GMM per al processat de veu

Pol Delgado Martin
Daniel Villaplana Gomez

May 2014

1 Introducció

Les GMM ens permeten estimar la densitat de probabilitat, son una eina molt potentja que mitjançant un corpus de entrenament permeten entrenarles mitjançant un algoritme iteratiu com ho es el de expectation-maximitzation.
Així doncs si seleccionem bé les dades de entrenament podem crear classificadors molt potents , sent alhora computacionalment eficients, donat que nomes hem de calcular la probabilitat de que la nostre caracteristica pertanyi a un dels nostres models.

En aquest projecte farem servir les GMM per a la classificació de locutor, es a dir dir a qui pertany una grabació donat un senyal (o al menys a qui es probable que pertanyi).
També usarem aquesta eina per a la verificació de veu, es a dir identificar a un usuari mitjançant unicament una grabació.

2 Analisis

Per a la correcta implementació dels nostres proposits hem implementat una classe GMM, a més hem creat tambe les funcions gmm_classify i gmm_verify.

gmm_classify ens permet a partir de unes dades i un array de GMM (els nostres locutors) dir probablement de qui és la grabació.

gmm_verify ens permet a partir de unes dades i un array de GMM identificar un locutor,la diferencia principal entre classify i verify és que en verificació no podem només donar la opcio més probable, un intrus es inaceptable, aixi doncs creem una gmm background , un model del conjunt de locutors, i la comparem amb un treshold, aixi doncs un locutor no només ha de semblarse a ell mateix sino que ha de ser identificable, distingirse del conjunt de parlants.

Per a la extracció de caracteristiques dels audios farem servir el SPTK, un conjunts de programes per al processat de veu, que conté funcions com la de calcular els mpcc els lpcc o enfinestraments.

A més a fi de probar i dissenyar el nostre sistema ens caldra un conjunt de audios prou ampli com per modelar un cas real, farem servir la base de dades speecon.

Així doncs hem realitzat les seguents tasques:

- Completar la classe GMM
- Completar la funció gmm_train , que és una implementacio del algoritme Expectation-Maximization per a entrenar les GMM.
- finalitzar la funció gmm_classify, que ens permetra decidir entre els locutors.
- S'analitzarà l'importància dels paràmetres.
  - Mètodes d'inicialització.
  - Nombre de Gaussianes.
  - lpcc i mpc.
- Utilitzarem programació bash o els scripts per a processos complexes i repetitius.

