

---

# 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

```
1 # This script generates the IdMap file that defines the enrollment set in
  # a Sidekit
2 # speaker identification experiment on the RSR2015 database.
3 # Speaker models for t-normalization are also trained the same way as
  # ordinary
4 # enrollment speaker models.
5 # In addition to the speaker models, a 'complementary' model w is trained
  # on a larger
6 # amount of data to model all speakers outside the enrollment and t-norm
  # set.
7 #
8 # The script is based on the script 'rsr2015_init.py'.
9 # The set is customly defined to fit to the context of the specific task,
  # and uses
10 # data from session set A (sessions {1,4,7}). Sentences {001,002,003} from
  # session 1
11 # plus command 032 from sessions {1,4,7} is used, in total 6 adaptation
  # utterances
12 # for each target speaker model.
13 #
14 # Speakers with id m/f 001-050 is reserved for the enrollment/test
  # speakers
15 # Speakers with id m/f 051-100 is reserved for z-norm and t-norm
16 # Speakers with id m/f 101-157/143 is reserved for complementary model
  # adaptation
17 #
18 # Written by Joergen Antonsen, April 23 2017
19
20
21 import sidekit
22 import h5py
23 import numpy as np
24 import random
```

---

```

25 import os
26 random.seed(42) # to make the selected data predictable
27
28 print('- IdMap file generation -')
29
30 # -----Set parameters-----
31 feat_path = '/home/studenter/jorgeja/Projects/master/data/feats/RSR2015/'
32
33 test_set_size = 10 # number of speakers in test set
34 num_unknown = 2 # number of unknown speakers in test set
35 enroll_set_size = test_set_size-num_unknown # number of known speakers in
    test set
36 num_sets = 100 # number of unique test sets
37
38 enroll_range = 50 # max index of speakers in enrollment set
39 norm_range = 100 # max index of speakers in normalization set
40 separate_gender = False # if true, gender separate IdMap files are
    generated
41
42 ww_adapt_num = 3 # number of wake words for model enrollment (IF > 3, YOU
    MUST ALSO EDIT generate_test.py)
43 stc_adapt_num = 3 # number of sentences for model enrollment
44 spkr_adapt_num = ww_adapt_num+stc_adapt_num # total number of enrollment
    utterances per speaker
45 world_adapt_num = 1 # total number of complementary model adaptation
    utterances (per speaker)
46
47 gender_name = ['male','female']
48 if ww_adapt_num <= 3:
49     enroll_session_list = ['01','04','07']
50 else:
51     enroll_session_list = ['01','04','07','02','05','08']
52 full_session_list = ['0'+str(i+1) for i in range(9)]
53 session_groups = [['01','04','07'],['02','05','08'],['03','06','09']]
54 sentence_id = ['001','002','003','004','005','006','007','008','009']
55 full_sentence_list = ['00'+str(i+1) for i in range(9)]+['0'+str(i+1) for i
    in range(9,73)]
56
57 ww_id = '032' # wake word id
58
59 assert norm_range >= 100, 'Script must be edited if norm_range is less
    than 100'
60 assert world_adapt_num <= 9, 'world_adapt_num is too big. Value or entire
    script must be edited.'
61 assert ww_adapt_num <= 6, 'ww_adapt_num is too big. Max size allowed is 6.
    '
62
63 # -----Generate model and segment lists-----
64 # First we make a list of the speakers in the enrollment set
65 num_list = [str(i+1) for i in range(enroll_range)]
66 num_list[0:9] = ['00'+num for num in num_list[0:9]]
67 num_list[9:enroll_range] = ['0'+num for num in num_list[9:enroll_range]]
68 spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]
69
70 # Then we make a speaker list for the models for t-normalization
71 num_list = [str(i+1) for i in range(enroll_range,norm_range)]
72 for i in range(len(num_list)):

```

---

---

```

73     if len(num_list[i]) == 2:
74         num_list[i] = '0'+num_list[i]
75 if not separate_gender:
76     spkr_norm_list = ['m'+num for num in num_list]+['f'+num for num in
num_list]
77 else:
78     spkr_norm_list = ['m'+num for num in num_list],['f'+num for num in
num_list]]
79
80 # Finally, we make a speaker list for the complementary model training set
81 num_list_m = [str(i+1) for i in range(norm_range,157)]
82 num_list_f = [str(i+1) for i in range(norm_range,143)]
83 if not separate_gender:
84     spkr_comp_list = ['m'+num for num in num_list_m]+['f'+num for num in
num_list_f]
85 else:
86     spkr_comp_list = ['m'+num for num in num_list_m],['f'+num for num in
num_list_f]]
87
88
89 # -----Generate IdMap file contents-----
90 if not separate_gender:
91     # MODEL LIST
92     models = []
93     # Add enrollment speaker model ids to model list
94     for spkr in spkr_list:
95         models.extend([spkr for i in range(spkr_adapt_num)])
96     # Add speaker models for t-normalization
97     for spkr in spkr_norm_list:
98         models.extend([spkr for i in range(spkr_adapt_num)])
99     # Add complementary model id ('w')
100    models.extend(['w' for i in range((300-norm_range*2)*world_adapt_num)
])
101
102
103    # SEGMENT LIST
104    segments = []
105    # Add enrollment and t-norm speaker utterances to segment list
106    for spkr in spkr_list+spkr_norm_list:
107        # 1: Add wake word from a number (ww_adapt_num) of sessions
108        for sess in enroll_session_list:
109            if spkr[0] == 'm':
110                file_path = os.path.join('male', spkr, spkr+'_'+sess+'_'+
ww_id)
111            elif spkr[0] == 'f':
112                file_path = os.path.join('female', spkr, spkr+'_'+sess+'_'+
ww_id)
113            if os.path.exists(os.path.join(feet_path, file_path+'.h5')):
114                segments.append(file_path)
115            else:
116                raise Exception(file_path+'.h5 does not exist.')
117
118        # 2: Add a number (stc_adapt_num) of phrases from the sessions in
enroll_session_list
119        # In 'most' cases, these phrases will be chosen from session '01'
120        sess_index = 0
121        sess_list = []

```

---

---

```

122         if spkr[0] == 'm':
123             gender = 'male'
124         else:
125             gender = 'female'
126
127         for i in range(stc_adapt_num):
128             sess = random.choice(enroll_session_list)
129             file_exists = False
130             sess_idx = 0 # To be increased if file does not exist
131             while not file_exists:
132                 sess = enroll_session_list[sess_idx]
133                 seg_id = sentence_id[i]
134                 file_path = os.path.join(gender, spkr, spkr+'_'+sess+'_'+
seg_id)
135                 if os.path.exists(os.path.join(feats_path, file_path+'.h5'))
and file_path not in segments:
136                     segments.append(file_path)
137                     file_exists = True
138                     sess_idx = 0
139                 else:
140                     sess_idx += 1
141
142         # Add utterances for complementary model adaptation
143         # Sessions and utterance id's are randomly selected
144         for spkr in spkr_comp_list:
145             files_exists = False
146             while not files_exists:
147                 if spkr[0] == 'm':
148                     gender = 'male'
149                 else:
150                     gender = 'female'
151                 sess = random.sample(full_session_list, world_adapt_num)
152                 utt = random.sample(full_sentence_list[0:30], world_adapt_num)
153                 # selecting only phrases, not short commands or digits
154                 # First check for existence (if not, redo selection)
155                 for idx in range(world_adapt_num):
156                     file_path = os.path.join(gender, spkr, spkr+'_'+sess[idx]+'_'
+utt[idx])
157                     files_exists = True
158                     if not os.path.exists(os.path.join(feats_path, file_path+'.
h5')):
159                         files_exists = False
160                         break
161                 # Then add segments to segment list
162                 if files_exists:
163                     for idx in range(world_adapt_num):
164                         file_path = os.path.join(gender, spkr, spkr+'_'+sess[idx
]+'_'+utt[idx])
165                         segments.append(file_path)
166             else:
167                 # Write gender-dependent code when needed
168                 pass
169
170
171
172 # -----Make and save IdMap-files-----

```

---

---

```

173 if separate_gender:
174     # Make two IdMap-files
175     for gender in [0,1]:
176         enroll_idmap = sidekit.IdMap()
177         enroll_idmap.lefttids = np.asarray(models[gender])
178         enroll_idmap.righttids = np.asarray(segments[gender])
179         enroll_idmap.start = np.empty(enroll_idmap.righttids.shape, 'iO')
180         enroll_idmap.stop = np.empty(enroll_idmap.righttids.shape, 'iO')
181         valid = enroll_idmap.validate()
182         if not valid:
183             raise ValueError('Error in generating IdMap file.')
184         enroll_idmap.write('enroll_'+gender_name[gender]+'_'+str(
185             enroll_size)+'_ww_'+str(ww_adapt_num)+'_'+str(stc_adapt_num)+'.h5')
186 else:
187     # Make one IdMap-file
188     enroll_idmap = sidekit.IdMap()
189     enroll_idmap.lefttids = np.asarray(models)
190     enroll_idmap.righttids = np.asarray(segments)
191     enroll_idmap.start = np.empty(enroll_idmap.righttids.shape, 'iO')
192     enroll_idmap.stop = np.empty(enroll_idmap.righttids.shape, 'iO')
193     valid = enroll_idmap.validate()
194     if not valid:
195         raise ValueError('Error in generating IdMap file.')
196     enroll_idmap.write('enroll_gidp_{t_norm}_{ww}_{stc}_{h5}'\
197                       .format(str(enroll_range), str(norm_range-
198                               enroll_range),\
199                               str(ww_adapt_num), str(stc_adapt_num)))
199 print('Success!')

```

## task/generate\_test.py

```

1 # This script generates a random test set for a speaker identification
2   experiment in Sidekit.
3 # Its output consists of two files:
4 # - 'eval....txt', which can be read directly by sidekit.Key() using Key('
5   eval....txt')
6 # - 'cat_list....sh', which lists files to be concatenated and stored in
7   '../data/sph/cat/',
8 # and requires permission change (chmod +x <filename>) to be executeable.
9 #
10 # All test data is from the RSR2015 database session sets B (sessions
11   {2,5,8}) and C (sessions {3,6,9})
12 # For each trial speaker, a concatenated phrase made by segments from the
13   same session is generated.
14 # The phrase consists of command segment 032 (wake word) and one of the
15   sentence segments 010-030.
16 #
17 # The test sets are open sets with a number of known and unknown speakers.
18 # The known speakers (enrollment speakers) have MAP-adapted speaker models
19   (id 01-50).
20 # All test speakers are scored against these models and the adapted
21   complementary model (trained on id 101-157/143).
22 # From the current parameters, there are 100 test sets, each consisting of
23   10 speakers.
24 # In addition, all test speakers are scored against a number of models (id
25   51-100) for t-normalization,

```

---

```

16 # and a number of utterances from the same set of speakers are scored
    against the enroll-set for z-normalization.
17 #
18 # NOTE: Subdirectories under cat_dir: male/ and female/, must exist in
    advance
19 #
20 # Written by Joergen Antonsen, April 24 2017
21
22
23 import os
24 import copy
25 import random
26 random.seed(42) # make random.shuffle() do the same every time this script
    is executed.
27
28 print('- Test set and concatenation script generation -')
29
30 # -----Set parameters-----
31 base_dir = '/home/studentenr/jorgeja/Projects/master/'
32 rsr2015path = base_dir+'data/sph/RSR2015/'
33 cat_dir = base_dir+'data/sph/cat/' # output location of concatenated files
34 data_path = base_dir+'data/'
35 feat_path = base_dir+'data/feats/RSR2015'
36
37 test_set_size = 10 # number of speakers in the test set
38 num_unknown = 2 # number of unknown speakers in test set
39 enroll_set_size = test_set_size-num_unknown # number of known speakers in
    test set
40 num_sets = 100 # number of unique test sets
41
42 enroll_range = 50 # max index of speakers in enrollment set
43 norm_range = 100 # max index of speakers in normalization set
44 separate_gender = False # if true, gender separate Key/Ndx files are
    generated
45
46 spkr_adapt_num = 6 # number of speaker model adaptation utterances (per
    speaker)
47 world_adapt_num = 1 # total number of complementary model adaptation
    utterances (per speaker)
48
49 gender_list = ['m','f']
50 gender_name = ['male/','female/'] # for path names
51 test_session_list = ['02','03','05','06','08','09']
52 full_session_list = ['0'+str(i+1) for i in range(9)]
53 session_groups = [['01','04','07'],['02','05','08'],['03','06','09']]
54 command_id = ['0'+str(i+1) for i in range(9,30)]
55 ww_id = '032' # wake word id
56
57 assert norm_range >= 100, 'Script must be edited if norm_range is less
    than 100'
58 assert world_adapt_num <= 9, 'world_adapt_num is too big. Value or entire
    script must be edited.'
59
60
61 # -----Generate model and segment lists-----
62 # First we make a list of the speakers in the enrollment set
63 num_list = [str(i+1) for i in range(enroll_range)]

```

---

---

```

64 num_list[0:9] = ['00'+num for num in num_list[0:9]]
65 num_list[9:enroll_range] = ['0'+num for num in num_list[9:enroll_range]]
66 spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]
67
68 # We then do random selection of num_sets sets of speakers
69 # These test sets are stored in test_set with the following format for
    each test:
70 # [tg, tg, tg, tg, tg, tg, tg, imp, imp], tg = target, imp = 'impostor'
71
72 if not separate_gender:
73     test_set = []
74     spkr_list = ['m'+num for num in num_list]+['f'+num for num in num_list]
75     for sets in range(num_sets):
76         test_set.append(random.sample(spkr_list, test_set_size))
77         with open('test_set_list_gidp_'+str(test_set_size)+'_spk_'+str(
            num_sets)+'_sets.txt', 'w') as fh:
78             for sets in test_set:
79                 for spkr in sets:
80                     fh.write(spkr+' ')
81             fh.write('\n')
82 else:
83     test_set = [[], []]
84     spkr_list = [['m'+num for num in num_list], ['f'+num for num in
        num_list]]
85     for gender in [0, 1]:
86         for sets in range(num_sets):
87             test_set[gender].append(random.sample(spkr_list[gender],
                test_set_size))
88             with open('test_set_list_'+gender_name[gender]+'_'+str(
                test_set_size)+'_spk_'+str(num_sets)+'_sets.txt', 'w') as fh:
89                 for sets in test_set[gender]:
90                     for spkr in sets:
91                         fh.write(spkr+' ')
92                     fh.write('\n')
93
94 # -----Define tests and make test lists-----
95 # First, we define some speaker lists:
96 enroll_spkr_list = spkr_list
97 num_list = [str(i+1) for i in range(enroll_range, norm_range)]
98 for i in range(len(num_list)):
99     if len(num_list[i]) == 2:
100         num_list[i] = '0'+num_list[i]
101 if not separate_gender:
102     norm_spkr_list = ['m'+num for num in num_list]+['f'+num for num in
        num_list]
103 else:
104     norm_spkr_list = [['m'+num for num in num_list], ['f'+num for num in
        num_list]]
105
106 # Then, lists to hold the models and test segments, and identity labels
    plus concatenation list
107 if not separate_gender:
108     model_list = []
109     test_segment_list = []
110     identity_lbl_list = []

```

---

---

```

111 cat_list = [] # format: [[wake_word],[command],[concatenated segment]]
112
113 # 1: For each test set, score all speakers against enrolled speaker
114 # models
115 for tests in test_set:
116     # Score against all speakers for each test
117     for j in range(test_set_size): # j denotes both known and unknown
118         speakers)
119         # Select session and utterances for testing (wake word+command
120         )
121         test_spkr = tests[j]
122         gender = 'male' if test_spkr[0] == 'm' else 'female'
123         file_exists = False
124         while not file_exists:
125             sess = random.choice(test_session_list)
126             com_id = random.choice(command_id)
127             ww_path = os.path.join(gender,test_spkr,test_spkr+'_'+sess
128             +'_'+ww_id)
129             com_path = os.path.join(gender,test_spkr,test_spkr+'_'+
130             sess+'_'+com_id)
131             cat_path = os.path.join('cat',gender,test_spkr+'_'+sess+'_'
132             +ww_id+'_'+com_id)
133             if os.path.exists(os.path.join(rsr2015path,'sph',ww_path+'
134             .sph')) and os.path.exists(os.path.join(rsr2015path,'sph',com_path+'
135             .sph')) and cat_path not in test_segment_list:
136                 for i in range(enroll_set_size): # i denotes speaker
137                 models to be tested
138                     enroll_spkr = tests[i]
139                     model_list.append(enroll_spkr)
140                     test_segment_list.append(cat_path)
141                     if i == j: # target trial
142                         identity_lbl_list.append('target')
143                     else: # nontarget trial
144                         identity_lbl_list.append('nontarget')
145                     cat_list.append([ww_path,com_path,cat_path])
146                     file_exists = True
147
148 # 2: t-normalization: Score each test speaker utterance against
149 # enrolled norm-models
150 orig_test_segment_list = list(set(test_segment_list))
151
152 for norm_spkr in norm_spkr_list:
153     for segment in orig_test_segment_list:
154         if norm_spkr[0] == segment.split('/')[2][0]: # NOTE:
155             Normalization is gender-dependent
156             # NOTE cont.: This can be avoided by removing the above if
157             -condition
158             # NOTE cont.: and using the commented code below instead
159             model_list.append(norm_spkr)
160             test_segment_list.append(segment)
161             identity_lbl_list.append('nontarget')
162             # The following commands can be used in case of gender-independent
163             normalization
164             # model_list.extend([norm_spkr for i in range(len(
165             orig_test_segment_list))])
166             # test_segment_list.extend(orig_test_segment_list)

