    )
segset = scores.segset # set of segments
scoremask = scores.scoremask # score mask, True for element (i,j) if
    segment j is tested on model i
scoremat = scores.scoremat # score matrix, contain llr scores from all
    trials
target_mask = key.tar # matrix that is True for all target trials
nontarget_mask = key.non # matrix that is True for all nontarget trials

# Get segment indices
num_models, num_segments = scoremat.shape

enroll_seg_idx_f = 0 # segment index of female enrollment speakers
norm_seg_idx_f = 0 # segment index of female norm speakers (temp value)
enroll_seg_idx_m = 0 # segment index of male enrollment speakers (temp
    value)
norm_seg_idx_m = 0 # segment index of male norm speakers (temp value)
i = 0

while int(segset[i].split('/')[2][1:4]) <= enroll_range:
    i += 1
norm_seg_idx_f = i

while segset[i].split('/')[2][0] != 'm':
    i += 1
enroll_seg_idx_m = i

while int(segset[i].split('/')[2][1:4]) <= enroll_range:
    i += 1
norm_seg_idx_m = i

# NOTE: The above code will not work unless segset elements have the
    format
#       <dir1>/<dir2>/{m,f}{001:xxx}<rest of filename>
#       e.g. cat/female/f001_01_032-049



# -------Estimate normalization variables-----------
# Get z-norm variables for each model
z_norm_est = estimate_z_norm_variables(enroll_range, norm_range, modelset,
     scoremat, norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m)

# Get t-norm variables for each enrollment speaker
t_norm_est = estimate_t_norm_variables(segset, scoremat, enroll_range,
    norm_range, modelset)


# ---------Calculate recognition rates-------------
# First get the 'raw', unprocessed scores
```

```
score_target_raw, score_nontarget_raw = get_raw_scores(scoremat,
    target_mask,nontarget_mask,enroll_range,norm_range)

# NAIVE DECISION - Choose the model that gives the highest score, without
    thresholding
print('NAIVE DECISION')
# 1: With raw scores
print('----------Using raw scores---------')
raw_recognition_rate,raw_far,raw_frr,raw_ear = naive_decision(
    norm_seg_idx_f,enroll_seg_idx_m,norm_seg_idx_m,score_target_raw,
    score_nontarget_raw)
print('Raw recognition rate: '+str(raw_recognition_rate*100)+'%')
print('False acceptance rate: '+str(raw_far*100)+'%')
print('False rejection rate: '+str(raw_frr*100)+'%')
print('Erroneous acceptance rate: '+str(raw_ear*100)+'%')

# 2: After z-norm
print('----------With z-normalization---------')
scoremat_z_norm = z_normalization(scoremat, z_norm_est)
score_target_z, score_nontarget_z = get_raw_scores(scoremat_z_norm,
    target_mask,nontarget_mask,enroll_range,norm_range)
z_recognition_rate,z_far,z_frr,z_ear = naive_decision(norm_seg_idx_f,
    enroll_seg_idx_m,norm_seg_idx_m,score_target_z,score_nontarget_z)
print('z-normalized recognition rate: '+str(z_recognition_rate*100)+'%')
print('False acceptance rate: '+str(z_far*100)+'%')
print('False rejection rate: '+str(z_frr*100)+'%')
print('Erroneous acceptance rate: '+str(z_ear*100)+'%')

# 3: After t-norm
print('----------With t-normalization---------')
scoremat_t_norm = t_normalization(scoremat, t_norm_est, enroll_range,
    norm_range)
score_target_t, score_nontarget_t = get_raw_scores(scoremat_t_norm,
    target_mask,nontarget_mask,enroll_range,norm_range)
t_recognition_rate,t_far,t_frr,t_ear = naive_decision(norm_seg_idx_f,
    enroll_seg_idx_m,norm_seg_idx_m,score_target_t,score_nontarget_t)
print('t-normalized recognition rate: '+str(t_recognition_rate*100)+'%')
print('False acceptance rate: '+str(t_far*100)+'%')
print('False rejection rate: '+str(t_frr*100)+'%')
print('Erroneous acceptance rate: '+str(t_ear*100)+'%')

# 4: After zt-norm
print('----------With zt-normalization---------')
scoremat_zt_norm = t_normalization(scoremat_z_norm, t_norm_est,
    enroll_range, norm_range)
score_target_zt, score_nontarget_zt = get_raw_scores(scoremat_zt_norm,
    target_mask,nontarget_mask,enroll_range,norm_range)
zt_recognition_rate,zt_far,zt_frr,zt_ear = naive_decision(norm_seg_idx_f,
    enroll_seg_idx_m,norm_seg_idx_m,score_target_zt,score_nontarget_zt)
print('zt-normalized recognition rate: '+str(zt_recognition_rate*100)+'%')
print('False acceptance rate: '+str(zt_far*100)+'%')
print('False rejection rate: '+str(zt_frr*100)+'%')
print('Erroneous acceptance rate: '+str(zt_ear*100)+'%')

# 5: After tz-norm
print('----------With tz-normalization---------')
scoremat_tz_norm = z_normalization(scoremat_t_norm, z_norm_est)
```

```
score_target_tz, score_nontarget_tz = get_raw_scores(scoremat_tz_norm,
    target_mask,nontarget_mask,enroll_range,norm_range)
tz_recognition_rate,tz_far,tz_frr,tz_ear = naive_decision(norm_seg_idx_f,
    enroll_seg_idx_m,norm_seg_idx_m,score_target_tz,score_nontarget_tz)
print('tz-normalized recognition rate: '+str(tz_recognition_rate*100)+'%')
print('False acceptance rate: '+str(tz_far*100)+'%')
print('False rejection rate: '+str(tz_frr*100)+'%')
print('Erroneous acceptance rate: '+str(tz_ear*100)+'%')

print('')



# DECISION WITH MOVING THRESHOLD - Choose model if score is solely above
    threshold
print('DECISION WITH MOVING THRESHOLD')

# 1: With raw scores
print('----------Using raw scores---------')
raw_rr,raw_far,raw_frr,raw_ear,raw_xaxis = moving_threshold_decision(
    norm_seg_idx_f,enroll_seg_idx_m,norm_seg_idx_m,score_target_raw,
    score_nontarget_raw,scoremat,num_points=500,additional_threshold=0.1)
print('Max recognition rate: '+str(raw_rr.max()*100)+'%')

# 2: With z-normalized scores
print('----------With z-normalization---------')
raw_rr_z,raw_far_z,raw_frr_z,raw_ear_z,raw_xaxis_z =
    moving_threshold_decision(norm_seg_idx_f,enroll_seg_idx_m,
    norm_seg_idx_m,score_target_z,score_nontarget_z,scoremat_z_norm,
    num_points=500,additional_threshold=1)
print('Max recognition rate: '+str(raw_rr_z.max()*100)+'%')

# 3: With t-normalized scores
print('----------With t-normalization---------')
raw_rr_t,raw_far_t,raw_frr_t,raw_ear_t,raw_xaxis_t =
    moving_threshold_decision(norm_seg_idx_f,enroll_seg_idx_m,
    norm_seg_idx_m,score_target_t,score_nontarget_t,scoremat_t_norm,
    num_points=500,additional_threshold=1)
print('Max recognition rate: '+str(raw_rr_t.max()*100)+'%')

# 4: With zt-normalized scores
print('----------With zt-normalization---------')
raw_rr_zt,raw_far_zt,raw_frr_zt,raw_ear_zt,raw_xaxis_zt =
    moving_threshold_decision(norm_seg_idx_f,enroll_seg_idx_m,
    norm_seg_idx_m,score_target_zt,score_nontarget_zt,scoremat_zt_norm,
    num_points=500,additional_threshold=1)
print('Max recognition rate: '+str(raw_rr_zt.max()*100)+'%')

# 5: With tz-normalized scores
print('----------With tz-normalization---------')
raw_rr_tz,raw_far_tz,raw_frr_tz,raw_ear_tz,raw_xaxis_tz =
    moving_threshold_decision(norm_seg_idx_f,enroll_seg_idx_m,
    norm_seg_idx_m,score_target_tz,score_nontarget_tz,scoremat_tz_norm,
    num_points=500,additional_threshold=1)
print('Max recognition rate: '+str(raw_rr_tz.max()*100)+'%')



```

```
# -----------------Save results--------------------
if save_results:
    print('Saving results')
    # 0: Score matrices from normalization
    scores_norm = sidekit.bosaris.Scores()
    scores_norm.scoremask = scoremask
    scores_norm.modelset = modelset
    scores_norm.segset = segset
    # - z-norm scores
    scores_norm.scoremat = scoremat_z_norm
    scores_norm.write(os.path.join(norm_score_dir,
                                   'scores_gmm_ubm_enroll_gidp_100_ww_{}
    _stc_{}_z.h5'\
                                       .format(str(ww_adapt_num),str(
    stc_adapt_num))))
    # - t-norm scores
    scores_norm.scoremat = scoremat_t_norm
    scores_norm.write(os.path.join(norm_score_dir,
                                   'scores_gmm_ubm_enroll_gidp_100_ww_{}
    _stc_{}_t.h5'\
                                       .format(str(ww_adapt_num),str(
    stc_adapt_num))))
    # - zt-norm scores
    scores_norm.scoremat = scoremat_zt_norm
    scores_norm.write(os.path.join(norm_score_dir,
                                   'scores_gmm_ubm_enroll_gidp_100_ww_{}
    _stc_{}_zt.h5'\
                                       .format(str(ww_adapt_num),str(
    stc_adapt_num))))
    # - tz-norm scores
    scores_norm.scoremat = scoremat_tz_norm
    scores_norm.write(os.path.join(norm_score_dir,
                                   'scores_gmm_ubm_enroll_gidp_100_ww_{}
    _stc_{}_tz.h5'\
                                       .format(str(ww_adapt_num),str(
    stc_adapt_num))))

    print('- normalized score matrices saved')

    # 1: Raw scores
    numpy.save(os.path.join(result_dir,'raw','rr'), raw_rr)
    numpy.save(os.path.join(result_dir,'raw','far'), raw_far)
    numpy.save(os.path.join(result_dir,'raw','frr'), raw_frr)
    numpy.save(os.path.join(result_dir,'raw','ear'), raw_ear)
    numpy.save(os.path.join(result_dir,'raw','xaxis'), raw_xaxis)
    print('- raw results saved')

    # 2: z-normalized scores
    numpy.save(os.path.join(result_dir,'z','rr'), raw_rr_z)
    numpy.save(os.path.join(result_dir,'z','far'), raw_far_z)
    numpy.save(os.path.join(result_dir,'z','frr'), raw_frr_z)
    numpy.save(os.path.join(result_dir,'z','ear'), raw_ear_z)
    numpy.save(os.path.join(result_dir,'z','xaxis'), raw_xaxis_z)
    print('- z-normalized results saved')

    # 3: t-normalized scores
    numpy.save(os.path.join(result_dir,'t','rr'), raw_rr_t)
```

```
    numpy.save(os.path.join(result_dir,'t','far'), raw_far_t)
    numpy.save(os.path.join(result_dir,'t','frr'), raw_frr_t)
    numpy.save(os.path.join(result_dir,'t','ear'), raw_ear_t)
    numpy.save(os.path.join(result_dir,'t','xaxis'), raw_xaxis_t)
    print('- t-normalized results saved')

    # 4: zt-normalized scores
    numpy.save(os.path.join(result_dir,'zt','rr'), raw_rr_zt)
    numpy.save(os.path.join(result_dir,'zt','far'), raw_far_zt)
    numpy.save(os.path.join(result_dir,'zt','frr'), raw_frr_zt)
    numpy.save(os.path.join(result_dir,'zt','ear'), raw_ear_zt)
    numpy.save(os.path.join(result_dir,'zt','xaxis'), raw_xaxis_zt)
    print('- zt-normalized results saved')

    # 5: tz-normalized scores
    numpy.save(os.path.join(result_dir,'tz','rr'), raw_rr_tz)
    numpy.save(os.path.join(result_dir,'tz','far'), raw_far_tz)
    numpy.save(os.path.join(result_dir,'tz','frr'), raw_frr_tz)
    numpy.save(os.path.join(result_dir,'tz','ear'), raw_ear_tz)
    numpy.save(os.path.join(result_dir,'tz','xaxis'), raw_xaxis_tz)
    print('- tz-normalized results saved')