3 Síntesis

3.1 gmm.cpp

```
/* Copyright (C) Universitat Politcnica de Catalunya,
   Barcelona, Spain.
 *
 * Permission to copy, use, modify, sell and distribute
     this software
 * is granted provided this copyright notice appears in
     all copies.
 * This software is provided ”as is” without express or
     implied
 * warranty, and with no claim as to its suitability for
     any purpose.
 * antonio.bonafonte@upc.edu
 * Barcelona, November 2011
 */
```

```
#include <fstream>
#include "gmm.h"

using namespace std;
using namespace upc;


namespace upc {

#define CTTE_GAUSSIAN 0.398942280401433 /* 1/sqrt(2 * PI)
     */

/*
   Compute log(x+y) from logx and logy.
   The basic expression,
        log(e^logx + e^logy),
   may result in 'nan', if e^logx or e^logy is too large

   => if logx is larger,
   log(e^logx + e^logy)=log(e^logx * (1+e^logy/e^logx)) =
        logx +log(1+e^(logy-logx))
 */

float add_logs(float logx, float logy) {
        if (logx > logy)
                return logx + log(1.0 + exp(logy-logx));
        else
                return logy + log(1.0 + exp(logx-logy));
}


/*
  // Working directly with probabilities can give
      numerical problems;
  // It is better to work with logprob (see below)

  float gaussian_prob(unsigned int vector_size, const
      float *mu, const float *inv_sigma, const float *x) {
    float e=0.0F, c=1.0F, p;
    unsigned int j;
    for (j=0; j<vector_size; ++j) {
      float f = (x[j]-mu[j]) * inv_sigma[j];
      e += (f*f);
      c *= CTTE_GAUSSIAN * inv_sigma[j];
    }
    e /= 2;
    p = c * exp(-e);
    return p;
  }
 */

float gaussian_logprob(unsigned int vector_size, const
    float *mu, const float *inv_sigma, const float *x) {
        float e=0.0F, c=0.0F, logp;
        unsigned int j;
        for (j=0; j<vector_size; ++j) {
                float f = (x[j]-mu[j]) * inv_sigma[j];
                e += (f*f);
                c += log(inv_sigma[j]);
        }
        e /= 2;
        logp = vector_size * CTTE_GAUSSIAN + c - e;
        return logp;
}

void GMM::delete_mixture(unsigned int k) {
        if (k >= nmix)
                return;

        unsigned int last = nmix-1;

        if (k != last) {
                //save last mixture in position k
                w[k] = w[last];
                unsigned int i;
                for (i=0; i<vector_size; ++i) {
                        mu[k][i] = mu[last][i];
                        inv_sigma[k][i] = inv_sigma[last
                            ][i];
                }
        }
        resize(nmix-1, vector_size);
}


///Compute logprob of the input data
float GMM::gmm_logprob(const float *x) const {
        float log_prob_x,f;
        unsigned int k;
        log_prob_x = log(w[0]) + gaussian_logprob(
            vector_size, mu[0], inv_sigma[0], x);
        for (k=1; k<nmix; ++k) {
                f = log(w[k]) +  gaussian_logprob(
                    vector_size, mu[k], inv_sigma[k], x);
                log_prob_x = add_logs(log_prob_x, f);
        }
        return log_prob_x;
}

///TODO: Compute logprob a sequence of input data
float GMM::logprob(const fmatrix &data) const {

        if (nmix == 0 || vector_size == 0 || vector_size
            != data.ncol())
                return -1e38F;

        float lprob = 0.0;
        unsigned int n;

        for (n=0; n<data.nrow(); ++n) {
                //         TODO
                lprob+=gmm_logprob(data[n]);

        }
        return lprob/n;
}


int GMM::centroid(const upc::fmatrix &data) {
        if (data.nrow() == 0 || data.ncol() == 0)
                return -1;

        resize(1, data.ncol());

        unsigned int n;
        fmatrix weights(data.nrow(), 1);

        for (n=0; n < weights.nrow(); ++n)
                weights[n][0] = 1.0F;
        em_maximization(data, weights);
        return 0;
}


///Compute the best mixtures (weights, means, variances)
    given
/// -the input data
/// -the weights that the input data is generated for each gaussian

int GMM::em_maximization(const upc::fmatrix &data, const upc::fmatrix &weights) {
        unsigned int n, j, k;

        w.reset();
        mu.reset();
        inv_sigma.reset();

        for (n=0; n<data.nrow(); ++n) {
                for (k=0; k < nmix; ++k) {
                        w[k] +=  weights[n][k];
                        for (j=0; j < vector_size; ++j) {
                                mu[k][j] += weights[n][k] * data[n][j]; /* sum{x w_i} */
                                inv_sigma[k][j] += weights[n][k] * data[n][j] * data[n][j]; /* sum{x^2 w_i} */
                        }
                }
        }
        for (k=0; k < nmix; ++k) {
                for (j=0; j < vector_size; ++j) {
                        mu[k][j] /= w[k]; /* sum{x w_i}/sum{w_i} */
                        inv_sigma[k][j] /= w[k]; /* sum{x^2 w_i}/sum{w_i} */
                        inv_sigma[k][j] = 1.0F/sqrt(inv_sigma[k][j] - mu[k][j]*mu[k][j]); /* 1/sigma */
                }
                w[k] /=  data.nrow();
        }
        return 0;
}

///For each input data compute the probability that the data is generated from each mixture
///We work with log of probabilities to avoid numerical problems on intermediate results
```

```
float GMM::em_expectation(const fmatrix &data, fmatrix &
    weights) const {
        unsigned int n, k;
        float log_prob_total, log_prob_x;

        if (data.ncol() != vector_size)
                return -1.0;

        if (weights.nrow() != data.nrow() ||
                        weights.ncol() != nmix)
                weights.resize(data.nrow(), nmix);