```

---



---

```

154         # identity_lbl_list.extend('nontarget' for i in range(len(
155         orig_test_segment_list)))
156
157     current_list_pos = len(test_segment_list) # current position in model
158     and test segment lists
159
160     # 3: z-normalization: Score z-/t-norm-speaker utterances against all
161     test speaker models,
162     # and the adapted complementary model
163     for enroll_spkr in enroll_spkr_list:
164         for spkr in norm_spkr_list:
165             if enroll_spkr[0] == spkr[0]: # NOTE: Only utterances with the
166                 same gender as the model is scored
167                 # NOTE cont.: Simply remove the above if-condition for
168                 gender-independent normalization
169                 gender = 'male' if spkr[0] == 'm' else 'female'
170                 # Select session and utterances (wake word+command)
171                 file_exists = False
172                 while not file_exists:
173                     sess = random.choice(full_session_list)
174                     com_id = random.choice(command_id)
175                     ww_path = os.path.join(gender, spkr, spkr+'_'+sess+'_'+
176                     ww_id)
177                     com_path = os.path.join(gender, spkr, spkr+'_'+sess+'_'+
178                     com_id)
179                     cat_path = os.path.join('cat', gender, spkr+'_'+sess+'_'
180                     +ww_id+'_'+com_id)
181                     if os.path.exists(os.path.join(rsr2015path, 'sph',
182                     ww_path+'.sph')) and os.path.exists(os.path.join(rsr2015path, 'sph',
183                     com_path+'.sph')) and cat_path not in test_segment_list[
184                     current_list_pos:len(test_segment_list)]:
185                         # Add scoring against enroll speaker
186                         model_list.append(enroll_spkr)
187                         test_segment_list.append(cat_path)
188                         identity_lbl_list.append('nontarget')
189                         # Add scoring against complementary model
190                         model_list.append('w')
191                         test_segment_list.append(cat_path)
192                         identity_lbl_list.append('nontarget')
193                         # Append to concatenation list
194                         cat_list.append([ww_path, com_path, cat_path])
195                         file_exists = True
196
197     # 4: Score each test speaker utterance against adapted complementary
198     model
199     for segment in orig_test_segment_list:
200         test_segment_list.append(segment)
201         model_list.append('w')
202         identity_lbl_list.append('nontarget')
203
204     print('Total number of tests: '+str(len(model_list)))
205     print('Total number of files to concatenate: '+str(len(cat_list)))
206     print(' -about '+str(130*len(cat_list)/1000)+' MB space for audio (sph
207     ) cat files')
208     print(' -about '+str(50*len(cat_list)/1000)+' MB space for feature cat

```

---

---

```

    files (.h5) after extraction')
198
199     if len(model_list) != len(test_segment_list) or len(test_segment_list)
        != len(identity_lbl_list):
200         raise Exception('Length of model_list, test_segment_list and
            identity_lbl_list is not equal!')
201
202     else:
203         # Write gender-dependent code when needed
204         pass
205
206
207
208     # -----Make sh-file to define concatenation-----
209     if not separate_gender:
210         cat_filename = 'concatenate_gidp_'+str(test_set_size)+'_spk_'+str(
            num_sets)+'_sets_more_data.sh'
211     else:
212         cat_filename = 'concatenate_gdp_'+str(test_set_size)+'_spk_'+str(
            num_sets)+'_sets_more_data.sh'
213
214     with open(cat_filename, 'w') as cat_file:
215         cat_file.write('#!/bin/bash\n')
216         # Write all files in cat_list to file
217         for segment in cat_list:
218             temp_ww_id = os.path.join(rsr2015path, 'sph', segment[0]+'_sph')
219             temp Utt_id = os.path.join(rsr2015path, 'sph', segment[1]+'_sph')
220             temp_cat_id = os.path.join(data_path, 'sph', segment[2]+'_sph')
221             cat_file.write('sox '+temp_ww_id+' '+temp Utt_id+' '+temp_cat_id+'
                \n')
222
223
224     # -----Make key-txtfile to define trials-----
225     if not separate_gender:
226         key_filename = 'eval_gidp_'+str(test_set_size)+'_spk_'+str(num_sets)+'
            _sets_more_data.txt'
227     else:
228         key_filename = 'eval_gdp_'+str(test_set_size)+'_spk_'+str(num_sets)+'
            _sets_more_data.txt'
229
230     with open(key_filename, 'w') as key_file:
231         # Add all test definitions
232         for i in range(len(model_list)):
233             key_file.write(model_list[i]+' '+test_segment_list[i]+' '+
                identity_lbl_list[i]+' \n')
234
235
236
237     # The following commands will not run on the sirkus8 server (location of
        Sidekit install),
238     # the reason of this being that 'sidekit/bosaris/key.py' does not support
        loading txt-files.
239     # 'key.py' can however be easily modified to support this, as Key.read_txt
        () already exists.
240     # See Key.__init__().
241
242

```

---

---

```

243 # -----Convert key txt-file to hdf5-----
244 """
245 key = sidekit.Key(key_name+'.txt')
246 key.write(key_name+'_key.h5')
247
248 # Make ndx-file from key and store in hdf5-format
249 ndx = key.to_ndx()
250 ndx.write(key_name+'_ndx.h5')
251 """

```

## 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

```

1 # This script extracts MFCC features from the utterances in /sph/cat/{male
  ,female}
2 # It is based on the svm-gmm tutorial from the SIDEKIT website
3
4 import numpy as np
5 import sidekit
6 import multiprocessing
7 import os
8 import sys
9 import time
10
11
12 print(time.strftime('%c'))
13 print('FEATURE EXTRACTION FROM ./sph/cat:')
14
15 base_dir = '/home/studenten/jorgeja/Projects/master/'
16
17
18 # Set the number of parallel process to run.
19 nbThread = max(multiprocessing.cpu_count()-1, 1)
20
21 # Make cat-dir file list
22 file_list = []
23 for gender_name in ['male/', 'female/']:
24     temp_list = os.listdir(base_dir+'data/sph/cat/'+gender_name)
25     temp_list = ['cat/'+gender_name+file.split('.')[0] for file in
        temp_list]
26     file_list.extend(temp_list)
27
28
29 # PROCESS THE AUDIO TO SAVE MFCC ON DISK
30 print("Initialize FeaturesExtractor")
31 extractor = sidekit.FeaturesExtractor(audio_filename_structure=base_dir+"
    data/sph/{}.sph",
32                                     feature_filename_structure=base_dir+
        "data/feats/RSR2015/more_data/{}.h5",
33                                     sampling_frequency=16000,

```

---

```

34         lower_frequency=300,
35         higher_frequency=3400,
36         filter_bank="log",
37         filter_bank_size=24,
38         window_size=0.02,
39         shift=0.01,
40         ceps_number=19,
41         vad="snr",
42         snr=40,
43         pre_emphasis=0.97,
44         save_param=["vad", "energy", "cep"],
45         keep_all_features=False)
46
47 # Get the complete list of features to extract
48 show_list = np.unique(np.hstack([file_list]))
49 channel_list = np.zeros_like(show_list, dtype = int)
50
51
52 print("Extract features and save to disk")
53 extractor.save_list(show_list=show_list,
54                    channel_list=channel_list,
55                    num_thread=nbThread)
56
57 print('Success!')
```

## UBM training

The following script trains the UBM used in all experiments.

### UBM/train\_ubm\_ldc.py

```

1 # This script trains a gender-independent ubm and stores it to gmm/
  gender_id_wo_fisher_ubm.h5
2 # It also trains gender-dependent ubms, stored to gmm/{male,female}
  _wo_fisher_ubm.h5
3 # It is based on the example scripts of the Sidekit documentation
4 # Written by Joergen Antonsen, April 2 2017
5 # Modified by Joergen Antonsen, April 4 2017
6
7 import sidekit
8 import os
9 import sys
10 import multiprocessing
11 import logging
12 import numpy as np
13 import time
14 import random
15
16 logging.basicConfig(filename='log/ldc_ubm.log', level=logging.DEBUG)
17
18 print(time.strftime('%c'))
19 start = time.time()
20 print('UBM TRAINING')
21 print('Prepare for training..')
22
```

---

---

```

23 # Set parameters
24 train_gender_models = True # if false, train gender-independent model
25 #gender_to_train = 'male' # 'male' or 'female'
26 distribNb = 1024 # number of GMM components
27 feat_dir = "/home/student/jorgeja/Projects/master/data/feats/LDC/new/"
28 selection_size = 1000 # size of ubm lists after data selection
29 random.seed(1) # to make the selected data predictable
30
31 # Automatically set the number of parallel process to run.
32 nbThread = max(multiprocessing.cpu_count()-1, 1)
33
34 """
35 # Read ubm-lists
36 if not train_gender_models:
37     with open('task/ubm_list_all.txt') as inputFile: # example line:
38         female/kcro
39         ubmList = inputFile.read().split('\n')
40         while '' in ubmList:
41             ubmList.remove('')
42 else:
43     with open('task/ubm_list_m.txt') as inputFile:
44         ubmList_m = inputFile.read().split('\n')
45         while '' in ubmList_m:
46             ubmList_m.remove('')
47     with open('task/ubm_list_f.txt') as inputFile:
48         ubmList_f = inputFile.read().split('\n')
49         while '' in ubmList_f:
50             ubmList_f.remove('')
51 """
52 # Make ubm-lists from random selections of features in ../data/feats
53 ubmList_m = os.listdir(feat_dir+'male')
54 ubmList_m = ['male/'+files.split('.')[0] for files in ubmList_m]
55 ubmList_f = os.listdir(feat_dir+'female')
56 ubmList_f = ['female/'+files.split('.')[0] for files in ubmList_f]
57
58 #ubmList = ubmList_m + ubmList_f
59
60 ubmList_m = random.sample(ubmList_m, selection_size)
61 ubmList_f = random.sample(ubmList_f, selection_size)
62 ubmList = ubmList_m[0:selection_size/2] + ubmList_f[0:selection_size/2]
63 """
64 with open('/home/student/jorgeja/Projects/master/ubm/lists/ubm_list_m_'+
65         str(selection_size)+'.txt','w') as fh:
66     for files in ubmList_m:
67         fh.write(files+'\n')
68 with open('/home/student/jorgeja/Projects/master/ubm/lists/ubm_list_f_'+
69         str(selection_size)+'.txt','w') as fh:
70     for files in ubmList_f:
71         fh.write(files+'\n')
72 with open('/home/student/jorgeja/Projects/master/ubm/lists/ubm_list_all_
73         '+str(selection_size)+'.txt','w') as fh:
74     for files in ubmList:
75         fh.write(files+'\n')
76 """
77 # Check size of file lists

```

---

---

```

76 temp_size_m = 0
77 for files in ubmList_m:
78     temp_size_m += os.stat(feats_dir+files+'.h5').st_size
79 temp_size_m /= 1000000
80 print('The size of the male set is '+str(temp_size_m)+'MB')
81
82 temp_size_f = 0
83 for files in ubmList_f:
84     temp_size_f += os.stat(feats_dir+files+'.h5').st_size
85 temp_size_f /= 1000000
86 print('The size of the female set is '+str(temp_size_f)+'MB')
87
88 temp_size = 0
89 for files in ubmList:
90     temp_size += os.stat(feats_dir+files+'.h5').st_size
91 temp_size /= 1000000
92 print('The size of the mixed set is '+str(temp_size)+'MB')
93
94 #if temp_size_m > 7000 or temp_size_f > 7000 or temp_size > 7000:
95 #    raise ValueError('Training set is too large.')
96
97
98 # Create a FeaturesServer to load features and feed the other methods
99 features_server = sidekit.FeaturesServer(features_extractor=None,
100                                         feature_filename_structure=
101                                             feats_dir+"{}.h5",
102                                         sources=None,
103                                         dataset_list=["vad", "cep"], #
104                                         NOTE: vad not in feature vector, only for frame selection
105                                         mask=None,
106                                         feat_norm="cmvn",
107                                         global_cmvn=None,
108                                         dct_pca=False,
109                                         dct_pca_config=None,
110                                         sdc=False,
111                                         sdc_config=None,
112                                         delta=True,
113                                         double_delta=True,
114                                         delta_filter=None,
115                                         context=None,
116                                         traps_dct_nb=None,
117                                         rasta=True,
118                                         keep_all_features=False)
119
120 # Load and preprocess features, and train ubm
121
122 # ----- GENDER INDEPENDENT -----
123 if not train_gender_models:
124     print('Train gender-independent UBM')
125     ubm = sidekit.Mixture()
126     llk = ubm.EM_split(features_server, ubmList, distribNb, num_thread=
127                         nbThread, save_partial=True)
128     ubm.write('gmm/gender_id_diag_ubm_'+str(distribNb)+'.h5')
129     print('Training done!')

```

---

---

```

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

135
136 """
137 print('Train full covariance UBM')
138 ubm_full = sidekit.Mixture()
139 ubm_full.EM_convert_full(diagonal_mixture=ubm,
140                          features_server=features_server,
141                          featureList=ubmList,
142                          distrib_nb=distribNb,
143                          iterations=2,
144                          num_thread=nbThread
145                          )
146 """
147
148 else: # gender_to_train == 'male':
149     # ----- MALE -----
150     print('Train male UBM')
151     start = time.time()
152     ubm_m = sidekit.Mixture()
153     llk_m = ubm_m.EM_split(features_server, ubmList_m, distribNb,
154                            num_thread=nbThread, save_partial=False)
155     ubm_m.write('gmm/male_diag_ubm_'+str(distribNb)+'.h5')
156     print('Training done!')
157     print(time.strftime('%c'))
158     end = time.time()
159     m, s = divmod(end-start, 60)
160     h, m = divmod(m, 60)
161     print("Time elapsed: "+str(round(h))+ 'H ' +str(round(m))+ 'M ' +str(round
(s))+ 'S')
162
163     # ----- FEMALE -----
164     print('Train female UBM')
165     start = time.time()
166     ubm_f = sidekit.Mixture()
167     llk_f = ubm_f.EM_split(features_server, ubmList_f, distribNb,
168                            num_thread=nbThread, save_partial=False)
169     ubm_f.write('gmm/female_diag_ubm_'+str(distribNb)+'.h5')
170     print('Training done!')
171     print(time.strftime('%c'))
172     end = time.time()
173     m, s = divmod(end-start, 60)
174     h, m = divmod(m, 60)
175     print("Time elapsed: "+str(round(h))+ 'H ' +str(round(m))+ 'M ' +str(round
(s))+ 'S')
176
177 #else:
178 #     raise ValueError("Error in settings. Please change settings by
179 #         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

```
1 # Runs a gmm-ubm system
2 # This system serves as a baseline together with the i-vector/PLDA script
3 # Differing from run_ubm_gmm_sid.py, this version of the file takes
   arguments
4
5 import sidekit
6 import os
7 import sys
8 import multiprocessing
9 import matplotlib.pyplot as mpl
10 import logging
11 import numpy as np
12 import time
13
14 logging.basicConfig(filename='log/ubm_gmm.log', level=logging.DEBUG)
15
16 if len(sys.argv) != 4:
17     print('Usage: '+sys.argv[0]+' <use more test data {0,1}> <number of
   enrollment wake words>\'
18         +\'<number of enrollment sentences>\'')
19     exit()
20
21 print('RUN GMM-UBM SPEAKER IDENTIFICATION TESTS')
22 print(time.strftime('%c'))
23
24 # Set parameters
25 gender = 'gdp' # 'gdp' 'gidp'
26 distribNb = 1024 # number of Gaussian distributions for each GMM
27 base_dir = '/home/studententer/jorgeja/Projects/master/'
28 feat_dir = base_dir+'data/feats/RSR2015/'
29 ubm_dir = base_dir+'ubm/gmm/gender_id_diag_ubm.h5'
30 selection_size = 1000 # number of utterances in ubm training set
31
32 ww_adapt_num = int(sys.argv[2]) # number of wakewords per speaker for
   enrollment data
33 stc_adapt_num = int(sys.argv[3]) # number of sentences per speaker for
   enrollment data
34 more_test_data = sys.argv[1] # decides length of test data (0: wakeword+
   command, 1: wakeword+sentence)
35
36 save_stats = False # if True, save sufficient statistics
37
38 print('- ww_adapt_num = '+str(ww_adapt_num))
39 print('- stc_adapt_num = '+str(stc_adapt_num))
40 print('- more_test_data = '+more_test_data)
41
42 # Automatically set the number of parallel process to run.
43 nbThread = max(multiprocessing.cpu_count()-1, 1)
44
45
46 # Load task definition
47 if stc_adapt_num == 3:
48     # Adaptation data: 3 wake-words and 3 sentences
49     enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/enroll_'+gender+'
   _50_t_norm_50.h5')
```



---

```

50 elif stc_adapt_num == 6:
51     # Adaptation data: 3 wake-words and 6 sentences
52     enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/more_data/enroll_'
53     +gender+'_50_t_norm_50_ww_3_stc_6.h5')
54 elif stc_adapt_num == 9:
55     # Adaptation data: 3 wake-words and 9 sentences
56     enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/more_data/enroll_'
57     +gender+'_50_t_norm_50_ww_3_stc_9.h5')
58 else:
59     raise ValueError('Enrollment IdMap file not found with stc_adapt_num =
60     '+str(stc_adapt_num))
61
62 if more_test_data == '0':
63     test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/eval_'+gender+'
64     _10_spk_100_sets_ndx.h5')
65     key = sidekit.Key(base_dir+'task/gmm_ubm/eval_'+gender+'
66     _10_spk_100_sets_key.h5')
67 elif more_test_data == '1':
68     test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/more_data/eval_'+gender+
69     '_10_spk_100_sets_more_data_ndx.h5')
70     key = sidekit.Key(base_dir+'task/gmm_ubm/more_data/eval_'+gender+'
71     _10_spk_100_sets_more_data_key.h5')
72 else:
73     raise ValueError('Invalid value of parameter <use more test data
74     {0,1}>: '+more_test_data)
75
76 # Create a FeaturesServer to load features and feed the other methods
77 features_server = sidekit.FeaturesServer(features_extractor=None,
78     feature_filename_structure=
79     feat_dir+"{}.h5",
80     sources=None,
81     dataset_list=["cep", "vad"],
82     mask=None,
83     feat_norm="cmvn",
84     global_cmvn=None,
85     dct_pca=False,
86     dct_pca_config=None,
87     sdc=False,
88     sdc_config=None,
89     delta=True,
90     double_delta=True,
91     delta_filter=None,
92     context=None,
93     traps_dct_nb=None,
94     rasta=True,
95     keep_all_features=False)
96
97 print('Load the UBM')
98 # Load ubm
99 ubm = sidekit.Mixture()
100 ubm.read(ubm_dir)
101
102 print('Compute the sufficient statistics')
103 # Create a StatServer for the enrollment data and compute the statistics
104 enroll_stat = sidekit.StatServer(enroll_idmap,

```

---

---