# ------------Plot recognition rate----------------
# For moving threshold scoring
if plot_fig:
    print('-------------Save plots-------------')
    # ----------Raw scores----------
    fig = plt.figure()
    ax = fig.add_subplot(111)
    rr, = ax.plot(raw_xaxis,raw_rr,'r')
    far, = ax.plot(raw_xaxis,raw_far,'g')
    frr, = ax.plot(raw_xaxis,raw_frr,'b')
    ear = ax.plot(raw_xaxis,raw_ear,'m')
    red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
    green_line = mlines.Line2D([],[],color='green',label='False acceptance
     rate')
    blue_line = mlines.Line2D([],[],color='blue',label='False rejection
    rate')
    mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
    acceptance rate')
    ax.legend(handles=[red_line,green_line,blue_line,mag_line])
    ax.set_xlabel('Threshold')

    # Annotate maxima/minima
    rr_max_x, rr_max_y = get_global_minmax(raw_xaxis,raw_rr,'max')
    ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
    'data')
    far_max_x, far_max_y = get_global_minmax(raw_xaxis,raw_far,'max')
    ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
    textcoords='data')
    frr_max_x, frr_max_y = get_global_minmax(raw_xaxis,raw_frr,'min')
    ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
    textcoords='data')
    ear_max_x, ear_max_y = get_global_minmax(raw_xaxis,raw_ear,'max')
    ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
    textcoords='data')
```

```
316     ax.set_title('Raw results')
317     fig.savefig(os.path.join(fig_dir,'rr_raw_gidp_'+str(num_set_size)+'
        _sets_'+str(test_set_size)+'_spk_ww_'+str(ww_adapt_num)+'_stc_'+str(
        stc_adapt_num)+'.png'))
318     ax.set_yscale('log')
319     ax.set_title('Raw results - log')
320     fig.savefig(os.path.join(fig_dir,'rr_raw_gidp_'+str(num_set_size)+'
        _sets_'+str(test_set_size)+'_spk_ww_'+str(ww_adapt_num)+'_stc_'+str(
        stc_adapt_num)+'_log.png'))
321     print('- saved raw scores')
322
323     # ----------z-normalized scores----------
324     fig = plt.figure()
325     ax = fig.add_subplot(111)
326     rr, = ax.plot(raw_xaxis_z,raw_rr_z,'r')
327     far, = ax.plot(raw_xaxis_z,raw_far_z,'g')
328     frr, = ax.plot(raw_xaxis_z,raw_frr_z,'b')
329     ear = ax.plot(raw_xaxis_z,raw_ear_z,'m')
330     red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
331     green_line = mlines.Line2D([],[],color='green',label='False acceptance
         rate')
332     blue_line = mlines.Line2D([],[],color='blue',label='False rejection
        rate')
333     mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
        acceptance rate')
334     ax.legend(handles=[red_line,green_line,blue_line,mag_line])
335     ax.set_xlabel('Threshold')
336
337     # Annotate maxima/minima
338     rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_z,raw_rr_z,'max')
339     ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
        'data')
340     far_max_x, far_max_y = get_global_minmax(raw_xaxis_z,raw_far_z,'max')
341     ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
        textcoords='data')
342     frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_z,raw_frr_z,'min')
343     ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
        textcoords='data')
344     ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_z,raw_ear_z,'max')
345     ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
        textcoords='data')
346     ax.set_title('z-normalized results')
347     fig.savefig(os.path.join(fig_dir,'rr_z_norm_gidp_'+str(num_set_size)+'
        _sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
        ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.png'))
348     ax.set_yscale('log')
349     ax.set_title('z-normalized results - log')
350     fig.savefig(os.path.join(fig_dir,'rr_z_norm_gidp_'+str(num_set_size)+'
        _sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
        ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_log.png'))
351     print('- saved z-normalized scores')
352
353
354     # ----------t-normalized scores----------
355     fig = plt.figure()
356     ax = fig.add_subplot(111)
357     rr, = ax.plot(raw_xaxis_t,raw_rr_t,'r')
```

```
    far, = ax.plot(raw_xaxis_t,raw_far_t,'g')
    frr, = ax.plot(raw_xaxis_t,raw_frr_t,'b')
    ear = ax.plot(raw_xaxis_t,raw_ear_t,'m')
    red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
    green_line = mlines.Line2D([],[],color='green',label='False acceptance
     rate')
    blue_line = mlines.Line2D([],[],color='blue',label='False rejection
    rate')
    mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
    acceptance rate')
    ax.legend(handles=[red_line,green_line,blue_line,mag_line])
    ax.set_xlabel('Threshold')

    # Annotate maxima/minima
    rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_t,raw_rr_t,'max')
    ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
    'data')
    far_max_x, far_max_y = get_global_minmax(raw_xaxis_t,raw_far_t,'max')
    ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
    textcoords='data')
    frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_t,raw_frr_t,'min')
    ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
    textcoords='data')
    ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_t,raw_ear_t,'max')
    ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
    textcoords='data')
    ax.set_title('t-normalized results')
    fig.savefig(os.path.join(fig_dir,'rr_t_norm_gidp_'+str(num_set_size)+'
    _sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.png'))
    ax.set_yscale('log')
    ax.set_title('t-normalized results - log')
    fig.savefig(os.path.join(fig_dir,'rr_t_norm_gidp_'+str(num_set_size)+'
    _sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_log.png'))
    print('- saved t-normalized scores')

    # ----------zt-normalized scores----------
    fig = plt.figure()
    ax = fig.add_subplot(111)
    rr, = ax.plot(raw_xaxis_zt,raw_rr_zt,'r')
    far, = ax.plot(raw_xaxis_zt,raw_far_zt,'g')
    frr, = ax.plot(raw_xaxis_zt,raw_frr_zt,'b')
    ear = ax.plot(raw_xaxis_zt,raw_ear_zt,'m')
    red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
    green_line = mlines.Line2D([],[],color='green',label='False acceptance
     rate')
    blue_line = mlines.Line2D([],[],color='blue',label='False rejection
    rate')
    mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
    acceptance rate')
    ax.legend(handles=[red_line,green_line,blue_line,mag_line])
    ax.set_xlabel('Threshold')

    # Annotate maxima/minima
    rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_zt,raw_rr_zt,'max')
    ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
```

```
    'data')
401 far_max_x, far_max_y = get_global_minmax(raw_xaxis_zt,raw_far_zt,'max'
    )
402 ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
    textcoords='data')
403 frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_zt,raw_frr_zt,'min'
    )
404 ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
    textcoords='data')
405 ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_zt,raw_ear_zt,'max'
    )
406 ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
    textcoords='data')
407 ax.set_title('zt-normalized results')
408 fig.savefig(os.path.join(fig_dir,'rr_zt_norm_gidp_'+str(num_set_size)+
    '_sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.png'))
409 ax.set_yscale('log')
410 ax.set_title('zt-normalized results - log')
411 fig.savefig(os.path.join(fig_dir,'rr_zt_norm_gidp_'+str(num_set_size)+
    '_sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_log.png'))
412 print('- saved zt-normalized scores')
413
414 # ----------tz-normalized scores----------
415 fig = plt.figure()
416 ax = fig.add_subplot(111)
417 rr, = ax.plot(raw_xaxis_tz,raw_rr_tz,'r')
418 far, = ax.plot(raw_xaxis_tz,raw_far_tz,'g')
419 frr, = ax.plot(raw_xaxis_tz,raw_frr_tz,'b')
420 ear = ax.plot(raw_xaxis_tz,raw_ear_tz,'m')
421 red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
422 green_line = mlines.Line2D([],[],color='green',label='False acceptance
     rate')
423 blue_line = mlines.Line2D([],[],color='blue',label='False rejection
    rate')
424 mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
    acceptance rate')
425 ax.legend(handles=[red_line,green_line,blue_line,mag_line])
426 ax.set_xlabel('Threshold')
427
428 # Annotate maxima/minima
429 rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_tz,raw_rr_tz,'max')
430 ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
    'data')
431 far_max_x, far_max_y = get_global_minmax(raw_xaxis_tz,raw_far_tz,'max'
    )
432 ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
    textcoords='data')
433 frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_tz,raw_frr_tz,'min'
    )
434 ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
    textcoords='data')
435 ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_tz,raw_ear_tz,'max'
    )
436 ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
    textcoords='data')
```

```
    ax.set_title('tz-normalized results')
    fig.savefig(os.path.join(fig_dir,'rr_tz_norm_gidp_'+str(num_set_size)+'_sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.png'))
    ax.set_yscale('log')
    ax.set_title('tz-normalized results - log')
    fig.savefig(os.path.join(fig_dir,'rr_tz_norm_gidp_'+str(num_set_size)+'_sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_log.png'))
    print('- saved tz-normalized scores')



print('Done!')
```

GMM-UBM plotting: GMM-UBM/plots.py

```
# This script contains a few plot functions for presenting results

# Written by Joergen Antonsen May 16, 2017

import numpy
import os
from sidekit.bosaris import PlotWindow # for DET-plot
from sidekit.bosaris.detplot import __probit__ # for DET-plot
import matplotlib.pyplot as plt
import matplotlib.lines as mlines

# -------------Set parameters-----------
ww_adapt_num = 3 # number of wakeword adaptation utterances in enrollment data
stc_adapt_num = 3 # number of sentence adaptation utterances in enrollment data
more_test_data = True # decides length of test data (0: wakeword+command, 1: wakeword+sentence)

if not more_test_data:
    result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
    fig_dir = 'fig'
else:
    result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
    fig_dir = 'fig/more_test_data'

log_plot = False # if True plot with logarithmic y-axis

# Set to True if plotting recognition rate vs. enrollment data:
plot_rr = True
# Set to True if plotting recognition rate of known/unknown speakers:
plot_known_unknown = False
# Set to True if plotting DET-curve
plot_DET = False

# -------------Load results-------------
print('Loading results:')
raw_rr, raw_far, raw_frr, raw_ear, raw_xaxis = [],[],[],[],[]
```

```
raw_rr_z, raw_far_z, raw_frr_z, raw_ear_z, raw_xaxis_z = [],[],[],[],[]
raw_rr_t, raw_far_t, raw_frr_t, raw_ear_t, raw_xaxis_t = [],[],[],[],[]
raw_rr_zt, raw_far_zt, raw_frr_zt, raw_ear_zt, raw_xaxis_zt =
    [],[],[],[],[]
raw_rr_tz, raw_far_tz, raw_frr_tz, raw_ear_tz, raw_xaxis_tz =
    [],[],[],[],[]

for stc_adapt_num in [3,6,9]:
    if not more_test_data:
        result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(
    stc_adapt_num)
    else:
        result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_
    '+str(stc_adapt_num)

    print('stc_adapt_num = '+str(stc_adapt_num)+' :')
    # 1: Raw scores
    raw_rr.append(numpy.load(os.path.join(result_dir,'raw','rr.npy')))
    raw_far.append(numpy.load(os.path.join(result_dir,'raw','far.npy')))
    raw_frr.append(numpy.load(os.path.join(result_dir,'raw','frr.npy')))
    raw_ear.append(numpy.load(os.path.join(result_dir,'raw','ear.npy')))
    raw_xaxis.append(numpy.load(os.path.join(result_dir,'raw','xaxis.npy')
    ))
    print('- raw results loaded')

    # 2: z-normalized scores
    raw_rr_z.append(numpy.load(os.path.join(result_dir,'z','rr.npy')))
    raw_far_z.append(numpy.load(os.path.join(result_dir,'z','far.npy')))
    raw_frr_z.append(numpy.load(os.path.join(result_dir,'z','frr.npy')))
    raw_ear_z.append(numpy.load(os.path.join(result_dir,'z','ear.npy')))
    raw_xaxis_z.append(numpy.load(os.path.join(result_dir,'z','xaxis.npy')
    ))
    print('- z-normalized results loaded')

    # 3: t-normalized scores
    raw_rr_t.append(numpy.load(os.path.join(result_dir,'t','rr.npy')))
    raw_far_t.append(numpy.load(os.path.join(result_dir,'t','far.npy')))
    raw_frr_t.append(numpy.load(os.path.join(result_dir,'t','frr.npy')))
    raw_ear_t.append(numpy.load(os.path.join(result_dir,'t','ear.npy')))
    raw_xaxis_t.append(numpy.load(os.path.join(result_dir,'t','xaxis.npy')
    ))
    print('- t-normalized results loaded')