        //use log(prob) for intermediate computation, to
            avoid underflow

        //For each input data ...
        for (n=0, log_prob_total = 0.0F; n<data.nrow();
            ++n) {
                //For each mixture ...
                for (k=0, log_prob_x = -1e20F; k < nmix;
                    ++k) {
                        weights[n][k] = log(w[k]) +
                            gaussian_logprob(vector_size,
                            mu[k], inv_sigma[k], data[n]);
                        log_prob_x = add_logs(log_prob_x,
                             weights[n][k]);
                }

                for (k=0; k < nmix; ++k)
                        weights[n][k] = exp(weights[n][k
                           ]-log_prob_x);
                log_prob_total += log_prob_x;
        }

        log_prob_total /= data.nrow();
        return log_prob_total;
}

int GMM::em(const fmatrix &data, unsigned int max_it,
    float inc_threshold, int verbose) {
        unsigned int iteration;
        float old_prob=0.0F, new_prob=0.0F, inc_prob=1.0F
            ;

        fmatrix weights(data.nrow(), nmix);
        for (iteration=0; iteration<max_it; ++iteration)
           {

                ///TODO: loop, em_expectation +
                   em_maximization
                ///Stop if the prob. does not increases
                   more than inc_threshold
                ///Update old_prob, new_prob, inc_prob
                em_expectation(data,weights);
                em_maximization(data,weights);

                new_prob=logprob(data);

                inc_prob=new_prob-old_prob;
                old_prob=new_prob;
                if(inc_prob<inc_threshold){break;}
                if (verbose & 01)
                        cout << "GMM nmix=" << nmix << "\
                           tite=" << iteration << "\tlog(
                           prob)=" << new_prob << "\tinc=
                           " << inc_prob << endl;

        }
        return 0;
}

int GMM::em_split(const fmatrix &data, unsigned int
   final_nmix, unsigned int max_it, float inc_threshold,
   int verbose) {
        centroid(data);
        while (nmix <final_nmix) {
                split(final_nmix);
                em(data, max_it, inc_threshold, verbose);
        }
        return 0;
}

void GMM::split_mixture(unsigned int src, unsigned int
   dest) {
        unsigned int j;
        int sign;
        float r;

        for (j=0; j<vector_size; ++j) {
                r = (float) 2.0F *rand()/(float) RAND_MAX
                    - 1.0F; /* r: (-1,1) */
                sign = (r > 0 ? 1 : -1);

                mu[dest][j] = mu[src][j] + sign * 0.5/
                    inv_sigma[src][j];
                mu[src][j] = mu[src][j] - sign * 0.5/
                    inv_sigma[src][j];

                inv_sigma[src][j] *= 2;
                inv_sigma[dest][j] = inv_sigma[src][j];

        }
        w[src] /= 2.0F;
        w[dest] = w[src];
}

int GMM::split(unsigned int target_size) {
        unsigned int i, j, old_size;

        if (nmix >= target_size)
                return nmix;

        if (2*nmix <= target_size) {
                target_size = 2*nmix;
                old_size = nmix;
                resize(2*nmix, vector_size);
                for (i=old_size, j=0; i<nmix; ++i, ++j)
                        split_mixture(j, i);
        } else {
                old_size = nmix;
                resize(target_size, vector_size);
                /* TO DO: select mixtures with larger
                    variance (now, the first ones) */
                for (i=old_size, j=0; i<nmix; ++i, ++j)
                        split_mixture(j, i);
        }
        return nmix;
}

int GMM::random_init(const upc::fmatrix &data, unsigned
    int nmix) {
        if (data.nrow() == 0 || data.ncol() == 0)
                return -1;
        resize(nmix, data.ncol());

        unsigned int n, k;
        fmatrix weights(data.nrow(), nmix);
        weights.reset();
        for (n=0; n < data.nrow(); ++n) {
                float r = (float) rand()/(float) RAND_MAX
                    ; /* r: [0,1] */
                k = (int) (nmix * r);
                if (k == nmix) k = nmix-1;
                weights[n][k] = 1.0F;
        }
        em_maximization(data, weights);
        return 0;
}

#define HEADER_SIZE 15
static char header[HEADER_SIZE] = "UPC:␣GMM␣V␣2.0";

std::istream& GMM::read(std::istream &is) {
        char s[HEADER_SIZE];
        is.read(s, HEADER_SIZE);
        if (string(s) != string(header))
                is.setstate(ios::failbit);
        else
                is >> w >> mu >> inv_sigma;
        nmix = mu.nrow();
        vector_size = mu.ncol();
        return is;
}
std::ostream& GMM::write(std::ostream &os) const {
        os.write(header, HEADER_SIZE);
        os << w << mu << inv_sigma;
        return os;
}

std::ostream& GMM::print(std::ostream &os) const {
        unsigned int k, i;
        os << "GMM:␣nmix=" << nmix << ";␣vector_size=" <<
             vector_size << endl;
        for (k=0; k<nmix; ++k) {
                os << "w[" << k << "]=\t" << w[k] << '\n'
                    ;

                os << "mu[" << k << "]=" << mu[k][0];
                for (i=1; i<vector_size; ++i)
                        os << "\t" << mu[k][i];
                os << '\n';
```

```
                    os << "sig[" << k << "]=" << 1/inv_sigma[
                        k][0];
                    for (i=1; i<vector_size; ++i)
                          os << "\t" << 1/inv_sigma[k][i];
                    os << '\n' << endl;

            }
            return os;
}
}
```