```

98         distrib_nb=distribNb,
99         feature_size=57)
100 enroll_stat.accumulate_stat(ubm=ubm,
101                             feature_server=features_server,
102                             seg_indices=range(enroll_stat.segset.shape[0])
103         ,
104         num_thread=nbThread)
105 if not os.path.exists('data/stat_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
106     ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5'):
107     enroll_stat.write('data/stat_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
108         ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
109
110 print('MAP adaptation of the speaker models')
111 regulation_factor = 3 # MAP regulation factor
112 enroll_sv = enroll_stat.adapt_mean_map_multisession(ubm, regulation_factor
113 )
114 if not os.path.exists('data/sv_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
115     ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5'):
116     enroll_sv.write('data/sv_gmm_ubm_enroll_'+gender+'_100_ww_'+str(
117         ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
118
119 print('Compute trial scores')
120 scores_gmm_ubm = sidekit.gmm_scoring(ubm,
121                                     enroll_sv,
122                                     test_ndx,
123                                     features_server,
124                                     num_thread=nbThread)
125 if more_test_data == '0':
126     scores_gmm_ubm.write('scores/scores_gmm_ubm_enroll_'+gender+'_100_ww_'+
127         str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
128 else:
129     scores_gmm_ubm.write('scores/more_test_data/scores_gmm_ubm_enroll_'+
130         gender+'_100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
131
132 print('Completed scoring')
133 print('')
134
135 # Analyzation of scores, including z-/t-norm, will be done in a separate
136 script

```

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

```

1 # This script contains definitions of scoring and normalization
2 # functions to be used in GMM-UBM system evaluations
3
4 import sidekit
5 import numpy as np
6 import copy
7
8
9 def get_second_largest(mylist):
10     # Returns the second largest element in a numpy.ndarray
11     newlist = np.delete(mylist, mylist.argmax())
12     return newlist.max()
13
14 def get_global_minmax(x_val, y_val, max_or_min='', log_y=False):
15     # Returns global max or global min of a numpy.ndarray

```

---

---

```

16 # x_val :      numpy.ndarray. The x-axis
17 # y_val :      numpy.ndarray. The y-axis
18 # max_or_min : string. Defines if global min or max should be returned
19 # log_y :      bool. If true, return logarithmic y-value
20 assert max_or_min == 'max' or max_or_min == 'min', 'max_or_min must be
    either \'max\' or \'min\''
21
22 if max_or_min == 'max':
23     x_max, y_max = [x_val[y_val.argmax()], y_val.max()]
24 else:
25     x_max, y_max = [x_val[y_val.argmin()], y_val.min()]
26
27 if log_y:
28     return x_max, np.log(y_val)
29 else:
30     return x_max, y_max
31
32 def normalization(orig_score, norm_mean, norm_std):
33     # Returns z- or t- (depending on input) normalized log-likelihood
    score
34     # orig_score : single llr-score from speaker verification test
35     # norm_mean :  estimated mean for normalization
36     # norm_std :   estimated standard deviation for normalization
37
38     # Return normalized score
39     return (orig_score-norm_mean)/norm_std
40
41 def get_mean_std(norm_scores):
42     # Returns estimated mean and standard deviation of elements in
    norm_scores
43     # mean: estimated mean of elements in norm_scores
44     # std: estimated standard deviation of elements in norm_scores
45
46     norm_scores_np = np.array(norm_scores)
47
48     return [norm_scores_np.mean(), norm_scores_np.std()]
49
50 def z_normalization(scoremat, z_norm_est):
51     # Returns a z-normalized matrix of scores
52     # scoremat:  matrix of scores from the Sidekit.bosaris.Scores object
53     # z_norm_est: an array of estimated zero normalization variables
54
55     assert len(z_norm_est) == scoremat.shape[0], 'scoremat and z_norm_est
    dimension mismatch'
56
57     scoremat_norm = copy.deepcopy(scoremat)
58     for i in range(scoremat.shape[0]):
59         if z_norm_est[i] != []:
60             for j in range(scoremat.shape[1]):
61                 scoremat_norm[i,j] = normalization(scoremat_norm[i,j],
    z_norm_est[i][0], z_norm_est[i][1])
62     return scoremat_norm
63
64 def t_normalization(scoremat, t_norm_est, enroll_range, norm_range):
65     # Returns a t-normalized matrix of scores
66     # scoremat :    matrix of scores from the Sidekit.bosaris.Scores
    object

```

---

---

```

67 # t_norm_est : an array of estimated t-normalization variables
68 # enroll_range : number of enrollment speakers (assumed equal for m/f)
69 # norm_range : number of normalization speakers (assumed equal for m
    /f)
70
71 assert len(t_norm_est) == scoremat.shape[1], 'scoremat and t_norm_est
    dimension mismatch'
72
73 scoremat_norm = copy.deepcopy(scoremat)
74 for i in range(scoremat.shape[1]):
75     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]:
76         scoremat_norm[j,i] = normalization(scoremat_norm[j,i],
            t_norm_est[i][0],t_norm_est[i][1])
77     return scoremat_norm
78
79
80 def estimate_z_norm_variables(enroll_range, norm_range, modelset, scoremat
    , norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m):
81     # Get z-norm variables for each model
82     z_norm_est = []
83     num_models, num_segments = scoremat.shape
84
85     for i in range(num_models):
86         if i < enroll_range or (i >= norm_range and i < norm_range+
            enroll_range): # z-norm is only applied to enrolled models
87             if modelset[i][0] == 'f':
88                 z_norm_est.append(get_mean_std(scoremat[i,norm_seg_idx_f:
                    enroll_seg_idx_m]))
89             elif modelset[i][0] == 'm':
90                 z_norm_est.append(get_mean_std(scoremat[i,norm_seg_idx_m:
                    num_segments]))
91             else:
92                 raise Exception('Unknown model ID in scores.modelset: '+
                    modelset[i])
93             else:
94                 z_norm_est.append([])
95     # finally, add the complementary model's z-norm variables
96     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])
97     return z_norm_est
98
99 def estimate_t_norm_variables(segset, scoremat, enroll_range, norm_range,
    modelset):
100     # Get t-norm variables for each enrollment speaker
101     # All trial segments are scored against a number of speaker models
102     # outside the enrollment set (the norm models),
103     # then the mean and standard deviation of these scores are calculated
104     # and used for score normalization
105     t_norm_est = []
106     num_models, num_segments = scoremat.shape
107
108     for i in range(num_segments):
109         # t-norm is applied to all trial segments (excluding normalization
            segments)
110         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
150 #                        score arrays
151 # enroll_seg_idx_m :    index of male enrollment segments
152 # norm_seg_idx_m :      index of male normalization segments
153 # Out:
154 # diff_to_next :        estimated mean distance between target and
155 #                        nontarget scores
156
157 subset_ratio = 5 # 1/subset_ratio is the relative size of the subset
158 # of the smallest distances in diff_vec
159 num_segments = len(score_target)
160
161 # Make array of differences between the two highest scores
162 diff_vec_tg = np.array([])
163 diff_vec_ntg = np.array([])
164 for i in [j for j in range(0, norm_seg_idx_f)] + [j for j in range(
165     enroll_seg_idx_m, norm_seg_idx_m)]:
166     # Target trials:
167     # Go through all scores, and pick only the cases where the target
168     # score is the largest one
169     if score_target[i].size != 0 and score_target[i].max() >
170     score_nontarget[i].max():
171         diff_vec_tg = np.append(diff_vec_tg, [score_target[i].max() -
172     score_nontarget[i].max()])
173     # Nontarget trials:
174     # Add all differences between best and second best score
175     if score_target[i].size == 0:
176         diff_vec_ntg = np.append(diff_vec_ntg, [score_nontarget[i].max
177     () - get_second_largest(score_nontarget[i])])
178
179 # Estimate mean difference from subset
180 diff_vec_tg = np.sort(diff_vec_tg)
181
182 #print(diff_vec_tg[0:100])
183 #print(diff_vec_ntg[0:100])
184 #print('Total number of highest target scores: '+str(len(diff_vec_tg))
185 )
186 #print('Scores included in estimation: '+str(len(diff_vec_tg)/
187 subset_ratio))
188
189 diff_mean = diff_vec_tg[0:len(diff_vec_tg)/subset_ratio].mean()
190
191 #fr_num = 0
192 #for diff in diff_vec_tg:
193 #    if diff < diff_mean:
194 #        fr_num += 1
195 #print('Number of false rejections: '+str(fr_num))
196
197 return diff_mean
198
199 def naive_decision(norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m,
200     score_target, score_nontarget):
201     # Performs naive decision, i.e. chooses the models that gives the
202     # highest score after a trial, and assigns the segment the identity
203     # of that model.
204     # In:

```

---

---

```

194 # norm_seg_idx_f : index of female normalization segments in the
    original score objects
195 # enroll_seg_idx_m : index of male enrollment segments
196 # norm_seg_idx_m : index of male normalization segments
197 # score_target : array of target scores
198 # score_nontarget : array of nontarget scores
199 # Out:
200 # rr : overall recognition rate
201 # far : false acceptance ratio
202 # frr : false rejection ratio
203 # ear : erroneous acceptance ratio
204
205 correct_trials = 0
206 correct_target_trials = 0
207 correct_nontarget_trials = 0
208 num_target_trials = 0 # total number of target trials
209 num_fa = 0 # number of false acceptations
210 num_fr = 0 # number of false rejections
211 num_ea = 0 # erroneous acceptance (speaker is classified as the wrong
    enrollement speaker)
212 num_trials = norm_seg_idx_f+norm_seg_idx_m-enroll_seg_idx_m
213
214 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)]:
215     # Unknown test speaker: Correct trial if complementary model has
    the highest score
216     if score_target[i].size == 0:
217         if score_nontarget[i].argmax() == len(score_nontarget[i])-1:
218             correct_trials += 1 # unknown speaker is rejected as it
    scores highest for comp. model
219             correct_nontarget_trials += 1
220         else:
221             num_fa += 1 # unknown speaker is falsely accepted as one
    of the enrollment speakers
222             # Known (enrolled) test speaker: Correct trial if the speakers'
    enrolled model
223             # has the highest score
224             else: # enrolled test speakers
225                 num_target_trials += 1
226                 if score_target[i].max() > score_nontarget[i].max():
227                     correct_trials += 1
228                     correct_target_trials += 1
229                 elif score_nontarget[i].argmax() == len(score_nontarget[i])-1:
230                     num_fr += 1 # known speaker is falsely rejected
231                 else:
232                     num_ea += 1 # known speaker is identified as the wrong
    known speaker
233
234 rr = correct_trials/float(num_trials)
235 far = num_fa/float(num_trials)
236 frr = num_fr/float(num_trials)
237 ear = num_ea/float(num_trials)
238
239 print('Number of correct target trials: '+str(correct_target_trials) \
240       + ' of '+str(num_target_trials)+' target trials' \
241       + '; ' + 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     # threshold :        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:

```

---



---

```

287         if score_target[i].max() > threshold and score_nontarget[i].
max() < threshold:
288             correct_trials += 1 # known speaker is assigned to its own
model
289         elif score_nontarget[i].argmax() != len(score_nontarget[i])-1
and score_nontarget[i].max() > threshold:
290             if get_second_largest(score_nontarget[i]) < threshold:
291                 num_ea += 1 # known speaker is assigned to the wrong
speaker model
292             else:
293                 num_fr += 1 # known speaker is falsely rejected
294         else:
295             num_fr += 1 # known speaker is falsely rejected
296
297     rr = correct_trials/float(num_trials)
298     far = num_fa/float(num_trials)
299     frr = num_fr/float(num_trials)
300     ear = num_ea/float(num_trials)
301
302     return rr, far, frr, ear
303
304 def threshold_decision(norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m,
score_target, score_nontarget, threshold, diff_to_next):
305     # Performs threshold decision, i.e. chooses the models that gives the
306     # highest score after a trial, and assigns the segment the identity
307     # of that model, if the score is above a set threshold value.
308     # It differs from threshold_decision_strict in the sense that trial
309     # scores are not rejected if
310     # there are more than one score above the threshold value. A score is
311     # rejected if the difference
312     # to the second best value is larger than diff_to_next
313     # In:
314     # norm_seg_idx_f :    index of female normalization segments in the
315     #                    original score object
316     # enroll_seg_idx_m : index of male enrollment segments
317     # norm_seg_idx_m :    index of male normalization segments
318     # score_target :      array of target scores
319     # score_nontarget :   array of nontarget scores
320     # threshold :         threshold value
321     # diff_to_next :      minimum acceptable difference between best and
322     #                    second best score
323     # Out:
324     # rr :                overall recognition rate
325     # far :               false acceptance ratio
326     # frr :               false rejection ratio
327     # ear :               erroneous acceptance ratio
328
329     correct_trials = 0
330     correct_target_trials = 0
331     correct_nontarget_trials = 0
332     num_fa = 0 # number of false acceptations
333     num_fr = 0 # number of false rejections
334     num_ea = 0 # erroneous acceptance (speaker is classified as the wrong
enrollment speaker)
335     num_trials = norm_seg_idx_f+norm_seg_idx_m-enroll_seg_idx_m
336
337     for i in [j for j in range(0,norm_seg_idx_f)]+[j for j in range(

```

---

---

```

335     enroll_seg_idx_m, norm_seg_idx_m)]:
336         # Unknown test speaker: Correct trial if complementary model has
the highest score
337         if score_target[i].size == 0:
338             if score_nontarget[i].argmax() == len(score_nontarget[i])-1:
339                 correct_trials += 1 # unknown speaker is assigned to comp.
models, i.e. rejected
340                 correct_nontarget_trials += 1
341             elif score_nontarget[i].max() < threshold:
342                 correct_trials += 1 # unknown speaker scores lower than
threshold for all models, rejection
343                 correct_nontarget_trials += 1
344             elif score_nontarget[i].max() - get_second_largest(
score_nontarget[i]) < diff_to_next:
345                 correct_trials += 1 # difference between best and second
best score too small, rejection
346                 correct_nontarget_trials += 1
347             else:
348                 num_fa += 1 # unknown speaker is falsely accepted as one
of the enrollment speakers
349                 # Known (enrolled) test speaker: Correct trial if the speakers'
enrolled model
350                 # has the highest score
351             else:
352                 if score_target[i].max() > threshold and score_target[i].max()
- score_nontarget[i].max() > diff_to_next:
353                     correct_trials += 1 # known speaker is assigned to its own
model
354                     correct_target_trials += 1
355                 elif score_nontarget[i].argmax() != len(score_nontarget[i])-1
and score_nontarget[i].max() > threshold:
356                     if score_nontarget[i].max() - get_second_largest(
score_nontarget[i]) > diff_to_next:
357                         num_ea += 1 # known speaker is assigned to the wrong
speaker model
358                     else:
359                         num_fr += 1 # known speaker is falsely rejected
360                 else:
361                     num_fr += 1 # known speaker is falsely rejected
362
363     rr = correct_trials/float(num_trials)
364     far = num_fa/float(num_trials)
365     frr = num_fr/float(num_trials)
366     ear = num_ea/float(num_trials)
367
368     return rr, far, frr, ear, correct_target_trials,
correct_nontarget_trials
369
370
371 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):
372     # Performs moving threshold decision
373     # In:
374     # norm_seg_idx_f :         index of female normalization segments in the
original score object
375     #

```

---

---

```

376 # enroll_seg_idx_m :      index of male enrollment segments
377 # norm_seg_idx_m :       index of male normalization segments
378 # score_target :         array of target scores
379 # score_nontarget :      array of nontarget scores
380 # scoremat :             scoremat from a Sidekit.bosaris.Scores object
381 # num_points :           resolution of output arrays
382 # additional_threshold : additional threshold so that all threshold
                           values are included
383 # Out:
384 # rr :                   array of overall recognition rate
385 # far :                  array of false acceptance ratio
386 # frr :                  array of false rejection ratio
387 # ear :                  array of erroneous acceptance ratio
388 # threshold_array :      threshold array for plots
389 # NOTE: if function get_raw_scores is edited to return numpy arrays,
390 # scoremat is not needed for finding max and min values of scores
391
392 # Deciding min and max values of threshold
393 threshold_min = min(scoremat[:, :norm_seg_idx_f].min(), scoremat[:,
enroll_seg_idx_m: norm_seg_idx_m].min())
394 threshold_max = max(scoremat[:, :norm_seg_idx_f].max(), scoremat[:,
enroll_seg_idx_m: norm_seg_idx_m].max()) + additional_threshold
395 # Initialize lists:
396 rr, far, frr, ear, tar_correct, nontar_correct = np.array([]), np.
array([]), np.array([]), np.array([]), np.array([]), np.array([])
397 threshold_array = np.linspace(threshold_min, threshold_max, num_points
)
398
399 # Estimate diff_to_next from score
400 diff_to_next = estimate_diff_to_next(score_target, score_nontarget,
norm_seg_idx_f, enroll_seg_idx_m, norm_seg_idx_m)
401
402 # Do decision for moving threshold
403 for threshold in threshold_array:
404     tmp_rr, tmp_far, tmp_frr, tmp_ear, correct_target_trials,
correct_nontarget_trials = threshold_decision(norm_seg_idx_f,
405
                                     enroll_seg_idx_m,
406
                                     norm_seg_idx_m,
407
                                     score_target,
408
                                     score_nontarget,
409
                                     threshold,
410
                                     diff_to_next)
411     rr = np.append(rr, tmp_rr)
412     far = np.append(far, tmp_far)
413     frr = np.append(frr, tmp_frr)
414     ear = np.append(ear, tmp_ear)
415     tar_correct = np.append(tar_correct, correct_target_trials)
416     nontar_correct = np.append(nontar_correct, correct_nontarget_trials
)
417
418

```

---

---

```

419 # Display error rates for target and nontarget trials
420 print('Maximum number of correct target trials: '+str(int(tar_correct.
max()))\
421      +' of 800 target trials; '+str(round(tar_correct.max()
/800*100,2))+ '%')
422 tar_max_idx = int(tar_correct.argmax())
423 print('Corresponding number of correct nontarget trials: '+str(int(
nontar_correct[tar_max_idx]))\
424      +' of 200 nontarget trials; '+str(round(nontar_correct[
tar_max_idx]/200*100,2))+ '%')
425
426
427 return rr, far, frr, ear, threshold_array

```

## GMM-UBM score evaluation: GMM-UBM/evaluate\_GMM-UBM.py