    # 4: zt-normalized scores
    raw_rr_zt.append(numpy.load(os.path.join(result_dir,'zt','rr.npy')))
    raw_far_zt.append(numpy.load(os.path.join(result_dir,'zt','far.npy')))
    raw_frr_zt.append(numpy.load(os.path.join(result_dir,'zt','frr.npy')))
    raw_ear_zt.append(numpy.load(os.path.join(result_dir,'zt','ear.npy')))
    raw_xaxis_zt.append(numpy.load(os.path.join(result_dir,'zt','xaxis.npy
    ')))
    print('- zt-normalized results loaded')

    # 5: tz-normalized scores
    raw_rr_tz.append(numpy.load(os.path.join(result_dir,'tz','rr.npy')))
    raw_far_tz.append(numpy.load(os.path.join(result_dir,'tz','far.npy')))
    raw_frr_tz.append(numpy.load(os.path.join(result_dir,'tz','frr.npy')))
    raw_ear_tz.append(numpy.load(os.path.join(result_dir,'tz','ear.npy')))
```

```
    raw_xaxis_tz.append(numpy.load(os.path.join(result_dir,'tz','xaxis.npy
    ')))
    print('- tz-normalized results loaded')


# ----------------Plot-----------------
# Plot max recognition rate vs number of enrollment sentences
if plot_rr:
    print('Plot max recognition rate vs number of enrollment sentences')
    plot_all = False
    fig = plt.figure()
    if plot_all:
        # Plots subplots for all systems, including plots of the error
    rates
        # NOTE: NEEDS FIX TO WORK
        for i in range(5): # iterate through all normalization types
            rr = [raw_rr, raw_rr_z, raw_rr_t, raw_rr_zt, raw_rr_tz][i]
            far = [raw_far, raw_far_z, raw_far_t, raw_far_zt, raw_far_tz][
    i]
            frr = [raw_frr, raw_frr_z, raw_frr_t, raw_frr_zt, raw_frr_tz][
    i]
            ear = [raw_ear, raw_ear_z, raw_ear_t, raw_ear_zt, raw_ear_tz][
    i]
            xaxis = [raw_xaxis, raw_xaxis_z, raw_xaxis_t, raw_xaxis_zt,
    raw_xaxis_tz][i]
            x_idx = [3,6,9] # indexes on x-axis (effectively, the number
    of adaptation sentences)
            plot_title = ['Raw results', 'z-normalized results','t-
    normalized results',\
                          'zt-normalized results','tz-normalized
    results'][i]

            # Find index of maximum recognition rate and store values of
    all result
            # arrays on these indices
            max_rr, max_far, max_frr, max_ear = [],[],[],[]
            for j in range(3): # iterate through all numbers of enrollment
     sentences
                max_rr_idx = int(xaxis[j][int(rr[j].argmax())])
                max_rr.append(rr[j].max())
                max_far.append(far[j][max_rr_idx])
                max_frr.append(frr[j][max_rr_idx])
                max_ear.append(ear[j][max_rr_idx])

            # Plotting starts here
            ax = fig.add_subplot(5,1,i+1)
            rr_plt, = ax.plot(x_idx,max_rr,'r')
            far_plt, = ax.plot(x_idx,max_far,'g')
            frr_plt, = ax.plot(x_idx,max_frr,'b')
            ear_plt = ax.plot(x_idx,max_ear,'m')

            ax.set_xlabel('Adaptation data')
            ax.set_title(plot_title)
            if log_plot:
                ax.set_yscale('log')
                ax.set_title(plot_title+' - log')

```

```
131             red_line = mlines.Line2D([],[],color='red',label='Recognition
    rate')
132             green_line = mlines.Line2D([],[],color='green',label='False
    acceptance rate')
133             blue_line = mlines.Line2D([],[],color='blue',label='False
    rejection rate')
134             mag_line = mlines.Line2D([],[],color='magenta',label='
    Erroneous acceptance rate')
135             ax.legend(loc='bottom center', handles=[red_line,green_line,
    blue_line,mag_line])
136 
137             if log_plot:
138                 fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_
    '+str(ww_adapt_num)+'_stc_3_6_9_log.png'))
139             else:
140                 fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_
    '+str(ww_adapt_num)+'_stc_3_6_9.png'))
141             print('- saved figure')
142 
143     else:
144         # Plot only recognition rate, not error rates
145         for i in range(5):
146             rr = [raw_rr, raw_rr_z, raw_rr_t, raw_rr_zt, raw_rr_tz][i]
147             xaxis = [raw_xaxis, raw_xaxis_z, raw_xaxis_t, raw_xaxis_zt,
    raw_xaxis_tz][i]
148             x_idx = [3,6,9] # indexes on x-axis (effectively, the number
    of adaptation sentences)
149             plot_title = ['Raw results', 'z-normalized results','t-
    normalized results',
150                           'zt-normalized results','tz-normalized results'
    ][i]
151             color = ['r','g','b','m','k'][i]
152             marker = ['o','s','^','v','*'][i]
153 
154             # Find index of maximum recognition rate and store values of
    all result
155             # arrays on these indices
156             max_rr, max_far, max_frr, max_ear = [],[],[],[]
157             for j in range(3): # iterate through all numbers of enrollment
     sentences
158                 max_rr.append(rr[j].max()*100)
159 
160             # Plotting starts here
161             ax = fig.add_subplot(1,1,1)
162             rr_plt, = ax.plot(x_idx,max_rr,color,marker=marker)
163 
164             ax.set_xlabel('Enrollment data [num. sentences]')
165             ax.set_ylabel('Recognition rate [%]')
166 
167         if log_plot:
168             ax.set_yscale('log')
169             ax.set_title(plot_title+' - log')
170 
171         if not more_test_data:
172             ax.set_title('GMM-UBM recognition rate vs. enrollment data')
173         else:
174             ax.set_title('GMM-UBM recognition rate vs. enrollment data,
```

```
    more test data')
        ax.set_ylim([30,85])
        ax.grid()

        red_line = mlines.Line2D([],[],color='red',label='Raw',marker='o')
        green_line = mlines.Line2D([],[],color='green',label='z-norm',
    marker='s')
        blue_line = mlines.Line2D([],[],color='blue',label='t-norm',marker
    ='^')
        mag_line = mlines.Line2D([],[],color='magenta',label='zt-norm',
    marker='v')
        yellow_line = mlines.Line2D([],[],color='black',label='tz-norm',
    marker='*')

        # Shrink current axis's height by 10% on the bottom
        box = ax.get_position()
        ax.set_position([box.x0, box.y0 + box.height * 0.1,
                         box.width, box.height * 0.9])

        # Put a legend below current axis
        ax.legend(loc='upper center',
                  handles=[red_line,green_line,blue_line,mag_line,
    yellow_line],
                  bbox_to_anchor=(0.5, -0.15),
                  fancybox=True,
                  shadow=True,
                  ncol=5)

    if log_plot:
        fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_'+str(
    ww_adapt_num)+'_stc_3_6_9_log.png'))#,
                    #bbox_inches='tight')
    else:
        fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_'+str(
    ww_adapt_num)+'_stc_3_6_9.png'))#,
                    #bbox_inches='tight')
    print('- saved figure')


# Plot rate of recognized known and unknown speakers for each system
if plot_known_unknown:
    print('Plot known/unknown speaker recognition rates:')
    # Modified from https://matplotlib.org/examples/api/barchart_demo.html
    # Input data from naive scoring and threshold scoring
    # These numbers are obtained from 'log/evaluate_gmm_ubm_ww_3_stc_3.txt
    '
    # Indices corresponds to ['raw','z-norm','t-norm','zt-norm','tz-norm']
    if not more_test_data: # Numbers are obtained from 'log/
    evaluate_gmm_ubm_ww_3_stc_3.txt'
        kn_naive = [76.13, 73.25, 76.13, 73.25, 90.75]
        un_naive = [75.0, 89.5, 75.0, 89.5, 16.5]
        kn_thres = [68.88, 65.13, 68.38, 65.13, 82.13]
        un_thres = [89.5, 93.5, 90.5, 93.5, 63.5]
    else: # numbers are obtained from 'log/
    evaluate_gmm_ubm_ww_3_stc_3_more_data.txt'
        kn_naive = [65.13, 61.75, 65.13, 61.75, 90.63]
        un_naive = [91.5, 97.0, 91.5, 97.0, 29.5]
```

```
        kn_thres = [58.88, 55.38, 58.75, 56.25, 82.5]
        un_thres = [95.5, 97.5, 95.5, 97.5, 73.0]

    # 1: NAIVE DECISION
    x_idx = numpy.arange(5)
    y_idx = numpy.arange(0,101,10)
    width = 0.25

    fig, ax = plt.subplots()
    rects_kn = ax.bar(x_idx, kn_naive, width, color='b')
    rects_un = ax.bar(x_idx+width, un_naive, width, color='g')

    ax.set_ylabel('Relative recognition rate [%]')
    if not more_test_data:
        ax.set_title('GMM-UBM: Known/unknown speaker trials, naive
    decision',fontsize=10)
    else:
        ax.set_title('GMM-UBM: Known/unknown speaker trials, naive
    decision, more test data',fontsize=10)
    ax.set_xticks(x_idx+width/2)
    ax.set_xticklabels(('raw','z-norm','t-norm','zt-norm','tz-norm'))
    ax.set_yticks(y_idx)
    ax.set_ylim([0,100])


    # Shrink current axis's height by 10% on the bottom
    box = ax.get_position()
    ax.set_position([box.x0, box.y0 + box.height * 0.1,
                     box.width, box.height * 0.9])

    # Put a legend below current axis
    ax.legend((rects_kn[0],rects_un[0]), ('Known','Unknown'),
              bbox_to_anchor=(0.75, -0.06),
              fancybox=True,
              shadow=True,
              ncol=5)
    ax.grid()
    fig.savefig(os.path.join(fig_dir,'comparison','
    gmm_ubm_known_unknown_rr_naive.png'))
    print('- saved known/unknown recognition rate for naive decision')

    # 2: THRESHOLD DECISION
    x_idx = numpy.arange(5)
    y_idx = numpy.arange(0,101,10)
    width = 0.25

    fig, ax = plt.subplots()
    rects_kn = ax.bar(x_idx, kn_thres, width, color='b')
    rects_un = ax.bar(x_idx+width, un_thres, width, color='g')

    ax.set_ylabel('Relative recognition rate [%]')
    if not more_test_data:
        ax.set_title('GMM-UBM: Known/unknown speaker trials, threshold
    decision',fontsize=10)
    else:
        ax.set_title('GMM-UBM: Known/unknown speaker trials, threshold
    decision, more test data',fontsize=10)
```

```
    ax.set_xticks(x_idx+width/2)
    ax.set_xticklabels(('raw','z-norm','t-norm','zt-norm','tz-norm'))
    ax.set_yticks(y_idx)
    ax.set_ylim([0,100])

    # Shrink current axis's height by 10% on the bottom
    box = ax.get_position()
    ax.set_position([box.x0, box.y0 + box.height * 0.1,
                     box.width, box.height * 0.9])

    # Put a legend below current axis
    ax.legend((rects_kn[0],rects_un[0]), ('Known','Unknown'),
              bbox_to_anchor=(0.75, -0.06),
              fancybox=True,
              shadow=True,
              ncol=5)
    ax.grid()

    fig.savefig(os.path.join(fig_dir,'comparison','
    gmm_ubm_known_unknown_rr_threshold.png'))
    print('- saved known/unknown recognition rate for threshold decision')


# Plot DET-curve for all systems
# This is an evaluation of the systems as speaker verification systems,
# thus we define FAR as FAR+EAR in this plot
if plot_DET:
    print('Note: Recommended to use ../tools/detplot_many.py')
    print('Plot DET-curve')
    # Determine enrollment data size
    if stc_adapt_num == 3:
        adapt_idx = 0
    elif stc_adapt_num == 6:
        adapt_idx = 1
    else:
        adapt_idx = 2