3.2 gmm_classify.cpp

```
#include <unistd.h> //getopt function, to parse options
#include <iostream>
#include <fstream>
#include "filename.h"
#include "gmm.h"

using namespace std;
using namespace upc;

const string DEF_FEAT_EXT = "mcp";
const string DEF_GMM_EXT  = "gmc";

int usage(const char *progname, int err);

int read_options(int ArgC, const char *ArgV[], vector<
    Directory> &input_dirs, vector<Ext> &input_exts,
                    vector<Directory> &gmm_dirs, vector<Ext>
                        &gmm_exts,
                    vector<string> &input_filenames,
                    vector<string> &gmm_filenames);

int read_gmms(const Directory &dir, const Ext &ext, const
     vector<string> &gmm_filenames, vector<GMM> &vgmm);


int classify(const vector<GMM> &vgmm, const fmatrix &dat)
     {
  float  lprob, maxlprob = -1e38;
  int maxind  = -1;

  //TODO .. assign maxind to the best index of vgmm
  //for each gmm, call logprob. Implement this function
      in gmm.cpp
  maxind = 0;

  for(int i=0;i<vgmm.size();i++){
         if(maxlprob < vgmm[i].logprob(dat)){maxind=i;
            maxlprob=vgmm[i].logprob(dat);}
  }

  return maxind;
}

int main(int argc, const char *argv[]) {

  vector<Directory> input_dirs, gmm_dirs;
  vector<Ext> input_exts, gmm_exts;
  vector<string> input_filenames, gmm_filenames;

  int retv = read_options(argc, argv, input_dirs,
     input_exts,
                        gmm_dirs, gmm_exts,
                           input_filenames,
                           gmm_filenames);

  if (retv != 0)
    return usage(argv[0], retv);
#if 0
  cout << "IDIR——————\n"; for (unsigned int i=0; i<
     input_dirs.size(); ++i) cout << input_dirs[i] <<
     endl;
  cout << "GDIR——————\n"; for (unsigned int i=0; i<
     gmm_dirs.size(); ++i) cout << gmm_dirs[i] << endl;
  cout << "IEXT——————\n"; for (unsigned int i=0; i<
     input_exts.size(); ++i) cout << input_exts[i] <<
     endl;
  cout << "GEXT——————\n"; for (unsigned int i=0; i<
     gmm_exts.size(); ++i) cout << gmm_exts[i] << endl;
  cout << "INAM——————\n"; for (unsigned int i=0; i<
     input_filenames.size(); ++i) cout << input_filenames
     [i] << endl;
  cout << "GNAM——————\n"; for (unsigned int i=0; i<
     gmm_filenames.size(); ++i) cout << gmm_filenames[i]
     << endl;
#endif



  /*
```

```
    Toni: I have implemented the reading of arguments for
        multiple GMM/Features. Read GMMs
    But here I will only use the first set of GMM/vectors
        .

    You can use data like this ...
    <vector<vector<GMM> > mgmm; mgmm.resize(3); mgmm[0] =
        vgmm;
    <vector<fmatrix> vfmat;

  */

  vector<GMM> vgmm;
  retv = read_gmms(gmm_dirs[0], gmm_exts[0],
      gmm_filenames, vgmm);
  if (retv != 0)
    return usage(argv[0], retv);

  ///Read and classify files
  for (unsigned int i=0; i<input_filenames.size(); ++i) {
    fmatrix dat;
    string path = input_dirs[0] + input_filenames[i] +
        input_exts[0];
    ifstream ifs(path.c_str(), ios::binary);
    if (ifs.good())
      ifs >> dat;

    if (!ifs.good()) {
      cerr << "Error␣reading␣data␣file:␣" << path << endl
          ;
      return usage(argv[0],1);
    }

    int nclass;
    nclass = classify(vgmm, dat);
    cout << input_filenames[i] << '\t' << gmm_filenames[
        nclass] << endl;
  }

  return 0;
}

int read_gmms(const Directory &dir, const Ext &ext, const
    vector<string> &filenames, vector<GMM> &vgmm) {
  vgmm.clear();
  GMM gmm;
  for (unsigned int i=0; i<filenames.size(); ++i) {