```

1 # This script evaluates scores from a GMM-UBM speaker identification
  experiment
2 # z-/t-/zt-normalization is applied to scores before evaluation
3 # It is also possible to exclude the complementary model from the
  evaluation
4 #
5 # The system performance is measured and displayed in two ways:
6 # - Maximum recognition rate. The maximum recognition rate, for a given
  value
7 #   of the decision value
8 # - A plot of recognition rate vs. decision value
9 #
10 # Further:
11 # - score_file contains the actual scores against enrollment models,
  normalization
12 #   speaker models and the complementary model (w)
13 # - key_file contains a mapping between speaker models and test segments
  scored
14 #   on them, with boolean matrices indicating whether the trials are
  target/nontarget
15 #
16 # Written by Joergen Antonsen, April 26 2017
17
18 import sidekit
19 import numpy
20 import os
21 import sys
22 import normalize_gmm_ubm
23 from normalize_gmm_ubm import get_mean_std
24 from normalize_gmm_ubm import naive_decision
25 from normalize_gmm_ubm import get_raw_scores
26 from normalize_gmm_ubm import estimate_z_norm_variables
27 from normalize_gmm_ubm import estimate_t_norm_variables
28 from normalize_gmm_ubm import z_normalization
29 from normalize_gmm_ubm import t_normalization
30 from normalize_gmm_ubm import moving_threshold_decision
31 from normalize_gmm_ubm import get_global_minmax
32 import matplotlib.pyplot as plt
33 import matplotlib.lines as mlines # for legend
34
35 assert len(sys.argv) == 5, 'Usage: '+sys.argv[0]+' <score_file_path>'+\

```

---

```

36     +' <number of enrollment wakewords>' \
37     +' <number of enrollment sentences>\n' \
38     +' <use more test data {0,1}>' \
39     +'Example: '+sys.argv[0]+' scores/ww_3_stc_3/
      scores_gmm_ubm_gdp_100_ww_3_sct_3.h5 3 3 1'
40
41 print('GMM-UBM SYSTEM EVALUATION')
42 print('- score file: '+sys.argv[1])
43
44 # -----Set parameters-----
45 test_set_size = 10 # number of speakers in the test set
46 num_unknown = 2 # number of unknown speakers in the test set
47 enroll_set_size = test_set_size-num_unknown # number of known speakers in
      test set
48 num_set_size = 100 # number of unique test sets
49
50 enroll_range = 50
51 norm_range = 100
52
53 ww_adapt_num = int(sys.argv[2]) # number of wake words for model
      enrollment
54 stc_adapt_num = int(sys.argv[3]) # number of sentences for model
      enrollment
55
56 score_file = sys.argv[1]
57
58 plot_fig = True # saves recognition rate plots as png if True
59 save_results = True # saves recognition rates as numpy arrays if True
60 more_test_data = sys.argv[4]
61
62 if more_test_data == '0':
63     result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num
        )
64     norm_score_dir = 'scores/normalized/ww_'+str(ww_adapt_num)+'_stc_'+str
        (stc_adapt_num)
65     fig_dir = 'fig/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
66 elif more_test_data == '1':
67     result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+
        str(stc_adapt_num)
68     norm_score_dir = 'scores/more_test_data/normalized/ww_'+str(
        ww_adapt_num)+'_stc_'+str(stc_adapt_num)
69     fig_dir = 'fig/more_test_data/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(
        stc_adapt_num)
70 else:
71     raise ValueError('Invalid value of parameter <use more test data
        {0,1}>: '+more_test_data)
72 if not os.path.exists(result_dir):
73     os.makedirs(result_dir)
74 if not os.path.exists(norm_score_dir):
75     os.makedirs(norm_score_dir)
76 if not os.path.exists(fig_dir):
77     os.makedirs(fig_dir)
78
79 # -----Read scores and test files-----
80 scores = sidekit.bosaris.Scores(score_file)
81 if more_test_data == '0':
82     key = sidekit.Key('../task/gmm_ubm/eval_gdp_10_spk_100_sets_key.h5')

```

---

---

```

83 elif more_test_data == '1':
84     key = sidekit.Key('../task/gmm_ubm/more_data/
      eval_gidp_10_spk_100_sets_more_data_key.h5')
85
86
87 # -----Manage variables and sort segment indices-----
88 modelset = scores.modelset # set of models (f001, f002, ..., m099, m100, w
      )
89 segset = scores.segset # set of segments
90 scoremask = scores.scoremask # score mask, True for element (i,j) if
      segment j is tested on model i
91 scoremat = scores.scoremat # score matrix, contain llr scores from all
      trials
92 target_mask = key.tar # matrix that is True for all target trials
93 nontarget_mask = key.non # matrix that is True for all nontarget trials
94
95 # Get segment indices
96 num_models, num_segments = scoremat.shape
97
98 enroll_seg_idx_f = 0 # segment index of female enrollment speakers
99 norm_seg_idx_f = 0 # segment index of female norm speakers (temp value)
100 enroll_seg_idx_m = 0 # segment index of male enrollment speakers (temp
      value)
101 norm_seg_idx_m = 0 # segment index of male norm speakers (temp value)
102 i = 0
103
104 while int(segset[i].split('/')[2][1:4]) <= enroll_range:
105     i += 1
106 norm_seg_idx_f = i
107
108 while segset[i].split('/')[2][0] != 'm':
109     i += 1
110 enroll_seg_idx_m = i
111
112 while int(segset[i].split('/')[2][1:4]) <= enroll_range:
113     i += 1
114 norm_seg_idx_m = i
115
116 # NOTE: The above code will not work unless segset elements have the
      format
117 #     <dir1>/<dir2>/<m,f>{001:xxx}<rest of filename>
118 #     e.g. cat/female/f001_01_032-049
119
120
121
122 # -----Estimate normalization variables-----
123 # Get z-norm variables for each model
124 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)
125
126 # Get t-norm variables for each enrollment speaker
127 t_norm_est = estimate_t_norm_variables(segset, scoremat, enroll_range,
      norm_range, modelset)
128
129
130 # -----Calculate recognition rates-----
131 # First get the 'raw', unprocessed scores

```

---

---

```

132 score_target_raw, score_nontarget_raw = get_raw_scores(scoremat,
    target_mask, nontarget_mask, enroll_range, norm_range)
133
134 # NAIVE DECISION - Choose the model that gives the highest score, without
    thresholding
135 print('NAIVE DECISION')
136 # 1: With raw scores
137 print('-----Using raw scores-----')
138 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)
139 print('Raw recognition rate: '+str(raw_recognition_rate*100)+'%')
140 print('False acceptance rate: '+str(raw_far*100)+'%')
141 print('False rejection rate: '+str(raw_frr*100)+'%')
142 print('Erroneous acceptance rate: '+str(raw_ear*100)+'%')
143
144 # 2: After z-norm
145 print('-----With z-normalization-----')
146 scoremat_z_norm = z_normalization(scoremat, z_norm_est)
147 score_target_z, score_nontarget_z = get_raw_scores(scoremat_z_norm,
    target_mask, nontarget_mask, enroll_range, norm_range)
148 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)
149 print('z-normalized recognition rate: '+str(z_recognition_rate*100)+'%')
150 print('False acceptance rate: '+str(z_far*100)+'%')
151 print('False rejection rate: '+str(z_frr*100)+'%')
152 print('Erroneous acceptance rate: '+str(z_ear*100)+'%')
153
154 # 3: After t-norm
155 print('-----With t-normalization-----')
156 scoremat_t_norm = t_normalization(scoremat, t_norm_est, enroll_range,
    norm_range)
157 score_target_t, score_nontarget_t = get_raw_scores(scoremat_t_norm,
    target_mask, nontarget_mask, enroll_range, norm_range)
158 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)
159 print('t-normalized recognition rate: '+str(t_recognition_rate*100)+'%')
160 print('False acceptance rate: '+str(t_far*100)+'%')
161 print('False rejection rate: '+str(t_frr*100)+'%')
162 print('Erroneous acceptance rate: '+str(t_ear*100)+'%')
163
164 # 4: After zt-norm
165 print('-----With zt-normalization-----')
166 scoremat_zt_norm = t_normalization(scoremat_z_norm, t_norm_est,
    enroll_range, norm_range)
167 score_target_zt, score_nontarget_zt = get_raw_scores(scoremat_zt_norm,
    target_mask, nontarget_mask, enroll_range, norm_range)
168 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)
169 print('zt-normalized recognition rate: '+str(zt_recognition_rate*100)+'%')
170 print('False acceptance rate: '+str(zt_far*100)+'%')
171 print('False rejection rate: '+str(zt_frr*100)+'%')
172 print('Erroneous acceptance rate: '+str(zt_ear*100)+'%')
173
174 # 5: After tz-norm
175 print('-----With tz-normalization-----')
176 scoremat_tz_norm = z_normalization(scoremat_t_norm, z_norm_est)

```

---

---

```

177 score_target_tz, score_nontarget_tz = get_raw_scores(scoremat_tz_norm,
    target_mask, nontarget_mask, enroll_range, norm_range)
178 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)
179 print('tz-normalized recognition rate: '+str(tz_recognition_rate*100)+'%')
180 print('False acceptance rate: '+str(tz_far*100)+'%')
181 print('False rejection rate: '+str(tz_frr*100)+'%')
182 print('Erroneous acceptance rate: '+str(tz_ear*100)+'%')
183
184 print('')
185
186
187
188 # DECISION WITH MOVING THRESHOLD - Choose model if score is solely above
    threshold
189 print('DECISION WITH MOVING THRESHOLD')
190
191 # 1: With raw scores
192 print('-----Using raw scores-----')
193 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)
194 print('Max recognition rate: '+str(raw_rr.max()*100)+'%')
195
196 # 2: With z-normalized scores
197 print('-----With z-normalization-----')
198 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)
199 print('Max recognition rate: '+str(raw_rr_z.max()*100)+'%')
200
201 # 3: With t-normalized scores
202 print('-----With t-normalization-----')
203 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)
204 print('Max recognition rate: '+str(raw_rr_t.max()*100)+'%')
205
206 # 4: With zt-normalized scores
207 print('-----With zt-normalization-----')
208 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)
209 print('Max recognition rate: '+str(raw_rr_zt.max()*100)+'%')
210
211 # 5: With tz-normalized scores
212 print('-----With tz-normalization-----')
213 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)
214 print('Max recognition rate: '+str(raw_rr_tz.max()*100)+'%')
215
216

```

---



---

```

217 # -----Save results-----
218 if save_results:
219     print('Saving results')
220     # 0: Score matrices from normalization
221     scores_norm = sidekit.bosaris.Scores()
222     scores_norm.scoremask = scoremask
223     scores_norm.modelset = modelset
224     scores_norm.segset = segset
225     # - z-norm scores
226     scores_norm.scoremat = scoremat_z_norm
227     scores_norm.write(os.path.join(norm_score_dir,
228                                     'scores_gmm_ubm_enroll_gidp_100_ww_{}
229                                     _stc_{}_z.h5'\
230                                     .format(str(ww_adapt_num), str(
231                                     stc_adapt_num))))
232     # - t-norm scores
233     scores_norm.scoremat = scoremat_t_norm
234     scores_norm.write(os.path.join(norm_score_dir,
235                                     'scores_gmm_ubm_enroll_gidp_100_ww_{}
236                                     _stc_{}_t.h5'\
237                                     .format(str(ww_adapt_num), str(
238                                     stc_adapt_num))))
239     # - zt-norm scores
240     scores_norm.scoremat = scoremat_zt_norm
241     scores_norm.write(os.path.join(norm_score_dir,
242                                     'scores_gmm_ubm_enroll_gidp_100_ww_{}
243                                     _stc_{}_zt.h5'\
244                                     .format(str(ww_adapt_num), str(
245                                     stc_adapt_num))))
246     # - tz-norm scores
247     scores_norm.scoremat = scoremat_tz_norm
248     scores_norm.write(os.path.join(norm_score_dir,
249                                     'scores_gmm_ubm_enroll_gidp_100_ww_{}
250                                     _stc_{}_tz.h5'\
251                                     .format(str(ww_adapt_num), str(
252                                     stc_adapt_num))))
253     print('- normalized score matrices saved')
254
255     # 1: Raw scores
256     numpy.save(os.path.join(result_dir, 'raw', 'rr'), raw_rr)
257     numpy.save(os.path.join(result_dir, 'raw', 'far'), raw_far)
258     numpy.save(os.path.join(result_dir, 'raw', 'frr'), raw_frr)
259     numpy.save(os.path.join(result_dir, 'raw', 'ear'), raw_ear)
260     numpy.save(os.path.join(result_dir, 'raw', 'xaxis'), raw_xaxis)
261     print('- raw results saved')
262
263     # 2: z-normalized scores
264     numpy.save(os.path.join(result_dir, 'z', 'rr'), raw_rr_z)
265     numpy.save(os.path.join(result_dir, 'z', 'far'), raw_far_z)
266     numpy.save(os.path.join(result_dir, 'z', 'frr'), raw_frr_z)
267     numpy.save(os.path.join(result_dir, 'z', 'ear'), raw_ear_z)
268     numpy.save(os.path.join(result_dir, 'z', 'xaxis'), raw_xaxis_z)
269     print('- z-normalized results saved')
270
271     # 3: t-normalized scores
272     numpy.save(os.path.join(result_dir, 't', 'rr'), raw_rr_t)

```

---

---

```

266 numpy.save(os.path.join(result_dir,'t','far'), raw_far_t)
267 numpy.save(os.path.join(result_dir,'t','frr'), raw_frr_t)
268 numpy.save(os.path.join(result_dir,'t','ear'), raw_ear_t)
269 numpy.save(os.path.join(result_dir,'t','xaxis'), raw_xaxis_t)
270 print('- t-normalized results saved')
271
272 # 4: zt-normalized scores
273 numpy.save(os.path.join(result_dir,'zt','rr'), raw_rr_zt)
274 numpy.save(os.path.join(result_dir,'zt','far'), raw_far_zt)
275 numpy.save(os.path.join(result_dir,'zt','frr'), raw_frr_zt)
276 numpy.save(os.path.join(result_dir,'zt','ear'), raw_ear_zt)
277 numpy.save(os.path.join(result_dir,'zt','xaxis'), raw_xaxis_zt)
278 print('- zt-normalized results saved')
279
280 # 5: tz-normalized scores
281 numpy.save(os.path.join(result_dir,'tz','rr'), raw_rr_tz)
282 numpy.save(os.path.join(result_dir,'tz','far'), raw_far_tz)
283 numpy.save(os.path.join(result_dir,'tz','frr'), raw_frr_tz)
284 numpy.save(os.path.join(result_dir,'tz','ear'), raw_ear_tz)
285 numpy.save(os.path.join(result_dir,'tz','xaxis'), raw_xaxis_tz)
286 print('- tz-normalized results saved')
287
288
289 # -----Plot recognition rate-----
290 # For moving threshold scoring
291 if plot_fig:
292     print('-----Save plots-----')
293     # -----Raw scores-----
294     fig = plt.figure()
295     ax = fig.add_subplot(111)
296     rr, = ax.plot(raw_xaxis,raw_rr,'r')
297     far, = ax.plot(raw_xaxis,raw_far,'g')
298     frr, = ax.plot(raw_xaxis,raw_frr,'b')
299     ear = ax.plot(raw_xaxis,raw_ear,'m')
300     red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
301     green_line = mlines.Line2D([],[],color='green',label='False acceptance
302         rate')
302     blue_line = mlines.Line2D([],[],color='blue',label='False rejection
303         rate')
303     mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
304         acceptance rate')
304     ax.legend(handles=[red_line,green_line,blue_line,mag_line])
305     ax.set_xlabel('Threshold')
306
307     # Annotate maxima/minima
308     rr_max_x, rr_max_y = get_global_minmax(raw_xaxis,raw_rr,'max')
309     ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
310         'data')
310     far_max_x, far_max_y = get_global_minmax(raw_xaxis,raw_far,'max')
311     ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
312         textcoords='data')
312     frr_max_x, frr_max_y = get_global_minmax(raw_xaxis,raw_frr,'min')
313     ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
314         textcoords='data')
314     ear_max_x, ear_max_y = get_global_minmax(raw_xaxis,raw_ear,'max')
315     ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
316         textcoords='data')

```

---

---

```

316 ax.set_title('Raw results')
317 fig.savefig(os.path.join(fig_dir,'rr_raw_gdp_'+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_gdp_'+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_gdp_'+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_gdp_'+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')

```

---

---

```

358 far, = ax.plot(raw_xaxis_t,raw_far_t,'g')
359 frr, = ax.plot(raw_xaxis_t,raw_frr_t,'b')
360 ear = ax.plot(raw_xaxis_t,raw_ear_t,'m')
361 red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
362 green_line = mlines.Line2D([],[],color='green',label='False acceptance
rate')
363 blue_line = mlines.Line2D([],[],color='blue',label='False rejection
rate')
364 mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
acceptance rate')
365 ax.legend(handles=[red_line,green_line,blue_line,mag_line])
366 ax.set_xlabel('Threshold')
367
368 # Annotate maxima/minima
369 rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_t,raw_rr_t,'max')
370 ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
'data')
371 far_max_x, far_max_y = get_global_minmax(raw_xaxis_t,raw_far_t,'max')
372 ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
textcoords='data')
373 frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_t,raw_frr_t,'min')
374 ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
textcoords='data')
375 ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_t,raw_ear_t,'max')
376 ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
textcoords='data')
377 ax.set_title('t-normalized results')
378 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'))
379 ax.set_yscale('log')
380 ax.set_title('t-normalized results - log')
381 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'))
382 print('- saved t-normalized scores')
383
384 # -----zt-normalized scores-----
385 fig = plt.figure()
386 ax = fig.add_subplot(111)
387 rr, = ax.plot(raw_xaxis_zt,raw_rr_zt,'r')
388 far, = ax.plot(raw_xaxis_zt,raw_far_zt,'g')
389 frr, = ax.plot(raw_xaxis_zt,raw_frr_zt,'b')
390 ear = ax.plot(raw_xaxis_zt,raw_ear_zt,'m')
391 red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
392 green_line = mlines.Line2D([],[],color='green',label='False acceptance
rate')
393 blue_line = mlines.Line2D([],[],color='blue',label='False rejection
rate')
394 mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
acceptance rate')
395 ax.legend(handles=[red_line,green_line,blue_line,mag_line])
396 ax.set_xlabel('Threshold')
397
398 # Annotate maxima/minima
399 rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_zt,raw_rr_zt,'max')
400 ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=

```

---

---