    # Compute new FAR
    raw_far_sum = raw_far[adapt_idx]+raw_ear[adapt_idx]
    raw_far_sum_z = raw_far_z[adapt_idx]+raw_ear_z[adapt_idx]
    raw_far_sum_t = raw_far_t[adapt_idx]+raw_ear_t[adapt_idx]
    raw_far_sum_zt = raw_far_zt[adapt_idx]+raw_ear_zt[adapt_idx]
    raw_far_sum_tz = raw_far_tz[adapt_idx]+raw_ear_tz[adapt_idx]

    # Plot curves
    fig = plt.figure()
    ax = fig.add_subplot(111)
    #ax.set_xscale('log')
    #ax.set_yscale('log')
    ax.plot(numpy.multiply(raw_frr[adapt_idx],100), numpy.multiply(
    raw_far_sum,100))
    ax.plot(numpy.multiply(raw_frr_z[adapt_idx],100), numpy.multiply(
    raw_far_sum_z,100))
    ax.plot(numpy.multiply(raw_frr_t[adapt_idx],100), numpy.multiply(
    raw_far_sum_t,100))
    ax.plot(numpy.multiply(raw_frr_zt[adapt_idx],100), numpy.multiply(
    raw_far_sum_zt,100))
```

```
    ax.plot(numpy.multiply(raw_frr_tz[adapt_idx],100), numpy.multiply(
    raw_far_sum_tz,100))

    #yticks = numpy.array([0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 40, 80])
    #xticks = numpy.array([10, 20, 40, 60, 80, 100])
    #ax.set_yticks(yticks)
    #ax.set_yticklabels(numpy.array(['0.1', '0.2', '0.5', '1', '2', '5',
    '10', '20', '40', '80']), size='x-small')
    #ax.set_xticks(xticks)
    #ax.set_xticklabels(numpy.array(['10', '20', '40','60','80','100']),
    size='x-small')

    plt.title('GMM-UBM DET-curves')
    plt.grid(True)
    plt.xlabel('False rejection rate [%]')
    plt.ylabel('False acceptance rate [%]')

    fig.savefig(os.path.join(fig_dir,'sv','gmm_ubm_compared_ww_3_stc_'+str
    (stc_adapt_num)+'.png'))
    print('Saved DET-plot')
```

I-vector system

The following scripts define and run the I-vector systems.

Running and writing scores: I-vectors/run_gmm_ubm.sh

```
# Runs the i-vector-based speaker identification experiment
# Serves as a baseline together with the gmm-ubm system
# Scoring is done using
#   1: Cosine-distance scoring
#   2: Mahalanobis' matrix
#   3: Two-covariance scoring
#   4: PLDA

import sidekit
import multiprocessing
import time
import logging
import sys
import os
import copy


if len(sys.argv) != 5:
    print('Usage: '+sys.argv[0]+\
              ' <use more test data {0,1}>'+\
              ' <scoring method {1:cosine-distance,2:Mahalanobis,3:Two-
    covariance,4:PLDA}>'+\
              ' <number of enrollment wakewords>'+\
              ' <number of enrollment sentences>')
    exit()

logging.basicConfig(filename='log/ivec.log',level=logging.DEBUG)

```

```
print(time.strftime('%c'))
print('I-VECTOR-BASED SPEAKER IDENTIFICATION')

# Set parameters and filepaths
gender = 'gidp' # 'gdp' 'gidp'
distrib_nb = 1024  # number of Gaussian distributions for each GMM
feature_size = 57 # size of feature vector (mfcc+delta+deltadelta)
base_dir = '/home/studenter/jorgeja/Projects/master/'
feat_dir = base_dir+'data/feats/'
ubm_dir = base_dir+'ubm/gmm/gender_id_diag_ubm.h5'
selection_size = 1000 # number of utterances in ubm training set

ww_adapt_num = int(sys.argv[3]) # number of wakewords in enrollment data
stc_adapt_num = int(sys.argv[4]) # number of sentences in enrollment data
more_test_data = bool(int(sys.argv[1])) # decides length of test data(0:
    wakeword+command, 1: wakeword+sentence)

rank_TV = 400  # Rank of the total variability matrix
tv_iteration = 10  # number of iterations to run
plda_rk = 400  # rank of the PLDA eigenvoice matrix
nbThread = max(multiprocessing.cpu_count()-1, 1)  # Number of parallel
    process to run

scoring_method = int(sys.argv[2]) # 1: cos, 2: Mah, 3: Two-cov, 4: PLDA (
    see top)
old_method = False # if False, use StatServer.factor_analysis(), else use
    FactorAnalyser()
do_TV_training = False # if False, load existing TV matrix
TV_path = 'data/TV__1024_it-10.h5' # path of TV matrix
do_PLDA_training = False  # if False, load existing PLDA matrix
PLDA_path = 'data/PLDA_model_gidp_100.h5'

if do_TV_training:
    # Get iteration number
    assert len(sys.argv) == 2, 'Iteration number required for TV matrix
    training'
    it_num_tv = int(sys.argv[1])
    print('ITERATION NUMBER '+str(it_num_tv))

# Load task definition
# Enrollment data: 3 wakewords, 3 sentences:
if stc_adapt_num == 3:
    enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/enroll_'+gender+'
    _50_t_norm_50.h5')
# Enrollment data: 3 wakewords, 6 or 9 sentences:
else:
    enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/more_data/enroll_'
    +gender+'_50_t_norm_50_ww_'+str(ww_adapt_num)\
                                    +'_stc_'+str(stc_adapt_num)+'.h5')
if not more_test_data:
    test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/eval_'+gender+'
    _10_spk_100_sets_ndx.h5')
    keys = sidekit.Key(base_dir+'task/gmm_ubm/eval_'+gender+'
    _10_spk_100_sets_key.h5')
    test_idmap = sidekit.IdMap(base_dir+'task/ivec/eval_'+gender+'
    _10_spk_100_sets_idmap.h5')
else:
```

```
    test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/more_data/eval_'+gender+'_10_spk_100_sets_more_data_ndx.h5')
    keys = sidekit.Key(base_dir+'task/gmm_ubm/more_data/eval_'+gender+'_10_spk_100_sets_more_data_key.h5')
    test_idmap = sidekit.IdMap(base_dir+'task/ivec/eval_'+gender+'_10_spk_100_sets_more_data_idmap.h5')

tv_idmap = sidekit.IdMap(base_dir+"task/ivec/TV_idmap_"+gender+".h5")
plda_idmap = sidekit.IdMap(base_dir+"task/ivec/PLDA_idmap_"+gender+".h5")


# Create a FeaturesServer to load features and feed the other methods
fs_ldc = sidekit.FeaturesServer(features_extractor=None,
                                feature_filename_structure=feat_dir+"LDC/new/{}.h5",
                                sources=None,
                                dataset_list=["cep", "vad"],
                                mask=None,
                                feat_norm="cmvn",
                                global_cmvn=None,
                                dct_pca=False,
                                dct_pca_config=None,
                                sdc=False,
                                sdc_config=None,
                                delta=True,
                                double_delta=True,
                                delta_filter=None,
                                context=None,
                                traps_dct_nb=None,
                                rasta=True,
                                keep_all_features=False)

fs_enroll_test = sidekit.FeaturesServer(features_extractor=None,
                                        feature_filename_structure=feat_dir+"RSR2015/{}.h5",
                                        sources=None,
                                        dataset_list=["cep", "vad"],
                                        mask=None,
                                        feat_norm="cmvn",
                                        global_cmvn=None,
                                        dct_pca=False,
                                        dct_pca_config=None,
                                        sdc=False,
                                        sdc_config=None,
                                        delta=True,
                                        double_delta=True,
                                        delta_filter=None,
                                        context=None,
                                        traps_dct_nb=None,
                                        rasta=True,
                                        keep_all_features=False)


# Load ubm
print('Load the UBM')
ubm = sidekit.Mixture(ubm_dir)
```

```
126 assert ubm.get_distrib_nb() == distrib_nb, 'Number of mixtures in loaded
        UBM does not match parameter distrib_nb'
127
128 """
129 # Compute sufficient statistics
130 print('Compute the sufficient statistics')
131
132 enroll_stat = sidekit.StatServer(enroll_idmap, distrib_nb, feature_size)
133 enroll_stat.accumulate_stat(ubm=ubm, feature_server=fs_enroll_test,
        seg_indices=range(enroll_stat.segset.shape[0]), num_thread=nbThread)
134 enroll_stat.write('data/stat_ivec_enroll_'+gender+'_100_ww_'+str(
        ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
135 print('- enroll_stat done')
136
137 test_stat = sidekit.StatServer(test_idmap, distrib_nb, feature_size)
138 test_stat.accumulate_stat(ubm=ubm, feature_server=fs_enroll_test,
        seg_indices=range(test_stat.segset.shape[0]), num_thread=nbThread)
139 if not more_test_data:
140     test_stat.write('data/stat_ivec_test_'+gender+'_100.h5')
141 else:
142     test_stat.write('data/stat_ivec_test_'+gender+'_100_more_data.h5')
143 print('- test_stat done')
144
145 back_idmap = plda_idmap.merge(tv_idmap)
146 back_stat = sidekit.StatServer(back_idmap, distrib_nb, feature_size)
147 back_stat.accumulate_stat(ubm=ubm, feature_server=fs_ldc, seg_indices=
        range(back_stat.segset.shape[0]), num_thread=nbThread)
148 back_stat.write('data/stat_back_'+gender+'_100.h5')
149 print('- back_stat done')
150 """
151
152 print('Load sufficient statistics:')
153 enroll_stat = sidekit.StatServer('data/stat_ivec_enroll_'+gender+'_100_ww_
        '+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5', distrib_nb,
        feature_size)
154 print('- enroll_stat')
155 if not more_test_data:
156     test_stat = sidekit.StatServer('data/stat_ivec_test_'+gender+'_100.h5'
        , distrib_nb, feature_size)
157 else:
158     test_stat = sidekit.StatServer('data/stat_ivec_test_'+gender+'
        _100_more_data.h5', distrib_nb, feature_size)
159 print('- test_stat')
160
161 if do_TV_training:
162     if old_method: # Use StatServer.factor_analysis
163         print('Train TV matrix using StatServer.factor_analysis()')
164         sys.stdout.flush() # print output
165         tv_stat = sidekit.StatServer.read_subset('data/stat_back_'+gender+
        '_100.h5', tv_idmap)
166         tv_mean, tv, _, __, tv_sigma = tv_stat.factor_analysis(rank_f =
        rank_TV,
167                                                                rank_g = 0,
         # if 0, this is an i-vec extractor
168                                                                rank_h =
        None,
```

```
169
    re_estimate_residual = False,
170                                                                 it_nb = (
    tv_iteration,0,0),
171                                                                 min_div =
    True,
172                                                                 ubm = ubm,
173                                                                 batch_size
    = 100,
174                                                                 num_thread
    = nbThread,
175
    save_partial = "data/TV_{}".format(distrib_nb))
176         sidekit.sidekit_io.write_tv_hdf5((tv, tv_mean, tv_sigma), "data/
    TV_{}".format(distrib_nb))
177         print('- done')
178         sys.stdout.flush() # print output
179         exit()
180
181     else: # Use FactorAnalyser
182         print('Train TV matrix using FactorAnalyser.total_variability()')
183         sys.stdout.flush() # print output
184
185         """ # You only need to do this once
186         tv_stat = sidekit.StatServer.read_subset('data/stat_back_'+gender
    +'_100.h5', tv_idmap)
187         tv_stat.write('data/stat_tv_'+gender+'_100.h5')
188         print('- wrote TV_stat')
189         sys.stdout.flush()
190         """
191         for i in [it_num_tv]:
192
193             fa_load_TV = sidekit.FactorAnalyser('data/TV__1024_it-'+str(i
    -1)+'.h5')
194             print('- loaded fa_load_TV')
195             sys.stdout.flush()
196
197             fa = sidekit.FactorAnalyser()
198             fa.total_variability('data/stat_tv_'+gender+'_100.h5', #
    stat_server_filename must be a string, not an object
199                                  ubm,
200                                  rank_TV,
201                                  nb_iter=1, #tv_iteration,
202                                  min_div=True,
203                                  tv_init=fa_load_TV.F,
204                                  batch_size=100,
205                                  save_init=False,
206                                  output_file_name='data/TV_{}'.format(
    distrib_nb),
207                                  num_thread=8) # Sidekit documentation
    recommends a low number of processes for TV training
208             tv_mean = fa.mean
209             tv = fa.F
210             tv_sigma = fa.Sigma
211             os.rename('data/TV_1024.h5','data/TV__1024_it-'+str(i)+'.h5')
212
213             print('  iteration number '+str(i)+' done')
```