    string path = dir + filenames[i] + ext;
    ifstream ifs(path.c_str(), ios::binary);
    if (ifs.good())
      ifs >> gmm;

    if (!ifs.good()) {
      cerr << "Error␣reading␣GMM␣file:␣" << path << endl;
      return -1;
    }
    vgmm.push_back(gmm);
    //     gmm.print(cout) << "--------------------------------\n";
  }
  return 0;
}

int usage(const char *progname, int err)  {
  cerr << "Usage:␣" << progname << "␣[options]␣list_gmm␣
      list_of_test_files\n\n";

  cerr << "Options␣can␣be:␣\n"
       << "␣␣-d␣dir\tDirectory␣of␣the␣feature␣files␣(def.
          ␣\".\")\n"
       << "␣␣-e␣ext\tExtension␣of␣the␣feature␣files␣(def.
          ␣\"" << DEF_FEAT_EXT << "\")\n"
       << "␣␣-D␣dir\tDirectory␣of␣the␣gmm␣files␣(def.␣
          \".\")\n"
       << "␣␣-e␣ext\tExtension␣of␣the␣gmm␣files␣(def.␣\""
           << DEF_GMM_EXT << "\")\n\n";

  cerr << "For␣each␣input␣sentence,␣different␣feature␣
      files␣(and␣different␣GMMs)\n"
       << "can␣be␣provided␣using␣several␣times␣the␣
           options␣-d␣-e␣-D␣and␣-E\n";

  return err;
}

int read_options(int ArgC, const char *ArgV[], vector<
     Directory> &input_dirs, vector<Ext> &input_exts,
                    vector<Directory> &gmm_dirs, vector<Ext>
                        &gmm_exts,
                    vector<string> &input_filenames,
                    vector<string> &gmm_filenames) {
  char option;
```

```
//optarg and optind are global variables declared and
    set by the getopt() function

while ((option = getopt(ArgC, (char **)ArgV, "d:e:D:E:"
    )) != -1) {
  switch (option) {
  case 'd': input_dirs.push_back(optarg); break;
  case 'e': input_exts.push_back(optarg); break;
  case 'D': gmm_dirs.push_back(optarg); break;
  case 'E': gmm_exts.push_back(optarg); break;
  case '?': return -1;
  }
}
if (input_dirs.empty()) input_dirs.push_back("./");
if (gmm_dirs.empty()) gmm_dirs.push_back("./");
if (input_exts.empty()) input_exts.push_back(
    DEF_FEAT_EXT);
if (gmm_exts.empty()) gmm_exts.push_back(DEF_GMM_EXT);


if (input_dirs.size() != input_exts.size() ||
     input_dirs.size() != gmm_dirs.size() ||
     input_dirs.size() != gmm_exts.size()) {
  cerr << ArgV[0] << ":␣ERROR␣-␣Same␣number␣of␣feature/
      gmm␣directories/extensions␣need␣to␣be␣provided."
      << endl;
  return -2;
}

//Add ending '/' to directories, and leading '.' to
    extensions
for (unsigned int i=0; i<input_dirs.size(); ++i) {
  if (!input_dirs[i].empty() && *(input_dirs[i].rbegin
      ()) != '/') input_dirs[i] += '/';
  if (!gmm_dirs[i].empty() && *(gmm_dirs[i].rbegin())
      != '/') gmm_dirs[i] += '/';
  if (!input_exts[i].empty() && input_exts[i][0] != '.'
      ) input_exts[i] = '.' + input_exts[i];
  if (!gmm_exts[i].empty() && gmm_exts[i][0] != '.')
      gmm_exts[i] = '.' + gmm_exts[i];
}

//advance argc and argv to skip read options
ArgC -= optind;
ArgV += optind;
```

```
  if (ArgC != 2)
    return -3;

  //Save name of gmm files in vector 'gmm_filenames'
  ifstream is(ArgV[0]);
  if (!is.good()) {
    cerr << "ERROR␣opening␣list␣of␣gmm␣files:␣" << ArgV
        [0] << endl;
    return -4;
  }
  string s;
  while (is >> s)
    gmm_filenames.push_back(s);
  is.close();

  //Save name of files in vector 'input_filenames'
  is.open(ArgV[1]);
  if (!is.good()) {
    cerr << "ERROR␣opening␣list␣of␣test␣files:␣" << ArgV
        [1] << endl;
    return -5;
  }
  while (is >> s)
    input_filenames.push_back(s);
  is.close();

  return 0;
}
```