```

401     'data')
402     far_max_x, far_max_y = get_global_minmax(raw_xaxis_zt,raw_far_zt,'max'
403     )
404     ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
405     textcoords='data')
406     frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_zt,raw_frr_zt,'min'
407     )
408     ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
409     textcoords='data')
410     ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_zt,raw_ear_zt,'max'
411     )
412     ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
413     textcoords='data')
414     ax.set_title('zt-normalized results')
415     fig.savefig(os.path.join(fig_dir,'rr_zt_norm_gidp_'+str(num_set_size)+
416     '_sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
417     ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.png'))
418     ax.set_yscale('log')
419     ax.set_title('zt-normalized results - log')
420     fig.savefig(os.path.join(fig_dir,'rr_zt_norm_gidp_'+str(num_set_size)+
421     '_sets_'+str(test_set_size)+'_spk_'+str(test_set_size)+'_spk_ww_'+str(
422     ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_log.png'))
423     print('- saved zt-normalized scores')
424
425     # -----tz-normalized scores-----
426     fig = plt.figure()
427     ax = fig.add_subplot(111)
428     rr, = ax.plot(raw_xaxis_tz,raw_rr_tz,'r')
429     far, = ax.plot(raw_xaxis_tz,raw_far_tz,'g')
430     frr, = ax.plot(raw_xaxis_tz,raw_frr_tz,'b')
431     ear = ax.plot(raw_xaxis_tz,raw_ear_tz,'m')
432     red_line = mlines.Line2D([],[],color='red',label='Recognition rate')
433     green_line = mlines.Line2D([],[],color='green',label='False acceptance
434     rate')
435     blue_line = mlines.Line2D([],[],color='blue',label='False rejection
436     rate')
437     mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
438     acceptance rate')
439     ax.legend(handles=[red_line,green_line,blue_line,mag_line])
440     ax.set_xlabel('Threshold')
441
442     # Annotate maxima/minima
443     rr_max_x, rr_max_y = get_global_minmax(raw_xaxis_tz,raw_rr_tz,'max')
444     ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
445     'data')
446     far_max_x, far_max_y = get_global_minmax(raw_xaxis_tz,raw_far_tz,'max'
447     )
448     ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
449     textcoords='data')
450     frr_max_x, frr_max_y = get_global_minmax(raw_xaxis_tz,raw_frr_tz,'min'
451     )
452     ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
453     textcoords='data')
454     ear_max_x, ear_max_y = get_global_minmax(raw_xaxis_tz,raw_ear_tz,'max'
455     )
456     ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
457     textcoords='data')

```

---

---

```

437 ax.set_title('tz-normalized results')
438 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'))
439 ax.set_yscale('log')
440 ax.set_title('tz-normalized results - log')
441 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'))
442 print('- saved tz-normalized scores')
443
444
445
446 print('Done!')
```

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

```

1 # This script contains a few plot functions for presenting results
2
3 # Written by Joergen Antonsen May 16, 2017
4
5 import numpy
6 import os
7 from sidekit.bosaris import PlotWindow # for DET-plot
8 from sidekit.bosaris.detplot import __probit__ # for DET-plot
9 import matplotlib.pyplot as plt
10 import matplotlib.lines as mlines
11
12 # -----Set parameters-----
13 ww_adapt_num = 3 # number of wakeword adaptation utterances in enrollment
    data
14 stc_adapt_num = 3 # number of sentence adaptation utterances in enrollment
    data
15 more_test_data = True # decides length of test data (0: wakeword+command,
    1: wakeword+sentence)
16
17 if not more_test_data:
18     result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
    )
19     fig_dir = 'fig'
20 else:
21     result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+
    str(stc_adapt_num)
22     fig_dir = 'fig/more_test_data'
23
24 log_plot = False # if True plot with logarithmic y-axis
25
26 # Set to True if plotting recognition rate vs. enrollment data:
27 plot_rr = True
28 # Set to True if plotting recognition rate of known/unknown speakers:
29 plot_known_unknown = False
30 # Set to True if plotting DET-curve
31 plot_DET = False
32
33 # -----Load results-----
34 print('Loading results:')
35 raw_rr, raw_far, raw_frr, raw_ear, raw_xaxis = [], [], [], [], []
```

---

---

```

36 raw_rr_z, raw_far_z, raw_frr_z, raw_ear_z, raw_xaxis_z = [], [], [], [], []
37 raw_rr_t, raw_far_t, raw_frr_t, raw_ear_t, raw_xaxis_t = [], [], [], [], []
38 raw_rr_zt, raw_far_zt, raw_frr_zt, raw_ear_zt, raw_xaxis_zt =
    [], [], [], [], []
39 raw_rr_tz, raw_far_tz, raw_frr_tz, raw_ear_tz, raw_xaxis_tz =
    [], [], [], [], []
40
41 for stc_adapt_num in [3, 6, 9]:
42     if not more_test_data:
43         result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(
            stc_adapt_num)
44     else:
45         result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_
            '+str(stc_adapt_num)
46
47     print('stc_adapt_num = '+str(stc_adapt_num)+' :')
48     # 1: Raw scores
49     raw_rr.append(numpy.load(os.path.join(result_dir, 'raw', 'rr.npy')))
50     raw_far.append(numpy.load(os.path.join(result_dir, 'raw', 'far.npy')))
51     raw_frr.append(numpy.load(os.path.join(result_dir, 'raw', 'frr.npy')))
52     raw_ear.append(numpy.load(os.path.join(result_dir, 'raw', 'ear.npy')))
53     raw_xaxis.append(numpy.load(os.path.join(result_dir, 'raw', 'xaxis.npy')
        ))
54     print('- raw results loaded')
55
56     # 2: z-normalized scores
57     raw_rr_z.append(numpy.load(os.path.join(result_dir, 'z', 'rr.npy')))
58     raw_far_z.append(numpy.load(os.path.join(result_dir, 'z', 'far.npy')))
59     raw_frr_z.append(numpy.load(os.path.join(result_dir, 'z', 'frr.npy')))
60     raw_ear_z.append(numpy.load(os.path.join(result_dir, 'z', 'ear.npy')))
61     raw_xaxis_z.append(numpy.load(os.path.join(result_dir, 'z', 'xaxis.npy')
        ))
62     print('- z-normalized results loaded')
63
64     # 3: t-normalized scores
65     raw_rr_t.append(numpy.load(os.path.join(result_dir, 't', 'rr.npy')))
66     raw_far_t.append(numpy.load(os.path.join(result_dir, 't', 'far.npy')))
67     raw_frr_t.append(numpy.load(os.path.join(result_dir, 't', 'frr.npy')))
68     raw_ear_t.append(numpy.load(os.path.join(result_dir, 't', 'ear.npy')))
69     raw_xaxis_t.append(numpy.load(os.path.join(result_dir, 't', 'xaxis.npy')
        ))
70     print('- t-normalized results loaded')
71
72     # 4: zt-normalized scores
73     raw_rr_zt.append(numpy.load(os.path.join(result_dir, 'zt', 'rr.npy')))
74     raw_far_zt.append(numpy.load(os.path.join(result_dir, 'zt', 'far.npy')))
75     raw_frr_zt.append(numpy.load(os.path.join(result_dir, 'zt', 'frr.npy')))
76     raw_ear_zt.append(numpy.load(os.path.join(result_dir, 'zt', 'ear.npy')))
77     raw_xaxis_zt.append(numpy.load(os.path.join(result_dir, 'zt', 'xaxis.npy')
        ))
78     print('- zt-normalized results loaded')
79
80     # 5: tz-normalized scores
81     raw_rr_tz.append(numpy.load(os.path.join(result_dir, 'tz', 'rr.npy')))
82     raw_far_tz.append(numpy.load(os.path.join(result_dir, 'tz', 'far.npy')))
83     raw_frr_tz.append(numpy.load(os.path.join(result_dir, 'tz', 'frr.npy')))
84     raw_ear_tz.append(numpy.load(os.path.join(result_dir, 'tz', 'ear.npy')))

```

---

---

```

85     raw_axis_tz.append(numpy.load(os.path.join(result_dir,'tz','xaxis.npy
86     ')))
87     print('- tz-normalized results loaded')
88
89 # -----Plot-----
90 # Plot max recognition rate vs number of enrollment sentences
91 if plot_rr:
92     print('Plot max recognition rate vs number of enrollment sentences')
93     plot_all = False
94     fig = plt.figure()
95     if plot_all:
96         # Plots subplots for all systems, including plots of the error
97         # NOTE: NEEDS FIX TO WORK
98         for i in range(5): # iterate through all normalization types
99             rr = [raw_rr, raw_rr_z, raw_rr_t, raw_rr_zt, raw_rr_tz][i]
100             far = [raw_far, raw_far_z, raw_far_t, raw_far_zt, raw_far_tz][
101             i]
102             frr = [raw_frr, raw_frr_z, raw_frr_t, raw_frr_zt, raw_frr_tz][
103             i]
104             ear = [raw_ear, raw_ear_z, raw_ear_t, raw_ear_zt, raw_ear_tz][
105             i]
106             xaxis = [raw_xaxis, raw_xaxis_z, raw_xaxis_t, raw_xaxis_zt,
107             raw_xaxis_tz][i]
108             x_idx = [3,6,9] # indexes on x-axis (effectively, the number
109             of adaptation sentences)
110             plot_title = ['Raw results', 'z-normalized results', 't-
111             normalized results',\
112             'zt-normalized results', 'tz-normalized
113             results'][i]
114
115             # Find index of maximum recognition rate and store values of
116             all result
117             # arrays on these indices
118             max_rr, max_far, max_frr, max_ear = [],[],[],[]
119             for j in range(3): # iterate through all numbers of enrollment
120             sentences
121                 max_rr_idx = int(xaxis[j][int(rr[j].argmax())])
122                 max_rr.append(rr[j].max())
123                 max_far.append(far[j][max_rr_idx])
124                 max_frr.append(frr[j][max_rr_idx])
125                 max_ear.append(ear[j][max_rr_idx])
126
127             # Plotting starts here
128             ax = fig.add_subplot(5,1,i+1)
129             rr_plt, = ax.plot(x_idx,max_rr,'r')
130             far_plt, = ax.plot(x_idx,max_far,'g')
131             frr_plt, = ax.plot(x_idx,max_frr,'b')
132             ear_plt = ax.plot(x_idx,max_ear,'m')
133
134             ax.set_xlabel('Adaptation data')
135             ax.set_title(plot_title)
136             if log_plot:
137                 ax.set_yscale('log')
138                 ax.set_title(plot_title+' - log')
139

```

---



---

```

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,

```

---

---

```

175     more_test_data')
176     ax.set_ylim([30,85])
177     ax.grid()
178     red_line = mlines.Line2D([],[],color='red',label='Raw',marker='o')
179     green_line = mlines.Line2D([],[],color='green',label='z-norm',
180     marker='s')
181     blue_line = mlines.Line2D([],[],color='blue',label='t-norm',marker
182     ='^')
183     mag_line = mlines.Line2D([],[],color='magenta',label='zt-norm',
184     marker='v')
185     yellow_line = mlines.Line2D([],[],color='black',label='tz-norm',
186     marker='*')
187
188     # Shrink current axis's height by 10% on the bottom
189     box = ax.get_position()
190     ax.set_position([box.x0, box.y0 + box.height * 0.1,
191     box.width, box.height * 0.9])
192
193     # Put a legend below current axis
194     ax.legend(loc='upper center',
195     handles=[red_line,green_line,blue_line,mag_line,
196     yellow_line],
197     bbox_to_anchor=(0.5, -0.15),
198     fancybox=True,
199     shadow=True,
200     ncol=5)
201
202     if log_plot:
203         fig.savefig(os.path.join(fig_dir,'comparison','rr_gdp_ww_'+str(
204         ww_adapt_num)+'_stc_3_6_9_log.png'))#,
205         #bbox_inches='tight')
206     else:
207         fig.savefig(os.path.join(fig_dir,'comparison','rr_gdp_ww_'+str(
208         ww_adapt_num)+'_stc_3_6_9.png'))#,
209         #bbox_inches='tight')
210     print('- saved figure')
211
212 # Plot rate of recognized known and unknown speakers for each system
213 if plot_known_unknown:
214     print('Plot known/unknown speaker recognition rates:')
215     # Modified from https://matplotlib.org/examples/api/barchart_demo.html
216     # Input data from naive scoring and threshold scoring
217     # These numbers are obtained from 'log/evaluate_gmm_ubm_ww_3_stc_3.txt'
218     # Indices corresponds to ['raw','z-norm','t-norm','zt-norm','tz-norm']
219     if not more_test_data: # Numbers are obtained from 'log/
220     evaluate_gmm_ubm_ww_3_stc_3.txt'
221         kn_naive = [76.13, 73.25, 76.13, 73.25, 90.75]
222         un_naive = [75.0, 89.5, 75.0, 89.5, 16.5]
223         kn_thres = [68.88, 65.13, 68.38, 65.13, 82.13]
224         un_thres = [89.5, 93.5, 90.5, 93.5, 63.5]
225     else: # numbers are obtained from 'log/
226     evaluate_gmm_ubm_ww_3_stc_3_more_data.txt'
227         kn_naive = [65.13, 61.75, 65.13, 61.75, 90.63]
228         un_naive = [91.5, 97.0, 91.5, 97.0, 29.5]

```

---

---

```

221     kn_thres = [58.88, 55.38, 58.75, 56.25, 82.5]
222     un_thres = [95.5, 97.5, 95.5, 97.5, 73.0]
223
224     # 1: NAIVE DECISION
225     x_idx = numpy.arange(5)
226     y_idx = numpy.arange(0,101,10)
227     width = 0.25
228
229     fig, ax = plt.subplots()
230     rects_kn = ax.bar(x_idx, kn_naive, width, color='b')
231     rects_un = ax.bar(x_idx+width, un_naive, width, color='g')
232
233     ax.set_ylabel('Relative recognition rate [%]')
234     if not more_test_data:
235         ax.set_title('GMM-UBM: Known/unknown speaker trials, naive
236         decision', fontsize=10)
237     else:
238         ax.set_title('GMM-UBM: Known/unknown speaker trials, naive
239         decision, more test data', fontsize=10)
240     ax.set_xticks(x_idx+width/2)
241     ax.set_xticklabels(('raw', 'z-norm', 't-norm', 'zt-norm', 'tz-norm'))
242     ax.set_yticks(y_idx)
243     ax.set_ylim([0,100])
244
245     # Shrink current axis's height by 10% on the bottom
246     box = ax.get_position()
247     ax.set_position([box.x0, box.y0 + box.height * 0.1,
248                     box.width, box.height * 0.9])
249
250     # Put a legend below current axis
251     ax.legend((rects_kn[0], rects_un[0]), ('Known', 'Unknown'),
252             bbox_to_anchor=(0.75, -0.06),
253             fancybox=True,
254             shadow=True,
255             ncol=5)
256     ax.grid()
257     fig.savefig(os.path.join(fig_dir, 'comparison', '
258     gmm_ubm_known_unknown_rr_naive.png'))
259     print('- saved known/unknown recognition rate for naive decision')
260
261     # 2: THRESHOLD DECISION
262     x_idx = numpy.arange(5)
263     y_idx = numpy.arange(0,101,10)
264     width = 0.25
265
266     fig, ax = plt.subplots()
267     rects_kn = ax.bar(x_idx, kn_thres, width, color='b')
268     rects_un = ax.bar(x_idx+width, un_thres, width, color='g')
269
270     ax.set_ylabel('Relative recognition rate [%]')
271     if not more_test_data:
272         ax.set_title('GMM-UBM: Known/unknown speaker trials, threshold
273         decision', fontsize=10)
274     else:
275         ax.set_title('GMM-UBM: Known/unknown speaker trials, threshold
276         decision, more test data', fontsize=10)

```

---

---

```

273 ax.set_xticks(x_idx+width/2)
274 ax.set_xticklabels(('raw', 'z-norm', 't-norm', 'zt-norm', 'tz-norm'))
275 ax.set_yticks(y_idx)
276 ax.set_ylim([0,100])
277
278 # Shrink current axis's height by 10% on the bottom
279 box = ax.get_position()
280 ax.set_position([box.x0, box.y0 + box.height * 0.1,
281                 box.width, box.height * 0.9])
282
283 # Put a legend below current axis
284 ax.legend((rects_kn[0],rects_un[0]), ('Known', 'Unknown'),
285         bbox_to_anchor=(0.75, -0.06),
286         fancybox=True,
287         shadow=True,
288         ncol=5)
289 ax.grid()
290
291 fig.savefig(os.path.join(fig_dir, 'comparison',
292                          'gmm_ubm_known_unknown_rr_threshold.png'))
293 print('- saved known/unknown recognition rate for threshold decision')
294
295 # Plot DET-curve for all systems
296 # This is an evaluation of the systems as speaker verification systems,
297 # thus we define FAR as FAR+EAR in this plot
298 if plot_DET:
299     print('Note: Recommended to use ../tools/detplot_many.py')
300     print('Plot DET-curve')
301     # Determine enrollment data size
302     if stc_adapt_num == 3:
303         adapt_idx = 0
304     elif stc_adapt_num == 6:
305         adapt_idx = 1
306     else:
307         adapt_idx = 2
308
309     # Compute new FAR
310     raw_far_sum = raw_far[adapt_idx]+raw_ear[adapt_idx]
311     raw_far_sum_z = raw_far_z[adapt_idx]+raw_ear_z[adapt_idx]
312     raw_far_sum_t = raw_far_t[adapt_idx]+raw_ear_t[adapt_idx]
313     raw_far_sum_zt = raw_far_zt[adapt_idx]+raw_ear_zt[adapt_idx]
314     raw_far_sum_tz = raw_far_tz[adapt_idx]+raw_ear_tz[adapt_idx]
315
316     # Plot curves
317     fig = plt.figure()
318     ax = fig.add_subplot(111)
319     #ax.set_xscale('log')
320     #ax.set_yscale('log')
321     ax.plot(numpy.multiply(raw_frr[adapt_idx],100), numpy.multiply(
322         raw_far_sum,100))
323     ax.plot(numpy.multiply(raw_frr_z[adapt_idx],100), numpy.multiply(
324         raw_far_sum_z,100))
325     ax.plot(numpy.multiply(raw_frr_t[adapt_idx],100), numpy.multiply(
326         raw_far_sum_t,100))
327     ax.plot(numpy.multiply(raw_frr_zt[adapt_idx],100), numpy.multiply(
328         raw_far_sum_zt,100))

```

---

---