```
            sys.stdout.flush() # print output
            exit()

else:
    print('Load TV matrix')
    sys.stdout.flush() # print output
    fa = sidekit.FactorAnalyser(TV_path)
    tv_mean = fa.mean
    tv = fa.F
    tv_sigma = fa.Sigma


# Extract i-vectors
if old_method: # Extract i-vectors with StatServer.estimate_hidden()
    print('Extract i-vectors using StatServer.estimate_hidden()')
    enroll_iv = enroll_stat.estimate_hidden(tv_mean, tv_sigma, V=tv,
    batch_size=100, num_thread=nbThread)[0]
    enroll_iv.write('data/iv_enroll_'+gender+'_100.h5')
    print('- enroll_iv done')

    sys.stdout.flush()

    test_iv = test_stat.estimate_hidden(tv_mean, tv_sigma, V=tv,
    batch_size=100, num_thread=nbThread)[0]
    test_iv.write('data/iv_test_'+gender+'_100.h5')
    print('- test_iv done')

    sys.stdout.flush()

    plda_stat = sidekit.StatServer.read_subset('data/stat_back_'+gender+'
    _100.h5', plda_idmap)
    plda_iv = plda_stat.estimate_hidden(tv_mean, tv_sigma, V=tv,
    batch_size=100, num_thread=nbThread)[0]
    plda_iv.write('data/iv_plda_'+gender+'_100.h5')
    print('- plda_iv done')

    sys.stdout.flush()

else: # Extract i-vectors with FactorAnalyser.extract_ivectors()
    print('Extract i-vectors using FactorAnalyser.extract_ivectors()')
    """
    enroll_iv = fa.extract_ivectors(ubm,
                                    'data/stat_ivec_enroll_'+gender+'
    _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5',
                                    prefix='',
                                    batch_size=100,
                                    uncertainty=False,
                                    num_thread=nbThread)
    enroll_iv.write('data/iv_enroll_'+gender+'_100_ww_'+str(ww_adapt_num)
    +'_stc_'+str(stc_adapt_num)+'.h5')
    print('- enroll_iv done')

    if not more_test_data:
        test_iv = fa.extract_ivectors(ubm,
                                      'data/stat_ivec_test_'+gender+'_100.
    h5',
                                      prefix='',
```

```
                                    batch_size=100,
                                    uncertainty=False,
                                    num_thread=nbThread)
        test_iv.write('data/iv_test_'+gender+'_100.h5')
    else:
        test_iv = fa.extract_ivectors(ubm,
                                    'data/stat_ivec_test_'+gender+'
    _100_more_data.h5',
                                    prefix='',
                                    batch_size=100,
                                    uncertainty=False,
                                    num_thread=nbThread)
        test_iv.write('data/iv_test_'+gender+'_100_more_data.h5')
    print('- test_iv done')

    if not os.path.exists('data/stat_ivec_plda_'+gender+'_100.h5'):
        plda_stat = sidekit.StatServer.read_subset('data/stat_TV_'+gender
    +'_100.h5', plda_idmap) # TV stat file contains all PLDA stats
        plda_stat.write('data/stat_ivec_plda_'+gender+'_100.h5')
        print('- plda_stat done')

    plda_iv = fa.extract_ivectors(ubm,
                                'data/stat_ivec_plda_'+gender+'_100.h5',
                                prefix='',
                                batch_size=100,
                                uncertainty=False,
                                num_thread=nbThread)
    plda_iv.write('data/iv_plda_'+gender+'_100.h5')
    print('- plda_iv done')
    """

# ---------------------------------RUN THE TESTS
    -------------------------------------
print('Loading i-vectors')
enroll_iv = sidekit.StatServer('data/iv_enroll_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
if not more_test_data:
    test_iv = sidekit.StatServer('data/iv_test_'+gender+'_100.h5')
else:
    test_iv = sidekit.StatServer('data/iv_test_'+gender+'_100_more_data.h5
    ')
plda_iv = sidekit.StatServer('data/iv_plda_'+gender+'_100.h5')


# 1: Using Cosine similarity
if scoring_method == 1:
    print('Scoring with Cosine similarity: ')
    scores_cos = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv,
    test_ndx, wccn=None)
    if not more_test_data:
        scores_cos.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos.h5')
    else:
        scores_cos.write('scores/more_test_data/scores_ivec_'+gender+'
    _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos.h5')
    print('- cos done')
    sys.stdout.flush()
```

```
313
314     wccn = plda_iv.get_wccn_choleski_stat1()
315     scores_cos_wcnn = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv
        , test_ndx, wccn=wccn)
316     if not more_test_data:
317         scores_cos_wcnn.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
        ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos_wccn.h5')
318     else:
319         scores_cos_wcnn.write('scores/more_test_data/scores_ivec_'+gender+
        '_100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos_wccn.h5'
        )
320     print('- cos wccn done')
321     sys.stdout.flush()
322
323     LDA = plda_iv.get_lda_matrix_stat1(150)
324     plda_iv_lda = copy.deepcopy(plda_iv)
325     enroll_iv_lda = copy.deepcopy(enroll_iv)
326     test_iv_lda = copy.deepcopy(test_iv)
327     plda_iv_lda.rotate_stat1(LDA)
328     enroll_iv_lda.rotate_stat1(LDA)
329     test_iv_lda.rotate_stat1(LDA)
330     scores_cos_lda = sidekit.iv_scoring.cosine_scoring(enroll_iv_lda,
        test_iv_lda, test_ndx, wccn=None)
331     if not more_test_data:
332         scores_cos_lda.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
        ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos_lda.h5')
333     else:
334         scores_cos_lda.write('scores/more_test_data/scores_ivec_'+gender+'
        _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos_lda.h5')
335     print('- cos lda done')
336     sys.stdout.flush()
337
338     wccn = plda_iv_lda.get_wccn_choleski_stat1()
339     scores_cos_lda_wcnn = sidekit.iv_scoring.cosine_scoring(enroll_iv_lda,
         test_iv_lda, test_ndx, wccn=wccn)
340     if not more_test_data:
341         scores_cos_lda_wcnn.write('scores/scores_ivec_'+gender+'_100_ww_'+
        str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos_lda_wccn.h5')
342     else:
343          scores_cos_lda_wcnn.write('scores/more_test_data/scores_ivec_'+
        gender+'_100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'
        _cos_lda_wccn.h5')
344     print('- cos lda wccn done')
345     print('Success!')
346
347 # 2: Using Mahalanobis distance
348 elif scoring_method == 2:
349     raise Exception('Scoring with Mahalanobis distance not implemented yet
        .')
350     meanEFR, CovEFR = plda_iv.estimate_spectral_norm_stat1(3)
351
352     plda_iv_efr1 = copy.deepcopy(plda_iv)
353     enroll_iv_efr1 = copy.deepcopy(enroll_iv)
354     test_iv_efr1 = copy.deepcopy(test_iv)
355
356     plda_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
357     enroll_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
```

```
    test_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
    M1 = plda_iv_efr1.get_mahalanobis_matrix_stat1()
    scores_mah_efr1 = sidekit.iv_scoring.mahalanobis_scoring(
    enroll_iv_efr1, test_iv_efr1, test_ndx, M1)


# 3: Using Two-covariance scoring
elif scoring_method == 3:
    raise Exception('Scoring with two-covariance not implemented yet')
    W = plda_iv.get_within_covariance_stat1()
    B = plda_iv.get_between_covariance_stat1()
    scores_2cov = sidekit.iv_scoring.two_covariance_scoring(enroll_iv,
    test_iv, test_ndx, W, B)

    meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')

    plda_iv_sn1 = copy.deepcopy(plda_iv)
    enroll_iv_sn1 = copy.deepcopy(enroll_iv)
    test_iv_sn1 = copy.deepcopy(test_iv)

    plda_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
    enroll_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
    test_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])

    W1 = plda_iv_sn1.get_within_covariance_stat1()
    B1 = plda_iv_sn1.get_between_covariance_stat1()
    scores_2cov_sn1 = sidekit.iv_scoring.two_covariance_scoring(
    enroll_iv_sn1, test_iv_sn1, test_ndx, W1, B1)


# 4: Using PLDA
elif scoring_method == 4:
    print('Scoring with PLDA:')

    # Spherical nuisance normalization
    meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')
    plda_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
    enroll_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
    test_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
    print('- spherical nuisance normalization done')

    if do_PLDA_training:
        # Train PLDA matrix
        if old_method: # Train PLDA matrix using StatServer.
    factor_analysis()
            print('Train PLDA matrix using StatServer.factor_analysis()
    ...')
            plda_mean, plda_F, plda_G, plda_H, plda_Sigma = plda_iv.
    factor_analysis(rank_f=plda_rk,

              rank_g=0,

              rank_h=None,

              re_estimate_residual=True,

              it_nb=(10,0,0),
```

```
405
                min_div=True,
406
                ubm=None,
407
                batch_size=100,
408
                num_thread=nbThread)
409             print('- PLDA training done')
410             sidekit.sidekit_io.write_plda_hdf5((plda_mean, plda_F, plda_G,
     plda_Sigma), "data/plda_model_"+gender+"_100.h5")
411             scores_plda = sidekit.iv_scoring.PLDA_scoring(enroll_iv,
    test_iv, test_ndx, plda_mean, plda_F, plda_G, plda_Sigma, full_model=
    False)
412             print('Scoring done')
413
414         else: # Train PLDA matrix using FactorAnalyser.plda()
415             print('Train PLDA matrix using FactorAnalyser.plda() ...')
416             sys.stdout.flush()
417             fa_plda = sidekit.FactorAnalyser()
418             fa_plda.plda(stat_server=plda_iv,
419                          rank_f=plda_rk,
420                          nb_iter=10,
421                          scaling_factor=1,
422                          output_file_name=PLDA_path.split('.')[0],
423                          save_partial=True
424                          )
425             print('- PLDA training done')
426     else:
427         print('- load PLDA matrix')
428         fa_plda = sidekit.FactorAnalyser(PLDA_path)
429         print('- perform PLDA scoring')
430         sys.stdout.flush()
431         scores_plda = sidekit.iv_scoring.fast_PLDA_scoring(enroll=
    enroll_iv,
432                                                            test=test_iv,
433                                                            ndx=test_ndx,
434                                                            mu=fa_plda.mean
    ,
435                                                            F=fa_plda.F,
436                                                            Sigma=fa_plda.
    Sigma)
437         print('- done')
438
439     if not more_test_data:
440         scores_plda.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
    ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_plda.h5')
441     else:
442         scores_plda.write('scores/more_test_data/scores_ivec_'+gender+'
    _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_plda.h5')
443
444
445
446 # ------------------
447 # Analyzation of scores will be done in a separate script
```

I-vector score evaluation: I-vectors/evaluate_ivectors.py