3.3 gmm_train.cpp

```
#include <unistd.h> //getopt function, to parse options
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "gmm.h"
#include "filename.h"

using namespace std;
using namespace upc;

const string DEF_INPUT_EXT = "mcp";
const unsigned int DEF_ITERATIONS = 20;
const float DEF_THR = 1e-3;
const unsigned int DEF_NMIXTURES = 5;
const string DEF_GMMFILE = "output.gmc";
```

```
int read_data(const string & input_directory, const
   string & input_extension,
             const vector<string> &filenames, fmatrix &
                 dat);

int usage(const char *progname, int err);

int read_options(int ArgC, const char *ArgV[], Directory
   &input_dir, Ext &input_ext, vector<string> &filenames,
                 unsigned int &nmix, string &gmm_filename
                     ,
                 unsigned int &niterations, unsigned int
                     &ending_iterations, float &threshold,
                      float &ending_threshold,
                 int &init_method, unsigned int &verbose)
                     ;

int main(int argc, const char *argv[]) {

  Directory input_dir;
  Ext input_ext(DEF_INPUT_EXT);
  vector<string> filenames;
  unsigned int nmix = DEF_NMIXTURES, verbose;
  Filename gmm_filename(DEF_GMMFILE);
  unsigned int niterations=DEF_ITERATIONS,
      ending_iterations=DEF_ITERATIONS;
  float threshold=DEF_THR, ending_threshold=DEF_THR;
  int init_method=0;

  ///Read command line options
  int retv = read_options(argc, argv, input_dir,
      input_ext, filenames,
                       nmix, gmm_filename,
                       niterations, ending_iterations,
                           threshold, ending_threshold,
                       init_method, verbose);
  if (retv != 0)
    return usage(argv[0], retv);

  //Read data from filenames
  fmatrix data;
  read_data(input_dir, input_ext, filenames, data);
  cout << "DATA:␣" << data.nrow() << "␣x␣" << data.ncol()
       << endl;
  GMM gmm;
  //TODO: initialize GMM from data
  //You can use several options: random, vq, em_split ...
  gmm.vq_lbg(data,nmix,ending_iterations,threshold,
      verbose);

  //Apply EM to estimate GMM parameters (complete the
      funcion in gmm.cpp)
  gmm.em(data, ending_iterations, ending_threshold,
      verbose);


  //Create directory, if it is needed
  gmm_filename.checkDir();
  //Save gmm
  ofstream ofs(gmm_filename.c_str(), ios::binary);
  ofs << gmm;

  bool show_gmm=false;
  if (show_gmm)
    gmm.print(cout);

  return 0;
}

int usage(const char *progname, int err) {
  cerr << "Usage: " << progname << " [options] 
      list_of_train_files\n";
  cerr << "Usage: " << progname << " [options] -F 
      train_file1 ...\n\n";

  cerr << "Options can be: \n"
       << "  -d dir\tDirectory of the input files (def. 
           \".\")\n"
       << "  -e ext\tExtension of the input files (def. 
           \"" << DEF_INPUT_EXT << "\")\n"
       << "  -m mix\tNumber of mixtures (def. " <<
           DEF_NMIXTURES << ")\n"
       << "  -g name\tName of output GMM file  (def. " <<
            DEF_GMMFILE << ")\n"
       << "  -n ite\tNumber of (intermediate) iterations 
           of EM (def. " << DEF_ITERATIONS << ")\n"
       << "  -N ite\tNumber of final iterations of EM (
           def. " << DEF_ITERATIONS << ")\n"
       << "  -i init\tInitialization method (def. 0)\n"
       << "␣␣-t␣thr\tLogProbability␣threshold␣of␣(