```

325 ax.plot(numpy.multiply(raw_frr_tz[adapt_idx],100), numpy.multiply(
    raw_far_sum_tz,100))
326
327 #yticks = numpy.array([0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 40, 80])
328 #xticks = numpy.array([10, 20, 40, 60, 80, 100])
329 #ax.set_yticks(yticks)
330 #ax.set_yticklabels(numpy.array(['0.1', '0.2', '0.5', '1', '2', '5',
    '10', '20', '40', '80']), size='x-small')
331 #ax.set_xticks(xticks)
332 #ax.set_xticklabels(numpy.array(['10', '20', '40','60','80','100']),
    size='x-small')
333
334 plt.title('GMM-UBM DET-curves')
335 plt.grid(True)
336 plt.xlabel('False rejection rate [%]')
337 plt.ylabel('False acceptance rate [%]')
338
339 fig.savefig(os.path.join(fig_dir,'sv','gmm_ubm_compared_wv_3_stc_'+str
    (stc_adapt_num)+' .png'))
340 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

```

1 # Runs the i-vector-based speaker identification experiment
2 # Serves as a baseline together with the gmm-ubm system
3 # Scoring is done using
4 #   1: Cosine-distance scoring
5 #   2: Mahalanobis' matrix
6 #   3: Two-covariance scoring
7 #   4: PLDA
8
9 import sidekit
10 import multiprocessing
11 import time
12 import logging
13 import sys
14 import os
15 import copy
16
17
18 if len(sys.argv) != 5:
19     print('Usage: '+sys.argv[0]+'\\
20         ' <use more test data {0,1}>'+'\\
21         ' <scoring method {1:cosine-distance,2:Mahalanobis,3:Two-
22         covariance,4:PLDA}>'+'\\
23         ' <number of enrollment wakewords>'+'\\
24         ' <number of enrollment sentences>')
25     exit()
26 logging.basicConfig(filename='log/ivec.log', level=logging.DEBUG)
27

```

---

```

28 print(time.strftime('%c'))
29 print('I-VECTOR-BASED SPEAKER IDENTIFICATION')
30
31 # Set parameters and filepaths
32 gender = 'gidp' # 'gdp' 'gidp'
33 distrib_nb = 1024 # number of Gaussian distributions for each GMM
34 feature_size = 57 # size of feature vector (mfcc+delta+deltadelta)
35 base_dir = '/home/student/jorgeja/Projects/master/'
36 feat_dir = base_dir+'data/feats/'
37 ubm_dir = base_dir+'ubm/gmm/gender_id_diag_ubm.h5'
38 selection_size = 1000 # number of utterances in ubm training set
39
40 ww_adapt_num = int(sys.argv[3]) # number of wakewords in enrollment data
41 stc_adapt_num = int(sys.argv[4]) # number of sentences in enrollment data
42 more_test_data = bool(int(sys.argv[1])) # decides length of test data(0:
    wakeword+command, 1: wakeword+sentence)
43
44 rank_TV = 400 # Rank of the total variability matrix
45 tv_iteration = 10 # number of iterations to run
46 plda_rk = 400 # rank of the PLDA eigenvoice matrix
47 nbThread = max(multiprocessing.cpu_count()-1, 1) # Number of parallel
    process to run
48
49 scoring_method = int(sys.argv[2]) # 1: cos, 2: Mah, 3: Two-cov, 4: PLDA (
    see top)
50 old_method = False # if False, use StatServer.factor_analysis(), else use
    FactorAnalyser()
51 do_TV_training = False # if False, load existing TV matrix
52 TV_path = 'data/TV__1024_it-10.h5' # path of TV matrix
53 do_PLDA_training = False # if False, load existing PLDA matrix
54 PLDA_path = 'data/PLDA_model_gidp_100.h5'
55
56 if do_TV_training:
57     # Get iteration number
58     assert len(sys.argv) == 2, 'Iteration number required for TV matrix
        training'
59     it_num_tv = int(sys.argv[1])
60     print('ITERATION NUMBER '+str(it_num_tv))
61
62 # Load task definition
63 # Enrollment data: 3 wakewords, 3 sentences:
64 if stc_adapt_num == 3:
65     enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/enroll_'+gender+'
        _50_t_norm_50.h5')
66 # Enrollment data: 3 wakewords, 6 or 9 sentences:
67 else:
68     enroll_idmap = sidekit.IdMap(base_dir+'task/gmm_ubm/more_data/enroll_'
        +gender+'_50_t_norm_50_ww_'+str(ww_adapt_num)\
69                                     +'_stc_'+str(stc_adapt_num)+'_h5')
70 if not more_test_data:
71     test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/eval_'+gender+'
        _10_spk_100_sets_ndx.h5')
72     keys = sidekit.Key(base_dir+'task/gmm_ubm/eval_'+gender+'
        _10_spk_100_sets_key.h5')
73     test_idmap = sidekit.IdMap(base_dir+'task/ivec/eval_'+gender+'
        _10_spk_100_sets_idmap.h5')
74 else:

```

---

---

```

75 test_ndx = sidekit.Ndx(base_dir+'task/gmm_ubm/more_data/eval_'+gender+
76 '_10_spk_100_sets_more_data_ndx.h5')
77 keys = sidekit.Key(base_dir+'task/gmm_ubm/more_data/eval_'+gender+'
78 '_10_spk_100_sets_more_data_key.h5')
79 test_idmap = sidekit.IdMap(base_dir+'task/ivec/eval_'+gender+'
80 '_10_spk_100_sets_more_data_idmap.h5')
81
82 tv_idmap = sidekit.IdMap(base_dir+"task/ivec/TV_idmap_"+gender+".h5")
83 plda_idmap = sidekit.IdMap(base_dir+"task/ivec/PLDA_idmap_"+gender+".h5")
84
85 # Create a FeaturesServer to load features and feed the other methods
86 fs_ldc = sidekit.FeaturesServer(features_extractor=None,
87                                 feature_filename_structure=feat_dir+"LDC/
88                                 new/{}.h5",
89                                 sources=None,
90                                 dataset_list=["cep", "vad"],
91                                 mask=None,
92                                 feat_norm="cmvn",
93                                 global_cmvn=None,
94                                 dct_pca=False,
95                                 dct_pca_config=None,
96                                 sdc=False,
97                                 sdc_config=None,
98                                 delta=True,
99                                 double_delta=True,
100                                delta_filter=None,
101                                context=None,
102                                traps_dct_nb=None,
103                                rasta=True,
104                                keep_all_features=False)
105
106 fs_enroll_test = sidekit.FeaturesServer(features_extractor=None,
107                                         feature_filename_structure=
108                                         feat_dir+"RSR2015/{}.h5",
109                                         sources=None,
110                                         dataset_list=["cep", "vad"],
111                                         mask=None,
112                                         feat_norm="cmvn",
113                                         global_cmvn=None,
114                                         dct_pca=False,
115                                         dct_pca_config=None,
116                                         sdc=False,
117                                         sdc_config=None,
118                                         delta=True,
119                                         double_delta=True,
120                                         delta_filter=None,
121                                         context=None,
122                                         traps_dct_nb=None,
123                                         rasta=True,
124                                         keep_all_features=False)
125
126 # Load ubm
127 print('Load the UBM')
128 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
    rank_h =
168
    None,

```

---



---

```

169     re_estimate_residual = False,
170     tv_iteration, 0, 0),
171     True,
172     ubm = ubm,
173     batch_size
174     = 100,
175     num_thread
176     = nbThread,
177     save_partial = "data/TV_{}".format(distrib_nb))
178     sidekit.sidekit_io.write_tv_hdf5((tv, tv_mean, tv_sigma), "data/
179     TV_{}".format(distrib_nb))
180     print('- done')
181     sys.stdout.flush() # print output
182     exit()
183
184 else: # Use FactorAnalyser
185     print('Train TV matrix using FactorAnalyser.total_variability()')
186     sys.stdout.flush() # print output
187
188     """ # You only need to do this once
189     tv_stat = sidekit.StatServer.read_subset('data/stat_back_'+gender
190     +'_100.h5', tv_idmap)
191     tv_stat.write('data/stat_tv_'+gender+'_100.h5')
192     print('- wrote TV_stat')
193     sys.stdout.flush()
194     """
195     for i in [it_num_tv]:
196
197         fa_load_TV = sidekit.FactorAnalyser('data/TV__1024_it-'+str(i
198         -1)+'_100.h5')
199         print('- loaded fa_load_TV')
200         sys.stdout.flush()
201
202         fa = sidekit.FactorAnalyser()
203         fa.total_variability('data/stat_tv_'+gender+'_100.h5', #
204         stat_server_filename must be a string, not an object
205         ubm,
206         rank_TV,
207         nb_iter=1, #tv_iteration,
208         min_div=True,
209         tv_init=fa_load_TV.F,
210         batch_size=100,
211         save_init=False,
212         output_file_name='data/TV_{}'.format(
213         distrib_nb),
214         num_thread=8) # Sidekit documentation
215         recommends a low number of processes for TV training
216         tv_mean = fa.mean
217         tv = fa.F
218         tv_sigma = fa.Sigma
219         os.rename('data/TV_1024.h5', 'data/TV__1024_it-'+str(i)+'_100.h5')
220
221         print(' iteration number '+str(i)+' done')

```

---

---

```

214         sys.stdout.flush() # print output
215         exit()
216
217     else:
218         print('Load TV matrix')
219         sys.stdout.flush() # print output
220         fa = sidekit.FactorAnalyser(TV_path)
221         tv_mean = fa.mean
222         tv = fa.F
223         tv_sigma = fa.Sigma
224
225
226     # Extract i-vectors
227     if old_method: # Extract i-vectors with StatServer.estimate_hidden()
228         print('Extract i-vectors using StatServer.estimate_hidden()')
229         enroll_iv = enroll_stat.estimate_hidden(tv_mean, tv_sigma, V=tv,
230         batch_size=100, num_thread=nbThread)[0]
231         enroll_iv.write('data/iv_enroll_'+gender+'_100.h5')
232         print('- enroll_iv done')
233
234         sys.stdout.flush()
235
236         test_iv = test_stat.estimate_hidden(tv_mean, tv_sigma, V=tv,
237         batch_size=100, num_thread=nbThread)[0]
238         test_iv.write('data/iv_test_'+gender+'_100.h5')
239         print('- test_iv done')
240
241         sys.stdout.flush()
242
243         plda_stat = sidekit.StatServer.read_subset('data/stat_back_'+gender+'
244         _100.h5', plda_idmap)
245         plda_iv = plda_stat.estimate_hidden(tv_mean, tv_sigma, V=tv,
246         batch_size=100, num_thread=nbThread)[0]
247         plda_iv.write('data/iv_plda_'+gender+'_100.h5')
248         print('- plda_iv done')
249
250         sys.stdout.flush()
251
252     else: # Extract i-vectors with FactorAnalyser.extract_ivectors()
253         print('Extract i-vectors using FactorAnalyser.extract_ivectors()')
254         """
255         enroll_iv = fa.extract_ivectors(ubm,
256         'data/stat_ivec_enroll_'+gender+'
257         _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5',
258         prefix='',
259         batch_size=100,
260         uncertainty=False,
261         num_thread=nbThread)
262         enroll_iv.write('data/iv_enroll_'+gender+'_100_ww_'+str(ww_adapt_num)
263         +'_stc_'+str(stc_adapt_num)+'.h5')
264         print('- enroll_iv done')
265
266         if not more_test_data:
267             test_iv = fa.extract_ivectors(ubm,
268             'data/stat_ivec_test_'+gender+'_100.
269             h5',
270             prefix='',

```

---

---

```

264                                     batch_size=100,
265                                     uncertainty=False,
266                                     num_thread=nbThread)
267     test_iv.write('data/iv_test_'+gender+'_100.h5')
268 else:
269     test_iv = fa.extract_ivectors(ubm,
270                                   'data/stat_ivec_test_'+gender+'
271                                   _100_more_data.h5',
272                                   prefix='',
273                                   batch_size=100,
274                                   uncertainty=False,
275                                   num_thread=nbThread)
276     test_iv.write('data/iv_test_'+gender+'_100_more_data.h5')
277     print('- test_iv done')
278
279     if not os.path.exists('data/stat_ivec_plda_'+gender+'_100.h5'):
280         plda_stat = sidekit.StatServer.read_subset('data/stat_TV_'+gender
281         +'_100.h5', plda_idmap) # TV stat file contains all PLDA stats
282         plda_stat.write('data/stat_ivec_plda_'+gender+'_100.h5')
283         print('- plda_stat done')
284
285     plda_iv = fa.extract_ivectors(ubm,
286                                   'data/stat_ivec_plda_'+gender+'_100.h5',
287                                   prefix='',
288                                   batch_size=100,
289                                   uncertainty=False,
290                                   num_thread=nbThread)
291     plda_iv.write('data/iv_plda_'+gender+'_100.h5')
292     print('- plda_iv done')
293     """
294
295 # -----RUN THE TESTS
296 -----
297 print('Loading i-vectors')
298 enroll_iv = sidekit.StatServer('data/iv_enroll_'+gender+'_100_ww_'+str(
299     ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'.h5')
300 if not more_test_data:
301     test_iv = sidekit.StatServer('data/iv_test_'+gender+'_100.h5')
302 else:
303     test_iv = sidekit.StatServer('data/iv_test_'+gender+'_100_more_data.h5
304     ')
305 plda_iv = sidekit.StatServer('data/iv_plda_'+gender+'_100.h5')
306
307 # 1: Using Cosine similarity
308 if scoring_method == 1:
309     print('Scoring with Cosine similarity: ')
310     scores_cos = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv,
311     test_ndx, wccn=None)
312     if not more_test_data:
313         scores_cos.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
314         ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos.h5')
315     else:
316         scores_cos.write('scores/more_test_data/scores_ivec_'+gender+'
317         _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos.h5')
318     print('- cos done')
319     sys.stdout.flush()

```

---

---

```

313 wccn = plda_iv.get_wccn_choleski_stat1()
314 scores_cos_wccn = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv
315 , test_ndx, wccn=wccn)
316 if not more_test_data:
317     scores_cos_wccn.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_cos_wccn.h5')
318 else:
319     scores_cos_wccn.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_wccn = 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_wccn.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_wccn.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])

```

---

---

```

358 test_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
359 M1 = plda_iv_efr1.get_mahalanobis_matrix_stat1()
360 scores_mah_efr1 = sidekit.iv_scoring.mahalanobis_scoring(
    enroll_iv_efr1, test_iv_efr1, test_ndx, M1)
361
362
363 # 3: Using Two-covariance scoring
364 elif scoring_method == 3:
365     raise Exception('Scoring with two-covariance not implemented yet')
366     W = plda_iv.get_within_covariance_stat1()
367     B = plda_iv.get_between_covariance_stat1()
368     scores_2cov = sidekit.iv_scoring.two_covariance_scoring(enroll_iv,
    test_iv, test_ndx, W, B)
369
370     meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')
371
372     plda_iv_sn1 = copy.deepcopy(plda_iv)
373     enroll_iv_sn1 = copy.deepcopy(enroll_iv)
374     test_iv_sn1 = copy.deepcopy(test_iv)
375
376     plda_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
377     enroll_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
378     test_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
379
380     W1 = plda_iv_sn1.get_within_covariance_stat1()
381     B1 = plda_iv_sn1.get_between_covariance_stat1()
382     scores_2cov_sn1 = sidekit.iv_scoring.two_covariance_scoring(
    enroll_iv_sn1, test_iv_sn1, test_ndx, W1, B1)
383
384
385 # 4: Using PLDA
386 elif scoring_method == 4:
387     print('Scoring with PLDA:')
388
389     # Spherical nuisance normalization
390     meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')
391     plda_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
392     enroll_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
393     test_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
394     print('- spherical nuisance normalization done')
395
396     if do_PLDA_training:
397         # Train PLDA matrix
398         if old_method: # Train PLDA matrix using StatServer.
399             factor_analysis()
400             print('Train PLDA matrix using StatServer.factor_analysis()
    ...')
401             plda_mean, plda_F, plda_G, plda_H, plda_Sigma = plda_iv.
    factor_analysis(rank_f=plda_rk,
402
403                 rank_g=0,
404
405                 rank_h=None,
406
407                 re_estimate_residual=True,
408
409                 it_nb=(10,0,0),

```

---

---

```

405         min_div=True,
406
407         ubm=None,
408
409         batch_size=100,
410
411         num_thread=nbThread)
412     print('- PLDA training done')
413     sidekit.sidekit_io.write_plda_hdf5((plda_mean, plda_F, plda_G,
414     plda_Sigma), "data/plda_model_"+gender+"_100.h5")
415     scores_plda = sidekit.iv_scoring.P LDA_scoring(enroll_iv,
416     test_iv, test_ndx, plda_mean, plda_F, plda_G, plda_Sigma, full_model=
417     False)
418     print('Scoring done')
419
420     else: # Train PLDA matrix using FactorAnalyser.plda()
421     print('Train PLDA matrix using FactorAnalyser.plda() ...')
422     sys.stdout.flush()
423     fa_plda = sidekit.FactorAnalyser()
424     fa_plda.plda(stat_server=plda_iv,
425     rank_f=plda_rk,
426     nb_iter=10,
427     scaling_factor=1,
428     output_file_name=PLDA_path.split('.')[0],
429     save_partial=True
430     )
431     print('- PLDA training done')
432     else:
433     print('- load PLDA matrix')
434     fa_plda = sidekit.FactorAnalyser(PLDA_path)
435     print('- perform PLDA scoring')
436     sys.stdout.flush()
437     scores_plda = sidekit.iv_scoring.fast_PLDA_scoring(enroll=
438     enroll_iv,
439
440     test=test_iv,
441     ndx=test_ndx,
442     mu=fa_plda.mean
443
444     ,
445
446     F=fa_plda.F,
447     Sigma=fa_plda.
448     Sigma)
449     print('-- done')
450
451     if not more_test_data:
452     scores_plda.write('scores/scores_ivec_'+gender+'_100_ww_'+str(
453     ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_plda.h5')
454     else:
455     scores_plda.write('scores/more_test_data/scores_ivec_'+gender+'
456     _100_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)+'_plda.h5')
457
458     # -----
459     # Analyzation of scores will be done in a separate script