```
# This script evaluates scores from a i-vector speaker identification experiment
# z-/t-/zt-normalization is applied to scores before evaluation
# It is also possible to exclude the complementary model from the evaluation
#
# The system performance is measured and displayed in two ways:
# - Maximum recognition rate. The maximum recognition rate, for a given value
#   of the decision value
# - A plot of recognition rate vs. decision value
#
# Further:
# - score_file contains the actual scores against enrollment models, normalization
#   speaker models and the complementary model (w)
# - key_file contains a mapping between speaker models and test segments scored
#   on them, with boolean matrices indicating whether the trials are target/nontarget
#
# Written by Joergen Antonsen, April 26 2017

import sidekit
import numpy
import os
import sys
sys.path.append('/home/studenter/jorgeja/Projects/master/gmm_ubm')
import normalize_gmm_ubm
from normalize_gmm_ubm import get_mean_std
from normalize_gmm_ubm import naive_decision
from normalize_gmm_ubm import get_raw_scores
from normalize_gmm_ubm import estimate_z_norm_variables
from normalize_gmm_ubm import estimate_t_norm_variables
from normalize_gmm_ubm import z_normalization
from normalize_gmm_ubm import t_normalization
from normalize_gmm_ubm import moving_threshold_decision
from normalize_gmm_ubm import get_global_minmax
import matplotlib.pyplot as plt
import matplotlib.lines as mlines # for legend

assert len(sys.argv) == 6, 'Usage: '+sys.argv[0]+' <score_file_path>'+\
    +' <system suffix>'\
    +' <number of enrollment wakewords>'\
    +' <number of enrollment sentences>'\
    +' <use more test data {0,1}>\n'\
    +'Example: '+sys.argv[0]+' scores/ww_3_stc_3/
    scores_ivec_gidp_100_100_cos.h5 cos 3 3 1'
assert sys.argv[2] in ['cos','cos_lda','cos_wccn','cos_lda_wccn','plda'],\
    'Invalid system suffix. Must be one of the following: cos, cos_lda,
    cos_wccn, cos_lda_wccn, cos_plda'

print('I-VECTOR SYSTEM EVALUATION')
print('- score file: '+sys.argv[1])

```

```
# -----------------Set parameters------------------
test_set_size = 10 # number of speakers in the test set
num_unknown = 2 # number of unknown speakers in the test set
enroll_set_size = test_set_size-num_unknown # number of known speakers in
    test set
num_set_size = 100 # number of unique test sets

enroll_range = 50
norm_range = 100

ww_adapt_num = int(sys.argv[3]) # number of wakewords in enrollment data
stc_adapt_num = int(sys.argv[4]) # number of sentences in enrollment data

score_file = sys.argv[1]

plot_fig = True # saves recognition rate plots as png if True
sys_name = sys.argv[2] # name of system (for plot title)

save_results = True # if True, save recognition rates as numpy arrays
more_test_data = sys.argv[5]

if more_test_data == '0':
    result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num
    )
    fig_dir = 'fig/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
elif more_test_data == '1':
    result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+
    str(stc_adapt_num)
    fig_dir = 'fig/more_test_data/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(
    stc_adapt_num)
else:
    raise ValueError('Invalid value of parameter <use more test data
    {0,1}>: '+more_test_data)
if not os.path.exists(result_dir):
    os.makedirs(result_dir)
if not os.path.exists(fig_dir):
    os.makedirs(fig_dir)

# -----------Read scores and test files------------
scores = sidekit.bosaris.Scores(score_file)
if more_test_data == '0':
    key = sidekit.Key('../task/gmm_ubm/eval_gidp_10_spk_100_sets_key.h5')
elif more_test_data == '1':
    key = sidekit.Key('../task/gmm_ubm/more_data/
    eval_gidp_10_spk_100_sets_more_data_key.h5')

# -----Manage variables and sort segment indices----
modelset = scores.modelset # set of models (f001, f002, ..., m099, m100, w
    )
segset = scores.segset # set of segments
scoremask = scores.scoremask # score mask, True for element (i,j) if
    segment j is tested on model i
scoremat = scores.scoremat # score matrix, contain llr scores from all
    trials
target_mask = key.tar # matrix that is True for all target trials
nontarget_mask = key.non # matrix that is True for all nontarget trials

```

```
# Get segment indices
num_models, num_segments = scoremat.shape

enroll_seg_idx_f = 0 # segment index of female enrollment speakers
norm_seg_idx_f = 0 # segment index of female norm speakers (temp value)
enroll_seg_idx_m = 0 # segment index of male enrollment speakers (temp
    value)
norm_seg_idx_m = 0 # segment index of male norm speakers (temp value)
i = 0

while int(segset[i].split('/')[2][1:4]) <= enroll_range:
    i += 1
norm_seg_idx_f = i

while segset[i].split('/')[2][0] != 'm':
    i += 1
enroll_seg_idx_m = i

while int(segset[i].split('/')[2][1:4]) <= enroll_range:
    i += 1
norm_seg_idx_m = i

# NOTE: The above code will not work unless segset elements have the
    format
#       <dir1>/<dir2>/{m,f}{001:xxx}<rest of filename>
#       e.g. cat/female/f001_01_032-049


# ---------Calculate recognition rates-------------
# First get the 'raw', unprocessed scores
score_target_raw, score_nontarget_raw = get_raw_scores(scoremat,
    target_mask,nontarget_mask,enroll_range,norm_range)


# NAIVE DECISION - Choose the model that gives the highest score, without
    thresholding
print('NAIVE DECISION')
raw_recognition_rate,raw_far,raw_frr,raw_ear = naive_decision(
    norm_seg_idx_f,enroll_seg_idx_m,norm_seg_idx_m,score_target_raw,
    score_nontarget_raw)
print('Raw recognition rate: '+str(raw_recognition_rate*100)+'%')
print('False acceptance rate: '+str(raw_far*100)+'%')
print('False rejection rate: '+str(raw_frr*100)+'%')
print('Erroneous acceptance rate: '+str(raw_ear*100)+'%')
print('')


# DECISION WITH MOVING THRESHOLD - Choose model if score is solely above
    threshold
print('DECISION WITH MOVING THRESHOLD')
raw_rr,raw_far,raw_frr,raw_ear,raw_xaxis = moving_threshold_decision(
    norm_seg_idx_f,enroll_seg_idx_m,norm_seg_idx_m,score_target_raw,
    score_nontarget_raw,scoremat,num_points=500,additional_threshold=0.1)
print('Max raw recognition rate: '+str(raw_rr.max()*100)+'%')


# -----------------Save results--------------------
```

```
144 if save_results:
145     print('Saving results')
146     # 1: Raw scores
147     numpy.save(os.path.join(result_dir,sys_name,'raw','rr'), raw_rr)
148     numpy.save(os.path.join(result_dir,sys_name,'raw','far'), raw_far)
149     numpy.save(os.path.join(result_dir,sys_name,'raw','frr'), raw_frr)
150     numpy.save(os.path.join(result_dir,sys_name,'raw','ear'), raw_ear)
151     numpy.save(os.path.join(result_dir,sys_name,'raw','xaxis'), raw_xaxis)
152     print('- raw results saved')
153
154
155 # ------------Plot recognition rate----------------
156 # For moving threshold scoring
157 if more_test_data == '0':
158     sys_title = 'I-vector '+sys_name
159 else:
160     sys_title = 'I-vector '+sys_name+', more test data'
161
162 if plot_fig:
163     print('-------------Save plots-------------')
164     fig = plt.figure()
165     ax = fig.add_subplot(111)
166     rr, = ax.plot(raw_xaxis,raw_rr,'r')
167     far, = ax.plot(raw_xaxis,raw_far,'g')
168     frr, = ax.plot(raw_xaxis,raw_frr,'b')
169     ear = ax.plot(raw_xaxis,raw_ear,'m')
170     red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
171     green_line = mlines.Line2D([],[],color='green',label='False acceptance
          rate')
172     blue_line = mlines.Line2D([],[],color='blue',label='False rejection
          rate')
173     mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
          acceptance rate')
174     ax.legend(handles=[red_line,green_line,blue_line,mag_line])
175     ax.set_xlabel('Threshold')
176
177     # Annotate maxima/minima
178     rr_max_x, rr_max_y = get_global_minmax(raw_xaxis,raw_rr,'max')
179     ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
          'data')
180     far_max_x, far_max_y = get_global_minmax(raw_xaxis,raw_far,'max')
181     ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
          textcoords='data')
182     frr_max_x, frr_max_y = get_global_minmax(raw_xaxis,raw_frr,'min')
183     ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
          textcoords='data')
184     ear_max_x, ear_max_y = get_global_minmax(raw_xaxis,raw_ear,'max')
185     ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
          textcoords='data')
186     ax.set_title(sys_title)
187     fig.savefig(os.path.join(fig_dir,'rr_raw_'+score_file.split('.')[0].
          split('/')[2]+sys_name+'.png'))
188     ax.set_yscale('log')
189     ax.set_title(sys_title+' - log')
190     fig.savefig(os.path.join(fig_dir,'rr_raw_'+score_file.split('.')[0].
          split('/')[2]+sys_name+'_log.png'))
191     print('- saved scores')
```

```
192
193 print('Done!')
```

I-vector plotting: I-vectors/plots.py

```
1  # This script contains a few plot functions for presenting results
2  # For now, it makes a comparative plot between different normalization
       types
3
4  # Written by Joergen Antonsen May 16, 2017
5
6  import numpy
7  import os
8  import matplotlib.pyplot as plt
9  import matplotlib.lines as mlines
10
11 # -------------Set parameters-----------
12 ww_adapt_num = 3 # number of wakeword adaptation utterances in enrollment
       data
13 stc_adapt_num = 3 # number of sentence adaptation utterances in enrollment
        data
14 more_test_data = True # decides length of test data (0: wakeword+command,
       1: wakeword+sentence)
15
16 if not more_test_data:
17     result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num
       )
18     fig_dir = 'fig'
19 else:
20     result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+
       str(stc_adapt_num)
21     fig_dir = 'fig/more_test_data'
22
23 log_plot = False # if True plot with logarithmic y-axis
24
25 # Set to True if plotting recognition rate vs. enrollment data:
26 plot_rr = True
27 # Set to True if plotting recognition rate of known/unknown speakers:
28 plot_known_unknown = False
29
30 # -------------Load results-------------
31 print('Loading results:')
32 cos_rr, cos_far, cos_frr, cos_ear, cos_xaxis = [],[],[],[],[]
33 cos_lda_rr, cos_lda_far, cos_lda_frr, cos_lda_ear, cos_lda_xaxis =
       [],[],[],[],[]
34 cos_wccn_rr, cos_wccn_far, cos_wccn_frr, cos_wccn_ear, cos_wccn_xaxis =
       [],[],[],[],[]
35 cos_lda_wccn_rr, cos_lda_wccn_far, cos_lda_wccn_frr, cos_lda_wccn_ear,
       cos_lda_wccn_xaxis = [],[],[],[],[]
36 plda_rr, plda_far, plda_frr, plda_ear, plda_xaxis = [],[],[],[],[]
37
38 for stc_adapt_num in [3,6,9]:
39     if not more_test_data:
40         result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(
       stc_adapt_num)
41     else:
42         result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_
```

```
    '+str(stc_adapt_num)
43
44      print('stc_adapt_num = '+str(stc_adapt_num)+' :')
45      # 1: Cosine distance scores
46      cos_rr.append(numpy.load(os.path.join(result_dir,'cos','raw','rr.npy')
        ))
47      cos_far.append(numpy.load(os.path.join(result_dir,'cos','raw','far.npy
        ')))
48      cos_frr.append(numpy.load(os.path.join(result_dir,'cos','raw','frr.npy
        ')))
49      cos_ear.append(numpy.load(os.path.join(result_dir,'cos','raw','ear.npy
        ')))
50      cos_xaxis.append(numpy.load(os.path.join(result_dir,'cos','raw','xaxis
        .npy')))
51      print('- cosine distance results loaded')
52
53      # 2: Cosine distance + lda scores
54      cos_lda_rr.append(numpy.load(os.path.join(result_dir,'cos_lda','raw','
        rr.npy')))
55      cos_lda_far.append(numpy.load(os.path.join(result_dir,'cos_lda','raw',
        'far.npy')))
56      cos_lda_frr.append(numpy.load(os.path.join(result_dir,'cos_lda','raw',
        'frr.npy')))
57      cos_lda_ear.append(numpy.load(os.path.join(result_dir,'cos_lda','raw',
        'ear.npy')))
58      cos_lda_xaxis.append(numpy.load(os.path.join(result_dir,'cos_lda','raw
        ','xaxis.npy')))
59      print('- cosine distance + lda results loaded')
60
61      # 3: Cosine distance + wccn scores
62      cos_wccn_rr.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw'
        ,'rr.npy')))
63      cos_wccn_far.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw
        ','far.npy')))
64      cos_wccn_frr.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw
        ','frr.npy')))
65      cos_wccn_ear.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw
        ','ear.npy')))
66      cos_wccn_xaxis.append(numpy.load(os.path.join(result_dir,'cos_wccn','
        raw','xaxis.npy')))
67      print('- cosine distance + wccn results loaded')
68
69      # 4: Cosine distance + lda + wccn scores
70      cos_lda_wccn_rr.append(numpy.load(os.path.join(result_dir,'
        cos_lda_wccn','raw','rr.npy')))
71      cos_lda_wccn_far.append(numpy.load(os.path.join(result_dir,'
        cos_lda_wccn','raw','far.npy')))
72      cos_lda_wccn_frr.append(numpy.load(os.path.join(result_dir,'
        cos_lda_wccn','raw','frr.npy')))
73      cos_lda_wccn_ear.append(numpy.load(os.path.join(result_dir,'
        cos_lda_wccn','raw','ear.npy')))
74      cos_lda_wccn_xaxis.append(numpy.load(os.path.join(result_dir,'
        cos_lda_wccn','raw','xaxis.npy')))
75      print('- cosine distance + lda + wccn results loaded')
76
77      # 5: PLDA scores
78      plda_rr.append(numpy.load(os.path.join(result_dir,'plda','raw','rr.npy
```