           intermediate)␣EM␣iterations␣(def.␣" << DEF_THR
           << ")\n"
       << "␣␣-T␣thr\tLogProbability␣threshold␣of␣final␣EM
           ␣iterations␣(def.␣" << DEF_THR << ")\n"
       << "␣␣-v␣int\tBit␣code␣to␣control␣\"verbosity\";␣
           eg:␣5␣=>␣00000101" << ")\n";
  return err;
}

int read_options(int ArgC, const char *ArgV[], Directory
   &input_dir, Ext &input_ext, vector<string> &filenames,
                 unsigned int &nmix, string &gmm_filename
                    ,
                 unsigned int &niterations, unsigned int
                    &ending_iterations, float &threshold,
                     float &ending_threshold,
                 int &init_method, unsigned int &verbose)
                     {

  char option;
  bool use_list = true;
  filenames.clear();

  //optarg and optind are global variables declared and
     set by the getopt() function

  while ((option = getopt(ArgC, (char **)ArgV, "d:e:m:g:n
     :N:t:T:i:v:F")) != -1) {
    switch (option) {
    case 'd': input_dir = optarg; break;
    case 'e': input_ext = optarg; break;
    case 'm': nmix = atoi(optarg); break;
    case 'g': gmm_filename = optarg; break;
    case 'n': niterations = atoi(optarg); break;
    case 'N': ending_iterations = atoi(optarg); break;
    case 't': threshold = atof(optarg); break;
    case 'T': ending_threshold = atof(optarg); break;
    case 'i': init_method = atoi(optarg); break;
    case 'v': verbose = atoi(optarg); break;
    case 'F': use_list=false; break;
    case '?': return -1;
    }
  }

  if (nmix == 0) {
    cerr << ArgV[0] << ":␣nmixtures␣must␣be␣>␣0\n";
    return -2;
  }

  if (!input_dir.empty() && *input_dir.rbegin() != '/')
     input_dir += '/';
  if (!input_ext.empty() && input_ext[0] != '.')
     input_ext = '.' + input_ext;

  //advance argc and argv to skip read options
  ArgC -= optind;
  ArgV += optind;

  //Save name of files in vector 'filenames'
  if (use_list) {
    if (ArgC != 1) {
      cerr << "ERROR␣no␣list␣of␣files␣provided" << endl;
      return -2;
    }
    ifstream is(ArgV[0]);
    if (!is.good()) {
      cerr << "ERROR␣opening␣list␣of␣files:␣" << ArgV[0]
         << endl;
      return -3;
    }
    string s;
    while (is >> s)
      filenames.push_back(s);
  } else {
    for (int i=0; i<ArgC; ++i)
      filenames.push_back(ArgV[i]);
  }
  return 0;
}


int read_data(const string & input_directory, const
    string & input_extension,
              const vector<string> &filenames, fmatrix &
                 dat) {
  fmatrix dat1;
  for (unsigned int i=0; i<filenames.size(); ++i) {
    string path = input_directory + filenames[i] +
        input_extension;
    ifstream is(path.c_str(), ios::binary);
    if (!is.good()) {
      cerr << "Error reading file: " << path << endl;
      dat.reset();
      return -1;
    }
    if (i==0) {
      is >> dat;
    } else {
      is >> dat1;
      if (dat1.ncol() != dat.ncol()) {
        cerr << "Error in vector dimension: " <<
            filenames[i] << dat1.ncol()
             << " (expected: " << dat.ncol() << ")\n";
        dat.reset();
        return -1;
      }
      int row = dat.nrow();
      dat.resize(dat.nrow()+dat1.nrow(), dat.ncol());
      for (unsigned int i=0; i<dat1.nrow(); ++i, ++row)
        for (unsigned int j=0; j<dat1.ncol(); ++j)
          dat[row][j] = dat1[i][j];
    }
  }
  return 0;
}
```