```

---

---

## I-vector score evaluation: I-vectors/evaluate\_ivectors.py

```
1 # This script evaluates scores from a i-vector speaker identification
  # experiment
2 # z-/t-/zt-normalization is applied to scores before evaluation
3 # It is also possible to exclude the complementary model from the
  # evaluation
4 #
5 # The system performance is measured and displayed in two ways:
6 # - Maximum recognition rate. The maximum recognition rate, for a given
  #   value
7 #   of the decision value
8 # - A plot of recognition rate vs. decision value
9 #
10 # Further:
11 # - score_file contains the actual scores against enrollment models,
  #   normalization
12 #   speaker models and the complementary model (w)
13 # - key_file contains a mapping between speaker models and test segments
  #   scored
14 #   on them, with boolean matrices indicating whether the trials are
  #   target/nontarget
15 #
16 # Written by Joergen Antonsen, April 26 2017
17
18 import sidekit
19 import numpy
20 import os
21 import sys
22 sys.path.append('/home/student/jorgeja/Projects/master/gmm_ubm')
23 import normalize_gmm_ubm
24 from normalize_gmm_ubm import get_mean_std
25 from normalize_gmm_ubm import naive_decision
26 from normalize_gmm_ubm import get_raw_scores
27 from normalize_gmm_ubm import estimate_z_norm_variables
28 from normalize_gmm_ubm import estimate_t_norm_variables
29 from normalize_gmm_ubm import z_normalization
30 from normalize_gmm_ubm import t_normalization
31 from normalize_gmm_ubm import moving_threshold_decision
32 from normalize_gmm_ubm import get_global_minmax
33 import matplotlib.pyplot as plt
34 import matplotlib.lines as mlines # for legend
35
36 assert len(sys.argv) == 6, 'Usage: '+sys.argv[0]+' <score_file_path>'+\
37     +' <system suffix>'\
38     +' <number of enrollment wakewords>'\
39     +' <number of enrollment sentences>'\
40     +' <use more test data {0,1}>\n'\
41     +'Example: '+sys.argv[0]+' scores/ww_3_stc_3/\
42     scores_ivec_gidp_100_100_cos.h5 cos 3 3 1'
43 assert sys.argv[2] in ['cos', 'cos_lda', 'cos_wccn', 'cos_lda_wccn', 'plda'],\
44     'Invalid system suffix. Must be one of the following: cos, cos_lda,\
45     cos_wccn, cos_lda_wccn, cos_plda'
46
47 print('I-VECTOR SYSTEM EVALUATION')
48 print('- score file: '+sys.argv[1])
```

---

```

48 # -----Set parameters-----
49 test_set_size = 10 # number of speakers in the test set
50 num_unknown = 2 # number of unknown speakers in the test set
51 enroll_set_size = test_set_size-num_unknown # number of known speakers in
    test set
52 num_set_size = 100 # number of unique test sets
53
54 enroll_range = 50
55 norm_range = 100
56
57 ww_adapt_num = int(sys.argv[3]) # number of wakewords in enrollment data
58 stc_adapt_num = int(sys.argv[4]) # number of sentences in enrollment data
59
60 score_file = sys.argv[1]
61
62 plot_fig = True # saves recognition rate plots as png if True
63 sys_name = sys.argv[2] # name of system (for plot title)
64
65 save_results = True # if True, save recognition rates as numpy arrays
66 more_test_data = sys.argv[5]
67
68 if more_test_data == '0':
69     result_dir = 'results/ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num
    )
70     fig_dir = 'fig/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(stc_adapt_num)
71 elif more_test_data == '1':
72     result_dir = 'results/more_test_data/ww_'+str(ww_adapt_num)+'_stc_'+
    str(stc_adapt_num)
73     fig_dir = 'fig/more_test_data/sid_ww_'+str(ww_adapt_num)+'_stc_'+str(
    stc_adapt_num)
74 else:
75     raise ValueError('Invalid value of parameter <use more test data
    {0,1}>: '+more_test_data)
76 if not os.path.exists(result_dir):
77     os.makedirs(result_dir)
78 if not os.path.exists(fig_dir):
79     os.makedirs(fig_dir)
80
81 # -----Read scores and test files-----
82 scores = sidekit.bosaris.Scores(score_file)
83 if more_test_data == '0':
84     key = sidekit.Key('../task/gmm_ubm/eval_gidp_10_spk_100_sets_key.h5')
85 elif more_test_data == '1':
86     key = sidekit.Key('../task/gmm_ubm/more_data/
    eval_gidp_10_spk_100_sets_more_data_key.h5')
87
88 # -----Manage variables and sort segment indices-----
89 modelset = scores.modelset # set of models (f001, f002, ..., m099, m100, w
    )
90 segset = scores.segset # set of segments
91 scoremask = scores.scoremask # score mask, True for element (i,j) if
    segment j is tested on model i
92 scoremat = scores.scoremat # score matrix, contain llr scores from all
    trials
93 target_mask = key.tar # matrix that is True for all target trials
94 nontarget_mask = key.non # matrix that is True for all nontarget trials
95

```

---



---

```

96 # Get segment indices
97 num_models, num_segments = scoremat.shape
98
99 enroll_seg_idx_f = 0 # segment index of female enrollment speakers
100 norm_seg_idx_f = 0 # segment index of female norm speakers (temp value)
101 enroll_seg_idx_m = 0 # segment index of male enrollment speakers (temp
    value)
102 norm_seg_idx_m = 0 # segment index of male norm speakers (temp value)
103 i = 0
104
105 while int(segset[i].split('/')[2][1:4]) <= enroll_range:
106     i += 1
107 norm_seg_idx_f = i
108
109 while segset[i].split('/')[2][0] != 'm':
110     i += 1
111 enroll_seg_idx_m = i
112
113 while int(segset[i].split('/')[2][1:4]) <= enroll_range:
114     i += 1
115 norm_seg_idx_m = i
116
117 # NOTE: The above code will not work unless segset elements have the
    format
118 #     <dir1>/<dir2>/<m,f>{001:xxx}<rest of filename>
119 #     e.g. cat/female/f001_01_032-049
120
121
122 # -----Calculate recognition rates-----
123 # First get the 'raw', unprocessed scores
124 score_target_raw, score_nontarget_raw = get_raw_scores(scoremat,
    target_mask, nontarget_mask, enroll_range, norm_range)
125
126
127 # NAIVE DECISION - Choose the model that gives the highest score, without
    thresholding
128 print('NAIVE DECISION')
129 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)
130 print('Raw recognition rate: ' + str(raw_recognition_rate*100) + '%')
131 print('False acceptance rate: ' + str(raw_far*100) + '%')
132 print('False rejection rate: ' + str(raw_frr*100) + '%')
133 print('Erroneous acceptance rate: ' + str(raw_ear*100) + '%')
134 print('')
135
136
137 # DECISION WITH MOVING THRESHOLD - Choose model if score is solely above
    threshold
138 print('DECISION WITH MOVING THRESHOLD')
139 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)
140 print('Max raw recognition rate: ' + str(raw_rr.max()*100) + '%')
141
142
143 # -----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
172         rate')
173     blue_line = mlines.Line2D([],[],color='blue',label='False rejection
174         rate')
175     mag_line = mlines.Line2D([],[],color='magenta',label='Erroneous
176         acceptance rate')
177     ax.legend(handles=[red_line,green_line,blue_line,mag_line])
178     ax.set_xlabel('Threshold')
179
180     # Annotate maxima/minima
181     rr_max_x, rr_max_y = get_global_minmax(raw_xaxis,raw_rr,'max')
182     ax.annotate(str(rr_max_y*100)+'%', xy=(rr_max_x,rr_max_y), textcoords=
183         'data')
184     far_max_x, far_max_y = get_global_minmax(raw_xaxis,raw_far,'max')
185     ax.annotate(str(far_max_y*100)+'%', xy=(far_max_x,far_max_y),
186         textcoords='data')
187     frr_max_x, frr_max_y = get_global_minmax(raw_xaxis,raw_frr,'min')
188     ax.annotate(str(frr_max_y*100)+'%', xy=(frr_max_x,frr_max_y-0.05),
189         textcoords='data')
190     ear_max_x, ear_max_y = get_global_minmax(raw_xaxis,raw_ear,'max')
191     ax.annotate(str(ear_max_y*100)+'%', xy=(ear_max_x,ear_max_y),
192         textcoords='data')
193     ax.set_title(sys_title)
194     fig.savefig(os.path.join(fig_dir,'rr_raw'+score_file.split('.')[0].
195         split('/')[2]+sys_name+'.png'))
196     ax.set_yscale('log')
197     ax.set_title(sys_title+' - log')
198     fig.savefig(os.path.join(fig_dir,'rr_raw'+score_file.split('.')[0].
199         split('/')[2]+sys_name+'_log.png'))
200     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_
```

---

```

43     '+str(stc_adapt_num)
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     ))
48     cos_far.append(numpy.load(os.path.join(result_dir,'cos','raw','far.npy
49     ')))
50     cos_frr.append(numpy.load(os.path.join(result_dir,'cos','raw','frr.npy
51     ')))
52     cos_ear.append(numpy.load(os.path.join(result_dir,'cos','raw','ear.npy
53     ')))
54     cos_xaxis.append(numpy.load(os.path.join(result_dir,'cos','raw','xaxis
55     .npy')))
56     print('- cosine distance results loaded')
57
58     # 2: Cosine distance + lda scores
59     cos_lda_rr.append(numpy.load(os.path.join(result_dir,'cos_lda','raw','
60     rr.npy')))
61     cos_lda_far.append(numpy.load(os.path.join(result_dir,'cos_lda','raw',
62     'far.npy')))
63     cos_lda_frr.append(numpy.load(os.path.join(result_dir,'cos_lda','raw',
64     'frr.npy')))
65     cos_lda_ear.append(numpy.load(os.path.join(result_dir,'cos_lda','raw',
66     'ear.npy')))
67     cos_lda_xaxis.append(numpy.load(os.path.join(result_dir,'cos_lda','raw
68     ','xaxis.npy')))
69     print('- cosine distance + lda results loaded')
70
71     # 3: Cosine distance + wccn scores
72     cos_wccn_rr.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw'
73     ','rr.npy')))
74     cos_wccn_far.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw
75     ','far.npy')))
76     cos_wccn_frr.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw
77     ','frr.npy')))
78     cos_wccn_ear.append(numpy.load(os.path.join(result_dir,'cos_wccn','raw
79     ','ear.npy')))
80     cos_wccn_xaxis.append(numpy.load(os.path.join(result_dir,'cos_wccn','
81     raw','xaxis.npy')))
82     print('- cosine distance + wccn results loaded')
83
84     # 4: Cosine distance + lda + wccn scores
85     cos_lda_wccn_rr.append(numpy.load(os.path.join(result_dir,'
86     cos_lda_wccn','raw','rr.npy')))
87     cos_lda_wccn_far.append(numpy.load(os.path.join(result_dir,'
88     cos_lda_wccn','raw','far.npy')))
89     cos_lda_wccn_frr.append(numpy.load(os.path.join(result_dir,'
90     cos_lda_wccn','raw','frr.npy')))
91     cos_lda_wccn_ear.append(numpy.load(os.path.join(result_dir,'
92     cos_lda_wccn','raw','ear.npy')))
93     cos_lda_wccn_xaxis.append(numpy.load(os.path.join(result_dir,'
94     cos_lda_wccn','raw','xaxis.npy')))
95     print('- cosine distance + lda + wccn results loaded')
96
97     # 5: PLDA scores
98     plda_rr.append(numpy.load(os.path.join(result_dir,'plda','raw','rr.npy

```

---

---

```

    ')))
79     plda_far.append(numpy.load(os.path.join(result_dir, 'plda', 'raw', 'far.
        npy'))))
80     plda_frr.append(numpy.load(os.path.join(result_dir, 'plda', 'raw', 'frr.
        npy'))))
81     plda_ear.append(numpy.load(os.path.join(result_dir, 'plda', 'raw', 'ear.
        npy'))))
82     plda_xaxis.append(numpy.load(os.path.join(result_dir, 'plda', 'raw', '
        xaxis.npy'))))
83     print('- plda results loaded')
84
85
86 # -----Plot-----
87 # Plot max recognition rate vs number of enrollment sentences
88 plot_all = False
89 fig = plt.figure()
90 if plot_rr:
91     if plot_all:
92         # Plots subplots for all systems, including plots of the error
            rates
93         # NOTE: NEEDS FIX TO WORK
94         for i in range(5): # iterate through all normalization types
95             rr = [cos_rr, cos_lda_rr, cos_wccn_rr, cos_lda_wccn_rr,
                plda_rr][i]
96             far = [cos_far, cos_lda_far, cos_wccn_far, cos_lda_wccn_far,
                plda_far][i]
97             frr = [cos_frr, cos_lda_frr, cos_wccn_frr, cos_lda_wccn_frr,
                plda_frr][i]
98             ear = [cos_ear, cos_lda_ear, cos_wccn_ear, cos_lda_wccn_ear,
                plda_ear][i]
99             xaxis = [cos_xaxis, cos_lda_xaxis, cos_wccn_xaxis,
                cos_lda_wccn_xaxis, plda_xaxis][i]
100             x_idx = [3,6,9] # indexes on x-axis (effectively, the number
                of adaptation sentences)
101             plot_title = ['Cosine distance results',
                'cosine distance + lda results',
102                 'cosine distance + wccn results',
103                 'cosine distance + lda + wccn results',
104                 'plda results'][i]
105
106             # Find index of maximum recognition rate and store values of
                all result
107             # arrays on these indices
108             max_rr, max_far, max_frr, max_ear = [],[],[],[]
109             for j in range(3): # iterate through all numbers of enrollment
                sentences
110                 max_rr_idx = int(xaxis[j][int(rr[j].argmax())])
111                 max_rr.append(rr[j].max())
112                 max_far.append(far[j][max_rr_idx])
113                 max_frr.append(frr[j][max_rr_idx])
114                 max_ear.append(ear[j][max_rr_idx])
115
116             # Plotting starts here
117             ax = fig.add_subplot(5,1,i+1)
118             rr_plt, = ax.plot(x_idx,max_rr,'r')
119             far_plt, = ax.plot(x_idx,max_far,'g')
120             frr_plt, = ax.plot(x_idx,max_frr,'b')

```

---

---

```

122         ear_plt = ax.plot(x_idx, max_ear, 'm')
123
124         ax.set_xlabel('Adaptation data')
125         ax.set_title(plot_title)
126         if log_plot:
127             ax.set_yscale('log')
128             ax.set_title(plot_title+' - log')
129
130         red_line = mlines.Line2D([], [], color='red', label='Recognition
rate')
131         green_line = mlines.Line2D([], [], color='green', label='False
acceptance rate')
132         blue_line = mlines.Line2D([], [], color='blue', label='False
rejection rate')
133         mag_line = mlines.Line2D([], [], color='magenta', label='
Erroneous acceptance rate')
134         ax.legend(loc='bottom center', handles=[red_line, green_line,
blue_line, mag_line])
135
136         if log_plot:
137             fig.savefig('rr_gidp_ww_'+str(ww_adapt_num)+'
_stc_3_6_9_log.png')
138         else:
139             fig.savefig('rr_gidp_ww_'+str(ww_adapt_num)+'_stc_3_6_9.
png')
140         print('- saved figure')
141
142     else:
143         # Plot only recognition rate, not error rates
144         for i in range(5):
145             rr = [cos_rr, cos_lda_rr, cos_wccn_rr, cos_lda_wccn_rr,
plda_rr][i]
146             xaxis = [cos_xaxis, cos_lda_xaxis, cos_wccn_xaxis,
cos_lda_wccn_xaxis, plda_xaxis][i]
147             x_idx = [3, 6, 9] # indexes on x-axis (effectively, the number
of adaptation sentences)
148             plot_title = ['Cosine distance results',
'cosine distance + lda results',
'cosine distance + wccn results',
'cosine distance + lda + wccn results',
'plda results'][i]
149             color = ['r', 'g', 'b', 'm', 'k'][i]
150             marker = ['o', 's', '^', 'v', '*'][i]
151
152             # Find index of maximum recognition rate and store values of
all result
153             # arrays on these indices
154             max_rr, max_far, max_frr, max_ear = [], [], [], []
155             for j in range(3): # iterate through all numbers of enrollment
sentences
156                 max_rr.append(rr[j].max()*100)
157
158             # Plotting starts here
159             ax = fig.add_subplot(1, 1, 1)
160             rr_plt, = ax.plot(x_idx, max_rr, color, marker=marker)
161
162             ax.set_xlabel('Enrollment data [num. sentences]')