```
    '')))
    plda_far.append(numpy.load(os.path.join(result_dir,'plda','raw','far.
    npy')))
    plda_frr.append(numpy.load(os.path.join(result_dir,'plda','raw','frr.
    npy')))
    plda_ear.append(numpy.load(os.path.join(result_dir,'plda','raw','ear.
    npy')))
    plda_xaxis.append(numpy.load(os.path.join(result_dir,'plda','raw','
    xaxis.npy')))
    print('- plda results loaded')


# ----------------Plot-----------------
# Plot max recognition rate vs number of enrollment sentences
plot_all = False
fig = plt.figure()
if plot_rr:
    if plot_all:
        # Plots subplots for all systems, including plots of the error
    rates
        # NOTE: NEEDS FIX TO WORK
        for i in range(5): # iterate through all normalization types
            rr = [cos_rr, cos_lda_rr, cos_wccn_rr, cos_lda_wccn_rr,
    plda_rr][i]
            far = [cos_far, cos_lda_far, cos_wccn_far, cos_lda_wccn_far,
    plda_far][i]
            frr = [cos_frr, cos_lda_frr, cos_wccn_frr, cos_lda_wccn_frr,
    plda_frr][i]
            ear = [cos_ear, cos_lda_ear, cos_wccn_ear, cos_lda_wccn_ear,
    plda_ear][i]
            xaxis = [cos_xaxis, cos_lda_xaxis, cos_wccn_xaxis,
    cos_lda_wccn_xaxis, plda_xaxis][i]
            x_idx = [3,6,9] # indexes on x-axis (effectively, the number
    of adaptation sentences)
            plot_title = ['Cosine distance results',
                          'cosine distance + lda results',
                          'cosine distance + wccn results',
                          'cosine distance + lda + wccn results',
                          'plda results'][i]

            # Find index of maximum recognition rate and store values of
    all result
            # arrays on these indices
            max_rr, max_far, max_frr, max_ear = [],[],[],[]
            for j in range(3): # iterate through all numbers of enrollment
     sentences
                max_rr_idx = int(xaxis[j][int(rr[j].argmax())])
                max_rr.append(rr[j].max())
                max_far.append(far[j][max_rr_idx])
                max_frr.append(frr[j][max_rr_idx])
                max_ear.append(ear[j][max_rr_idx])

            # Plotting starts here
            ax = fig.add_subplot(5,1,i+1)
            rr_plt, = ax.plot(x_idx,max_rr,'r')
            far_plt, = ax.plot(x_idx,max_far,'g')
            frr_plt, = ax.plot(x_idx,max_frr,'b')
```

```
            ear_plt = ax.plot(x_idx,max_ear,'m')

            ax.set_xlabel('Adaptation data')
            ax.set_title(plot_title)
            if log_plot:
                ax.set_yscale('log')
                ax.set_title(plot_title+' - log')

            red_line = mlines.Line2D([],[],color='red',label='Recognition
    rate')
            green_line = mlines.Line2D([],[],color='green',label='False
    acceptance rate')
            blue_line = mlines.Line2D([],[],color='blue',label='False
    rejection rate')
            mag_line = mlines.Line2D([],[],color='magenta',label='
    Erroneous acceptance rate')
            ax.legend(loc='bottom center', handles=[red_line,green_line,
    blue_line,mag_line])

            if log_plot:
                fig.savefig('rr_gidp_ww_'+str(ww_adapt_num)+'
    _stc_3_6_9_log.png')
            else:
                fig.savefig('rr_gidp_ww_'+str(ww_adapt_num)+'_stc_3_6_9.
    png')
            print('- saved figure')

    else:
        # Plot only recognition rate, not error rates
        for i in range(5):
            rr = [cos_rr, cos_lda_rr, cos_wccn_rr, cos_lda_wccn_rr,
    plda_rr][i]
            xaxis = [cos_xaxis, cos_lda_xaxis, cos_wccn_xaxis,
    cos_lda_wccn_xaxis, plda_xaxis][i]
            x_idx = [3,6,9] # indexes on x-axis (effectively, the number
    of adaptation sentences)
            plot_title = ['Cosine distance results',
                          'cosine distance + lda results',
                          'cosine distance + wccn results',
                          'cosine distance + lda + wccn results',
                          'plda results'][i]
            color = ['r','g','b','m','k'][i]
            marker = ['o','s','^','v','*'][i]

            # Find index of maximum recognition rate and store values of
    all result
            # arrays on these indices
            max_rr, max_far, max_frr, max_ear = [],[],[],[]
            for j in range(3): # iterate through all numbers of enrollment
     sentences
                max_rr.append(rr[j].max()*100)

            # Plotting starts here
            ax = fig.add_subplot(1,1,1)
            rr_plt, = ax.plot(x_idx,max_rr,color,marker=marker)

            ax.set_xlabel('Enrollment data [num. sentences]')
```

```
            ax.set_ylabel('Recognition rate [%]')

        if log_plot:
            ax.set_yscale('log')
            ax.set_title(plot_title+' - log')

        if not more_test_data:
            ax.set_title('I-vector recognition rate vs. enrollment data')
        else:
            ax.set_title('I-vector recognition rate vs. enrollment data, more test data')
        ax.set_ylim([30,85])
        ax.grid()

        red_line = mlines.Line2D([],[],color='red',label='cos',marker='o')
        green_line = mlines.Line2D([],[],color='green',label='cos+lda',marker='s')
        blue_line = mlines.Line2D([],[],color='blue',label='cos+wccn',marker='^')
        mag_line = mlines.Line2D([],[],color='magenta',label='cos+lda+wccn',marker='v')
        yellow_line = mlines.Line2D([],[],color='black',label='plda',marker='*')

        # Shrink current axis's height by 10% on the bottom
        box = ax.get_position()
        ax.set_position([box.x0, box.y0 + box.height * 0.1,
                         box.width, box.height * 0.9])

        # Put a legend below current axis
        ax.legend(loc='upper center',
                  handles=[red_line,green_line,blue_line,mag_line,yellow_line],
                  bbox_to_anchor=(0.5, -0.15),
                  fancybox=True,
                  shadow=True,
                  ncol=5)

        if log_plot:
            fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_'+str(ww_adapt_num)+'_stc_3_6_9_log.png'))
        else:
            fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_'+str(ww_adapt_num)+'_stc_3_6_9.png'))
        print('- saved figure')


# Plot rate of recognized known and unknown speakers for each system
if plot_known_unknown:
    print('Plot known/unknown speaker recognition rates:')
    # Modified from https://matplotlib.org/examples/api/barchart_demo.html
    # Input data from naive scoring and threshold scoring
    # These numbers are obtained from 'log/ww_3_stc_3/evaluate_ivec_plda_ubm_ww_3_stc_3_{system}.txt'
    # Indices corresponds to ['cos','cos_lda','cos_wccn','cos_lda_wccn','cos_plda']
    if not more_test_data:
```

```
        kn_naive = [73.0, 60.0, 44.63, 58.13, 70.13]
        un_naive = [3.5, 10.5, 0.0, 8.0, 10.0]
        kn_thres = [65.63, 53.63, 40.5, 53.13, 62.5]
        un_thres = [31.5, 33.0, 20.0, 26.5, 39.0]
    else:
        kn_naive = [69.13, 53.63, 42.5, 52.5, 72.25]
        un_naive = [3.5, 6.0, 0.0, 3.0, 13.0]
        kn_thres = [62.5, 47.88, 38.38, 46.38, 64.63]
        un_thres = [33.5, 28.0, 19.0, 27.0, 36.5]

    # 1: NAIVE DECISION
    x_idx = numpy.arange(5)
    y_idx = numpy.arange(0,101,10)
    width = 0.25

    fig, ax = plt.subplots()
    rects_kn = ax.bar(x_idx, kn_naive, width, color='b')
    rects_un = ax.bar(x_idx+width, un_naive, width, color='g')

    ax.set_ylabel('Relative recognition rate [%]')
    if not more_test_data:
        ax.set_title('I-vector: Known/unknown speaker trials, naive
    decision',fontsize=10)
    else:
        ax.set_title('I-vector: Known/unknown speaker trials, naive
    decision, more test data',fontsize=10)
    ax.set_xticks(x_idx+width/2)
    ax.set_xticklabels(('cos','cos+lda','cos+wccn','cos+lda+wccn','plda'))
    ax.set_yticks(y_idx)
    ax.set_ylim([0,100])


    # Shrink current axis's height by 10% on the bottom
    box = ax.get_position()
    ax.set_position([box.x0, box.y0 + box.height * 0.1,
                     box.width, box.height * 0.9])

    # Put a legend below current axis
    ax.legend((rects_kn[0],rects_un[0]), ('Known','Unknown'),
              bbox_to_anchor=(0.75, -0.06),
              fancybox=True,
              shadow=True,
              ncol=5)
    ax.grid()

    fig.savefig(os.path.join(fig_dir,'comparison','
    ivec_known_unknown_rr_naive.png'))
    print('- saved known/unknown recognition rate for naive decision')

    # 2: THRESHOLD DECISION
    x_idx = numpy.arange(5)
    y_idx = numpy.arange(0,101,10)
    width = 0.25

    fig, ax = plt.subplots()
    rects_kn = ax.bar(x_idx, kn_thres, width, color='b')
    rects_un = ax.bar(x_idx+width, un_thres, width, color='g')
```

```
268
269     ax.set_ylabel('Relative recognition rate [%]')
270     if not more_test_data:
271         ax.set_title('I-vector: Known/unknown speaker trials, threshold
        decision',fontsize=10)
272     else:
273         ax.set_title('I-vector: Known/unknown speaker trials, threshold
        decision, more test data',fontsize=10)
274     ax.set_xticks(x_idx+width/2)
275     ax.set_xticklabels(('cos','cos+lda','cos+wccn','cos+lda+wccn','plda'))
276     ax.set_yticks(y_idx)
277     ax.set_ylim([0,100])
278
279     # Shrink current axis's height by 10% on the bottom
280     box = ax.get_position()
281     ax.set_position([box.x0, box.y0 + box.height * 0.1,
282                      box.width, box.height * 0.9])
283
284     # Put a legend below current axis
285     ax.legend((rects_kn[0],rects_un[0]), ('Known','Unknown'),
286               bbox_to_anchor=(0.75, -0.06),
287               fancybox=True,
288               shadow=True,
289               ncol=5)
290     ax.grid()
291
292     fig.savefig(os.path.join(fig_dir,'comparison','
        ivec_known_unknown_rr_threshold.png'))
293     print('- saved known/unknown recognition rate for threshold decision')
```

Bottleneck features

The following scripts train a phonetically aware DNN, then extracts bottleneck features from Fisher English part 1. It relies on first running phonetic alignment in Kaldi, located in Kaldi/egs/fisher_english.

Train DNN: DNN/train_dnn.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tuesday Dec 2 2015
4
5  @author: Anthony Larcher
6
7  This script has to be run after the sre10_init.py located in the same
       tutorial.
8  Once all filter-banks features have been extracted, run this script to
       train
9  a feed-forward deeep neural network including a bottleneck layer and
       extract
10 all necessary features for the NIST-SRE10 evaluation
11 """
12
13
14 # Edited by J rgen Antonsen
```

```
# May 4, 2017

import sys
import os
import random
import numpy as np
#import pandas as pd
#pd.set_option('display.mpl_style', 'default')
from fnmatch import fnmatch
from collections import namedtuple
import pickle
import multiprocessing
import sidekit

# For Theano debugging
import theano
theano.config.optimizer='fast_compile'
theano.config.exception_verbosity='high'
#theano.config.compute_test_value = 'warn'

# Set parameters
nbThread = max(multiprocessing.cpu_count()-1,1)  # Number of parallel
    processes (for feature extraction).

feature_root_dir = 'fisher_english/feats/'
feature_dim = 19 # number of mfcc coefficients
left_context = 7
right_context = 7
input_size = feature_dim * (left_context + 1 + right_context) # size of
    bottom layer of network

feature_extension = '.h5'
bnf_feature_extension = '.bnf'

train_dnn = True
extract_bnf = False

load_absolute_path = True


# Train DNN / Extract bnf
if train_dnn:
    """ Train the DNN and save parameters to disk """

    label_file_name = "fisher_english/ali_all.txt"
    training_feature_dir = 'fisher_english/feats/'
    show_prefix = 'LDC2004S13/fisher_eng_tr_sp/audio/'
    feature_extension = '.h5'
    absolute_file_path = 'fisher_english/ali_full_paths.txt'

    # Before training you need a file with absolute paths (takes approx. a
     few hours to make)
    if not load_absolute_path:
        print('Make file with absolute paths (takes a few hours..)')
        with open(absolute_file_path,'w') as fh_path:
            # Make list of absolute paths of feature files
            seg_paths = []
```

```
            for root, dirs, files in os.walk(training_feature_dir):
                for file in files:
                    if file.endswith(feature_extension):
                        seg_paths.append(os.path.join(root, file))

            # Load alignment file
            with open(label_file_name,'r') as fh_ali:
                seg_list = fh_ali.read().split('\n')
            if '' in seg_list:
                seg_list.remove('')

            # Align absolute paths to files in ali-file
            for segment in seg_list:
                for paths in seg_paths:
                    if paths.endswith(segment.split('-')[0]+'-'+segment.
    split('-')[1]+'.h5'):
                        fh_path.write(paths+'\n')
                        break
            print('- done')

    # Load the labels and get the number of output classes
    print("Load training feature labels and their absolute paths")
    sys.stdout.flush()
    with open(label_file_name,'r') as fh:
        seg_list = fh.read().split('\n')
    seg_list.remove('')
    with open(absolute_file_path,'r') as fh:
        path_list = fh.read().split('\n')
    path_list.remove('')
    assert len(seg_list) == len(path_list), 'Segment list and path list
    mismatch'

    # ----- REMOVE ----
    # Selection of 100 first lines, for debug
    seg_list = seg_list[:100]
    path_list = path_list[:100]
    # --- REMOVE END ---

    print('- processing contents..')
    sys.stdout.flush()
    nclasses = 0
    for i in range(len(seg_list)): # split into ('segment id', 'channel',
    'start', 'stop', 'alignments')
        temp1, ali = seg_list[i].split(' ')[0], seg_list[i].split(' ')[1:]
        while '' in ali:
            ali.remove('')

        ali = np.array([int(a)-1 for a in ali]) # extract 1 to make
    classes start at 0, not 1
        if ali == []:
            raise ValueError('Error in reading alignments, [] detected')

        segment, channel, start, stop = temp1.split('-')
        if segment == '' or channel == '' or start == '' or stop == '':
            raise ValueError('Error in reading segment, channel, start and
     stop values')
```

```
        # Add segment and info to seg_list
        seg_list[i] = (path_list[i], int(start), int(stop), ali)

        if ali.max() > nclasses:
            nclasses = ali.max()
    nclasses += 1 # need to count class 0 as well

    print("Number of output classes = {}".format(nclasses))


    # Split list of segments into lists for training and cross-validation
    print("Split the list of segments for training and cross-validation")
    sys.stdout.flush()
    idx = np.random.permutation(len(seg_list)).astype('int')
    training_seg_list = [seg_list[ii] for ii in idx[:int(len(seg_list)
    *0.9)]]
    cv_seg_list = [seg_list[ii] for ii in idx[int(len(seg_list)*0.9):]]

    # Free lists not needed anymore from memory
    del seg_list, path_list

    # Define the structure of the NN
    print('Initialize network and FeaturesServer')
    sys.stdout.flush()
    FfNn = sidekit.FForwardNetwork(filename=None,
                                   input_size=input_size,
                                   input_mean=np.empty(input_size),
                                   input_std=np.empty(input_size),
                                   hidden_layer_sizes=(1200, 1200, 80,
    1200, 1200),
                                   layers_activations=("sigmoid", "sigmoid
    ", None, "sigmoid", "sigmoid","softmax"),
                                   n_classes=nclasses)

    # Initialize FeaturesServer
    features_server = sidekit.FeaturesServer(features_extractor=None,
                                             feature_filename_structure=
    training_feature_dir+"{}.h5",
                                             sources=None,
                                             dataset_list=["cep"],#, "vad"],
                                             mask=None,
                                             feat_norm="cmvn",
                                             global_cmvn=None,
                                             dct_pca=False,
                                             dct_pca_config=None,
                                             sdc=False,
                                             sdc_config=None,
                                             delta=False,
                                             double_delta=False,
                                             delta_filter=None,
                                             context=(left_context,
    right_context),
                                             traps_dct_nb=None,
                                             rasta=True,
                                             keep_all_features=False)


```

```
    # Start training of the network
    print("Training the network on Fisher pt 1 ...")
    sys.stdout.flush()
    FfNn.train_acoustic(training_seg_list=training_seg_list,
                        cross_validation_seg_list=cv_seg_list,
                        features_server=features_server,
                        feature_size=input_size,
                        lr=0.008,
                        segment_buffer_size=10,#200,
                        batch_size=50,#512,
                        max_iters=20,
                        tolerance=0.003,
                        output_file_name="BNF_network_fe_p1",
                        save_tmp_nnet=True,
                        traps=False,
                        num_thread=nbThread)
    """
    # OLD METHOD
    # Start training of the network
    print("Training the network on Fisher pt 1 ...")
    sys.stdout.flush()
    FfNn.train(training_set=training_seg_list,
               cross_validation_set=cv_seg_list,
               lr = 0.008,
               batch_size=512,
               max_iters=20,
               tolerance=0.003,
               output_file_name="BNF_network_fe_p1",
               save_tmp_nnet=True,
               num_thread=nbThread)
    """
if extract_bnf:
    print(os.environ["THEANO_FLAGS"])

    # Get the list of features to process
    bnf_file_list = []
    with open('sph_files_to_process.p', "rb" ) as f:
        audio_file_list, feature_file_list = pickle.load (f)
    for ff in feature_file_list:
        possible_fname = [ff, ff + '_a', ff + '_b']
        for fn in possible_fname:
            if os.path.exists(os.path.join(feature_root_dir, fn +
    fb_feature_extension)):
                bnf_file_list.append(fn)
    print("{} bottleneck feature files to extract".format(len(
    bnf_file_list)))

    # Load NN parameters
    FfNn = sidekit.FForwardNetwork(filename="BNF_network_epoch10.npz")

    # Extract Bottleneck features using the first 3 bottom layers of the
    network

    # Run Bottleneck features computation in parallel on CPU
    if "cpu" in os.environ["THEANO_FLAGS"]:
        import multiprocessing

```

```
        print("Split the list of files to process")
        sub_lists = [list(l) for l in np.array_split(bnf_file_list,
    nbThread)]

        print("Extract Bottleneck features in parallel")
        jobs = []
        multiprocessing.freeze_support()
        for idx in range(nbThread):
            # Create argument list for each process
            ff_args = {'layer_number':3,
                       'feature_file_list':sub_lists[idx],
                       'input_dir':feature_root_dir,
                       'input_file_extension':".fb",
                       'label_dir':feature_root_dir,
                       'label_extension':".lbl",
                       'output_dir':feature_root_dir+"bnf/",
                       'output_file_extension':".bnf",
                       'input_feature_format':"spro4",
                       'output_feature_format':"spro4",
                       'feature_context':(7, 7),
                       'normalize_output':"cmvn",
                       'log':None}

            p = multiprocessing.Process(target=FfNn.feed_forward, kwargs=
    ff_args)
            jobs.append(p)
            p.start()
        for p in jobs:
            p.join()

    # Run Bottleneck features computation on a singleGPU
    elif "gpu" in os.environ["THEANO_FLAGS"]:
        FfNn.feed_forward(feature_file_list=bnf_file_list,
                      layer_number=3,
                      input_dir=feature_root_dir,
                      input_file_extension=".fb",
                label_dir=feature_root_dir,
                      label_extension=".lbl",
                      output_dir=feature_root_dir+"bnf/",
                      output_file_extension=".bnf",
                      input_feature_format="spro4",
                      output_feature_format="spro4",
                      feature_context=(7, 7),
                      normalize_output="cmvn",
                      log=None)
```

Extract bottleneck features: DNN/extract_bnf.py

```
# This script extracts bottleneck-features (bnf) from existing spectral
    features,
# storing them to feats/bnf/..
# A dnn (located in ../dnn) must be trained before running this script.
    Scripts
# for dnn training can be found in the same directory.

import sidekit
import os
```

```
import time

print(time.strftime('%c'))
print('BOTTLENECK-FEATURE EXTRACTION')

# Set parameters
dataset = 'LDC' # 'cat' 'RSR2015' 'LDC'
bn_layer = 3 # index of hidden layer to extract bnf from
feature_size = 19  # mfcc

print('Dataset: '+dataset)


# Set file paths
base_dir = '/home/studenter/jorgeja/Projects/master/'
dnn_path = os.path.join(base_dir,'dnn/dnn.h5')
data_path = os.path.join(base_dir,'data')
out_path = os.path.join(data_path,'feats/bnf')

if dataset == 'cat':
    feat_path = os.path.join(data_path,'feats/RSR2015/cat')
elif dataset == 'RSR2015':
    feat_path = os.path.join(data_path,'feats/RSR2015')
elif dataset == 'LDC':
    feat_path = os.path.join(data_path,'feats/LDC/new')
else:
    raise ValueError('Dataset named '+dataset+' does not exist.')
if not os.path.exists(feat_path):
    raise ValueError('Feature path '+feat_path+' not found.')

if not os.path.exists(dnn_path):
    raise ValueError('Dnn with path '+dnn_path+' not found.')


# Make file lists according to the inner structure of the hdf5 feature
    files
show_list = []
gender_list = ['male','female']
for gender in gender_list:
    if dataset == 'RSR2015':
        subdir_list = os.listdir(os.path.join(feat_path,gender))
        for dirs in subdir_list:
            temp_list = os.listdir(os.path.join(feat_path,gender,dirs))
            temp_list = [os.path.join(gender,dirs,files.split('.')[0]) for
     files in temp_list]
            show_list += temp_list
    else: # dataset == 'cat'|'LDC'
        temp_list = os.listdir(os.path.join(feat_path,gender))
        temp_list = [os.path.join(gender,files.split('.')[0]) for files in
     temp_list]
        show_list += temp_list
if dataset == 'cat':
    show_list = [os.path.join('cat',files) for files in show_list]
    feat_path = os.path.join(data_path,'feats/RSR2015')

print('Number of files: '+str(len(show_list)))

```

```
62
63
64 # Initialize FeaturesServer to load features
65 features_server = sidekit.FeaturesServer(features_extractor=None,
66                                          feature_filename_structure=
       feat_path+"{}.h5",
67                                          sources=None,
68                                          dataset_list=["vad","cep"],
69                                          mask=None,
70                                          feat_norm="cmvn",
71                                          global_cmvn=None,
72                                          dct_pca=False,
73                                          dct_pca_config=None,
74                                          sdc=False,
75                                          sdc_config=None,
76                                          delta=True,
77                                          double_delta=True,
78                                          delta_filter=None,
79                                          context=None,
80                                          traps_dct_nb=None,
81                                          rasta=True,
82                                          keep_all_features=False)
83
84 # Load trained DNN
85 dnn = sidekit.nnet.FForwardNetwork.read(dnn_path)
86
87 # Extract and save bnf from files in show_list
88 dnn.feed_forward_acoustic(feature_file_list=show_list,
89                           features_server=features_server,
90                           layer_number=bn_layer,
91                           output_file_structure=os.path.join(out_path,
       dataset,bn_layer,"{}.h5")
92                           )
93
94 print('Success!')
95 print(time.strftime('%c'))
```