```

---

---

```

167         ax.set_ylabel('Recognition rate [%]')
168
169         if log_plot:
170             ax.set_yscale('log')
171             ax.set_title(plot_title+' - log')
172
173         if not more_test_data:
174             ax.set_title('I-vector recognition rate vs. enrollment data')
175         else:
176             ax.set_title('I-vector recognition rate vs. enrollment data,
177 more test data')
178             ax.set_ylim([30,85])
179             ax.grid()
180
181             red_line = mlines.Line2D([],[],color='red',label='cos',marker='o')
182             green_line = mlines.Line2D([],[],color='green',label='cos+lda',
183 marker='s')
184             blue_line = mlines.Line2D([],[],color='blue',label='cos+wccn',
185 marker='^')
186             mag_line = mlines.Line2D([],[],color='magenta',label='cos+lda+wccn
187 ',marker='v')
188             yellow_line = mlines.Line2D([],[],color='black',label='plda',
189 marker='*')
190
191             # Shrink current axis's height by 10% on the bottom
192             box = ax.get_position()
193             ax.set_position([box.x0, box.y0 + box.height * 0.1,
194                             box.width, box.height * 0.9])
195
196             # Put a legend below current axis
197             ax.legend(loc='upper center',
198                     handles=[red_line,green_line,blue_line,mag_line,
199 yellow_line],
200                     bbox_to_anchor=(0.5, -0.15),
201                     fancybox=True,
202                     shadow=True,
203                     ncol=5)
204
205             if log_plot:
206                 fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_'+
207 str(ww_adapt_num)+'_stc_3_6_9_log.png'))
208             else:
209                 fig.savefig(os.path.join(fig_dir,'comparison','rr_gidp_ww_'+
210 str(ww_adapt_num)+'_stc_3_6_9.png'))
211                 print('- saved figure')
212
213 # Plot rate of recognized known and unknown speakers for each system
214 if plot_known_unknown:
215     print('Plot known/unknown speaker recognition rates:')
216     # Modified from https://matplotlib.org/examples/api/barchart_demo.html
217     # Input data from naive scoring and threshold scoring
218     # These numbers are obtained from 'log/ww_3_stc_3/
219     evaluate_ivec_plda_ubm_ww_3_stc_3_{system}.txt'
220     # Indices corresponds to ['cos','cos_lda','cos_wccn','cos_lda_wccn','
221     cos_plda']
222     if not more_test_data:

```

---

---

```

214     kn_naive = [73.0, 60.0, 44.63, 58.13, 70.13]
215     un_naive = [3.5, 10.5, 0.0, 8.0, 10.0]
216     kn_thres = [65.63, 53.63, 40.5, 53.13, 62.5]
217     un_thres = [31.5, 33.0, 20.0, 26.5, 39.0]
218     else:
219         kn_naive = [69.13, 53.63, 42.5, 52.5, 72.25]
220         un_naive = [3.5, 6.0, 0.0, 3.0, 13.0]
221         kn_thres = [62.5, 47.88, 38.38, 46.38, 64.63]
222         un_thres = [33.5, 28.0, 19.0, 27.0, 36.5]
223
224     # 1: NAIVE DECISION
225     x_idx = numpy.arange(5)
226     y_idx = numpy.arange(0,101,10)
227     width = 0.25
228
229     fig, ax = plt.subplots()
230     rects_kn = ax.bar(x_idx, kn_naive, width, color='b')
231     rects_un = ax.bar(x_idx+width, un_naive, width, color='g')
232
233     ax.set_ylabel('Relative recognition rate [%]')
234     if not more_test_data:
235         ax.set_title('I-vector: Known/unknown speaker trials, naive
236         decision', fontsize=10)
237     else:
238         ax.set_title('I-vector: Known/unknown speaker trials, naive
239         decision, more test data', fontsize=10)
240     ax.set_xticks(x_idx+width/2)
241     ax.set_xticklabels(('cos', 'cos+lda', 'cos+wccn', 'cos+lda+wccn', 'plda'))
242     ax.set_yticks(y_idx)
243     ax.set_ylim([0,100])
244
245     # Shrink current axis's height by 10% on the bottom
246     box = ax.get_position()
247     ax.set_position([box.x0, box.y0 + box.height * 0.1,
248                     box.width, box.height * 0.9])
249
250     # Put a legend below current axis
251     ax.legend((rects_kn[0], rects_un[0]), ('Known', 'Unknown'),
252             bbox_to_anchor=(0.75, -0.06),
253             fancybox=True,
254             shadow=True,
255             ncol=5)
256     ax.grid()
257
258     fig.savefig(os.path.join(fig_dir, 'comparison',
259                             'ivec_known_unknown_rr_naive.png'))
260     print('- saved known/unknown recognition rate for naive decision')
261
262     # 2: THRESHOLD DECISION
263     x_idx = numpy.arange(5)
264     y_idx = numpy.arange(0,101,10)
265     width = 0.25
266
267     fig, ax = plt.subplots()
268     rects_kn = ax.bar(x_idx, kn_thres, width, color='b')
269     rects_un = ax.bar(x_idx+width, un_thres, width, color='g')

```

---



---

```

268 ax.set_ylabel('Relative recognition rate [%]')
269 if not more_test_data:
270     ax.set_title('I-vector: Known/unknown speaker trials, threshold
271     decision', fontsize=10)
272 else:
273     ax.set_title('I-vector: Known/unknown speaker trials, threshold
274     decision, more test data', fontsize=10)
275 ax.set_xticks(x_idx+width/2)
276 ax.set_xticklabels(('cos', 'cos+lda', 'cos+wccn', 'cos+lda+wccn', 'plda'))
277 ax.set_yticks(y_idx)
278 ax.set_ylim([0,100])
279
280 # Shrink current axis's height by 10% on the bottom
281 box = ax.get_position()
282 ax.set_position([box.x0, box.y0 + box.height * 0.1,
283                 box.width, box.height * 0.9])
284
285 # Put a legend below current axis
286 ax.legend((rects_kn[0], rects_un[0]), ('Known', 'Unknown'),
287          bbox_to_anchor=(0.75, -0.06),
288          fancybox=True,
289          shadow=True,
290          ncol=5)
291
292 ax.grid()
293
294 fig.savefig(os.path.join(fig_dir, 'comparison',
295                         'ivec_known_unknown_rr_threshold.png'))
296 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
8 tutorial.
9 Once all filter-banks features have been extracted, run this script to
10 train
11 a feed-forward deeep neural network including a bottleneck layer and
12 extract
13 all necessary features for the NIST-SRE10 evaluation
14 """
15
16 # Edited by J rgen Antonsen

```

---

```

15 # May 4, 2017
16
17 import sys
18 import os
19 import random
20 import numpy as np
21 #import pandas as pd
22 #pd.set_option('display.mpl_style', 'default')
23 from fnmatch import fnmatch
24 from collections import namedtuple
25 import pickle
26 import multiprocessing
27 import sidekit
28
29 # For Theano debugging
30 import theano
31 theano.config.optimizer='fast_compile'
32 theano.config.exception_verbosity='high'
33 #theano.config.compute_test_value = 'warn'
34
35 # Set parameters
36 nbThread = max(multiprocessing.cpu_count()-1,1) # Number of parallel
        processes (for feature extraction).
37
38 feature_root_dir = 'fisher_english/feats/'
39 feature_dim = 19 # number of mfcc coefficients
40 left_context = 7
41 right_context = 7
42 input_size = feature_dim * (left_context + 1 + right_context) # size of
        bottom layer of network
43
44 feature_extension = '.h5'
45 bnf_feature_extension = '.bnf'
46
47 train_dnn = True
48 extract_bnf = False
49
50 load_absolute_path = True
51
52
53 # Train DNN / Extract bnf
54 if train_dnn:
55     """ Train the DNN and save parameters to disk """
56
57     label_file_name = "fisher_english/ali_all.txt"
58     training_feature_dir = 'fisher_english/feats/'
59     show_prefix = 'LDC2004S13/fisher_eng_tr_sp/audio/'
60     feature_extension = '.h5'
61     absolute_file_path = 'fisher_english/ali_full_paths.txt'
62
63     # Before training you need a file with absolute paths (takes approx. a
        few hours to make)
64     if not load_absolute_path:
65         print('Make file with absolute paths (takes a few hours..)')
66         with open(absolute_file_path,'w') as fh_path:
67             # Make list of absolute paths of feature files
68             seg_paths = []

```

---

---

```

69         for root, dirs, files in os.walk(training_feature_dir):
70             for file in files:
71                 if file.endswith(feature_extension):
72                     seg_paths.append(os.path.join(root, file))
73
74             # Load alignment file
75             with open(label_file_name, 'r') as fh_ali:
76                 seg_list = fh_ali.read().split('\n')
77             if '' in seg_list:
78                 seg_list.remove('')
79
80             # Align absolute paths to files in ali-file
81             for segment in seg_list:
82                 for paths in seg_paths:
83                     if paths.endswith(segment.split('-')[0]+'-'+segment.
split('-')[1]+'.h5'):
84                         fh_path.write(paths+'\n')
85                         break
86             print('- done')
87
88             # Load the labels and get the number of output classes
89             print("Load training feature labels and their absolute paths")
90             sys.stdout.flush()
91             with open(label_file_name, 'r') as fh:
92                 seg_list = fh.read().split('\n')
93             seg_list.remove('')
94             with open(absolute_file_path, 'r') as fh:
95                 path_list = fh.read().split('\n')
96             path_list.remove('')
97             assert len(seg_list) == len(path_list), 'Segment list and path list
mismatch'
98
99             # ----- REMOVE -----
100            # Selection of 100 first lines, for debug
101            seg_list = seg_list[:100]
102            path_list = path_list[:100]
103            # --- REMOVE END ---
104
105            print('- processing contents..')
106            sys.stdout.flush()
107            nclasses = 0
108            for i in range(len(seg_list)): # split into ('segment id', 'channel',
'start', 'stop', 'alignments')
109                templ, ali = seg_list[i].split(' ')[0], seg_list[i].split(' ')[1:]
110                while '' in ali:
111                    ali.remove('')
112
113                ali = np.array([int(a)-1 for a in ali]) # extract 1 to make
classes start at 0, not 1
114                if ali == []:
115                    raise ValueError('Error in reading alignments, [] detected')
116
117                segment, channel, start, stop = templ.split('-')
118                if segment == '' or channel == '' or start == '' or stop == '':
119                    raise ValueError('Error in reading segment, channel, start and
stop values')
120

```

---

---

```

121     # Add segment and info to seg_list
122     seg_list[i] = (path_list[i], int(start), int(stop), ali)
123
124     if ali.max() > nclasses:
125         nclasses = ali.max()
126     nclasses += 1 # need to count class 0 as well
127
128     print("Number of output classes = {}".format(nclasses))
129
130
131     # Split list of segments into lists for training and cross-validation
132     print("Split the list of segments for training and cross-validation")
133     sys.stdout.flush()
134     idx = np.random.permutation(len(seg_list)).astype('int')
135     training_seg_list = [seg_list[ii] for ii in idx[:int(len(seg_list)
136     *0.9)]]
137     cv_seg_list = [seg_list[ii] for ii in idx[int(len(seg_list)*0.9):]]
138
139     # Free lists not needed anymore from memory
140     del seg_list, path_list
141
142     # Define the structure of the NN
143     print('Initialize network and FeaturesServer')
144     sys.stdout.flush()
145     FfNn = sidekit.FForwardNetwork(filename=None,
146                                     input_size=input_size,
147                                     input_mean=np.empty(input_size),
148                                     input_std=np.empty(input_size),
149                                     hidden_layer_sizes=(1200, 1200, 80,
150     1200, 1200),
151                                     layers_activations=("sigmoid", "sigmoid",
152     ", None, "sigmoid", "sigmoid", "softmax"),
153                                     n_classes=nclasses)
154
155     # Initialize FeaturesServer
156     features_server = sidekit.FeaturesServer(features_extractor=None,
157     feature_filename_structure=
158     training_feature_dir+"{}.h5",
159     sources=None,
160     dataset_list=["cep"], #, "vad"],
161     mask=None,
162     feat_norm="cmvn",
163     global_cmvn=None,
164     dct_pca=False,
165     dct_pca_config=None,
166     sdc=False,
167     sdc_config=None,
168     delta=False,
169     double_delta=False,
170     delta_filter=None,
171     context=(left_context,
172     right_context),
173     traps_dct_nb=None,
174     rasta=True,
175     keep_all_features=False)

```

---

---

```

173 # Start training of the network
174 print("Training the network on Fisher pt 1 ...")
175 sys.stdout.flush()
176 FfNn.train_acoustic(training_seg_list=training_seg_list,
177                    cross_validation_seg_list=cv_seg_list,
178                    features_server=features_server,
179                    feature_size=input_size,
180                    lr=0.008,
181                    segment_buffer_size=10, #200,
182                    batch_size=50, #512,
183                    max_iters=20,
184                    tolerance=0.003,
185                    output_file_name="BNF_network_fe_p1",
186                    save_tmp_nnet=True,
187                    traps=False,
188                    num_thread=nbThread)
189
190 """
191 # OLD METHOD
192 # Start training of the network
193 print("Training the network on Fisher pt 1 ...")
194 sys.stdout.flush()
195 FfNn.train(training_set=training_seg_list,
196           cross_validation_set=cv_seg_list,
197           lr = 0.008,
198           batch_size=512,
199           max_iters=20,
200           tolerance=0.003,
201           output_file_name="BNF_network_fe_p1",
202           save_tmp_nnet=True,
203           num_thread=nbThread)
204
205 if extract_bnf:
206     print(os.environ["THEANO_FLAGS"])
207
208     # Get the list of features to process
209     bnf_file_list = []
210     with open('sph_files_to_process.p', "rb" ) as f:
211         audio_file_list, feature_file_list = pickle.load (f)
212     for ff in feature_file_list:
213         possible_fname = [ff, ff + '_a', ff + '_b']
214         for fn in possible_fname:
215             if os.path.exists(os.path.join(feature_root_dir, fn +
216             fb_feature_extension)):
217                 bnf_file_list.append(fn)
218     print("{} bottleneck feature files to extract".format(len(
219     bnf_file_list)))
220
221     # Load NN parameters
222     FfNn = sidekit.FForwardNetwork(filename="BNF_network_epoch10.npz")
223
224     # Extract Bottleneck features using the first 3 bottom layers of the
225     network
226
227     # Run Bottleneck features computation in parallel on CPU
228     if "cpu" in os.environ["THEANO_FLAGS"]:
229         import multiprocessing

```

---

---

```

227     print("Split the list of files to process")
228     sub_lists = [list(l) for l in np.array_split(bnf_file_list,
nbThread)]
229
230     print("Extract Bottleneck features in parallel")
231     jobs = []
232     multiprocessing.freeze_support()
233     for idx in range(nbThread):
234         # Create argument list for each process
235         ff_args = {'layer_number':3,
236                   'feature_file_list':sub_lists[idx],
237                   'input_dir':feature_root_dir,
238                   'input_file_extension':".fb",
239                   'label_dir':feature_root_dir,
240                   'label_extension':".lbl",
241                   'output_dir':feature_root_dir+"bnf/",
242                   'output_file_extension':".bnf",
243                   'input_feature_format':"spro4",
244                   'output_feature_format':"spro4",
245                   'feature_context':(7, 7),
246                   'normalize_output':"cmvn",
247                   'log':None}
248
249         p = multiprocessing.Process(target=FfNn.feed_forward, kwargs=
ff_args)
250         jobs.append(p)
251         p.start()
252     for p in jobs:
253         p.join()
254
255     # Run Bottleneck features computation on a singleGPU
256     elif "gpu" in os.environ["THEANO_FLAGS"]:
257         FfNn.feed_forward(feature_file_list=bnf_file_list,
258                           layer_number=3,
259                           input_dir=feature_root_dir,
260                           input_file_extension=".fb",
261                           label_dir=feature_root_dir,
262                           label_extension=".lbl",
263                           output_dir=feature_root_dir+"bnf/",
264                           output_file_extension=".bnf",
265                           input_feature_format="spro4",
266                           output_feature_format="spro4",
267                           feature_context=(7, 7),
268                           normalize_output="cmvn",
269                           log=None)

```

## Extract bottleneck features: DNN/extract.bnf.py

```

1 # This script extracts bottleneck-features (bnf) from existing spectral
   features,
2 # storing them to feats/bnf/..
3 # A dnn (located in ../dnn) must be trained before running this script.
   Scripts
4 # for dnn training can be found in the same directory.
5
6 import sidekit
7 import os

```

---

---

```

8 import time
9
10 print(time.strftime('%c'))
11 print('BOTTLENECK-FEATURE EXTRACTION')
12
13 # Set parameters
14 dataset = 'LDC' # 'cat' 'RSR2015' 'LDC'
15 bn_layer = 3 # index of hidden layer to extract bnf from
16 feature_size = 19 # mfcc
17
18 print('Dataset: '+dataset)
19
20
21 # Set file paths
22 base_dir = '/home/studenter/jorgeja/Projects/master/'
23 dnn_path = os.path.join(base_dir, 'dnn/dnn.h5')
24 data_path = os.path.join(base_dir, 'data')
25 out_path = os.path.join(data_path, 'feats/bnf')
26
27 if dataset == 'cat':
28     feat_path = os.path.join(data_path, 'feats/RSR2015/cat')
29 elif dataset == 'RSR2015':
30     feat_path = os.path.join(data_path, 'feats/RSR2015')
31 elif dataset == 'LDC':
32     feat_path = os.path.join(data_path, 'feats/LDC/new')
33 else:
34     raise ValueError('Dataset named '+dataset+' does not exist.')
35 if not os.path.exists(feat_path):
36     raise ValueError('Feature path '+feat_path+' not found.')
37
38 if not os.path.exists(dnn_path):
39     raise ValueError('Dnn with path '+dnn_path+' not found.')
40
41
42 # Make file lists according to the inner structure of the hdf5 feature
   files
43 show_list = []
44 gender_list = ['male', 'female']
45 for gender in gender_list:
46     if dataset == 'RSR2015':
47         subdir_list = os.listdir(os.path.join(feat_path, gender))
48         for dirs in subdir_list:
49             temp_list = os.listdir(os.path.join(feat_path, gender, dirs))
50             temp_list = [os.path.join(gender, dirs, files.split('.')[0]) for
   files in temp_list]
51             show_list += temp_list
52     else: # dataset == 'cat' | 'LDC'
53         temp_list = os.listdir(os.path.join(feat_path, gender))
54         temp_list = [os.path.join(gender, files.split('.')[0]) for files in
   temp_list]
55         show_list += temp_list
56 if dataset == 'cat':
57     show_list = [os.path.join('cat', files) for files in show_list]
58     feat_path = os.path.join(data_path, 'feats/RSR2015')
59
60 print('Number of files: '+str(len(show_list)))
61

```

---

---

```

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

```