3.4 gmm_verify.cpp

```
#include <unistd.h> //getopt function, to parse options
#include <iostream>
#include <fstream>
#include "filename.h"
#include "gmm.h"

using namespace std;
using namespace upc;

const string DEF_FEAT_EXT = "mcp";
const string DEF_GMM_EXT  = "gmc";


float verify(const vector<GMM> &vgmm, unsigned int u, const
    fmatrix &dat) {
        float back=vgmm[vgmm.size()-1].logprob(dat);
        float speaker=vgmm[u].logprob(dat);
        return (speaker-back);
}
```

```
int read_gmms(const Directory &dir, const Ext &ext, const
    vector<string> &filenames, vector<GMM> &vgmm) {
        vgmm.clear();
        GMM gmm;

        for (unsigned int i=0; i<filenames.size(); ++i) {
                string path = dir + filenames[i] + ext;
                ifstream ifs(path.c_str(), ios::binary);
                if (ifs.good())
                        ifs >> gmm;

                if (!ifs.good()) {
                        cerr << "Error␣reading␣GMM␣file:␣
                            " << path << endl;
                        return -1;
                }
                vgmm.push_back(gmm);
                //      gmm.print(cout) <<
                    "----------------------------\n";
        }
        return 0;
}

int usage(const char *progname, int err) {
        cerr << "Usage:␣" << progname << "␣[options]␣
            list_gmm␣list_of_test_files␣list_of_candidate\
            n\n";

        cerr << "Options␣can␣be:␣\n"
                        << "␣␣-d␣dir\tDirectory␣of␣the␣
                            feature␣files␣(def.␣\".\")\n"
                        << "␣␣-e␣ext\tExtension␣of␣the␣
                            feature␣files␣(def.␣\"" <<
                            DEF_FEAT_EXT << "\")\n"
                        << "␣␣-D␣dir\tDirectory␣of␣the␣
                            gmm␣files␣(def.␣\".\")\n"
                        << "␣␣-e␣ext\tExtension␣of␣the␣
                            gmm␣files␣(def.␣\"" <<
                            DEF_GMM_EXT << "\")\n\n"
                        << "For␣each␣input␣sentence,␣
                            different␣feature␣files␣(and␣
                            different␣GMMs)\n"
                        << "can␣be␣provided␣using␣several
                            ␣times␣the␣options␣-d␣-e␣-D␣
```

```
                                                  and␣-E\n";

        return err;
}

int read_options(int ArgC, const char *ArgV[], vector<
    Directory> &input_dirs, vector<Ext> &input_exts,
                vector<Directory> &gmm_dirs, vector<Ext>
                   &gmm_exts,
                vector<string> &input_filenames,
                vector<string> &gmm_filenames,
                vector<string> &candidates) {

        char option;
        //optarg and optind are global variables declared
             and set by the getopt() function

        while ((option = getopt(ArgC, (char **)ArgV, "d:e
            :D:E:")) != -1) {
                switch (option) {
                case 'd': input_dirs.push_back(optarg);
                    break;
                case 'e': input_exts.push_back(optarg);
                    break;
                case 'D': gmm_dirs.push_back(optarg);
                    break;
                case 'E': gmm_exts.push_back(optarg);
                    break;
                case '?': return -1;
                }
        }
        if (input_dirs.empty()) input_dirs.push_back("./"
            );
        if (gmm_dirs.empty()) gmm_dirs.push_back("./");
        if (input_exts.empty()) input_exts.push_back(
            DEF_FEAT_EXT);
        if (gmm_exts.empty()) gmm_exts.push_back(
            DEF_GMM_EXT);


        if (input_dirs.size() != input_exts.size() ||
                        input_dirs.size() != gmm_dirs.
                            size() ||
                        input_dirs.size() != gmm_exts.
                            size()) {
```

```
                cerr << ArgV[0] << ":␣ERROR␣-␣Same␣number
                    ␣of␣feature/gmm␣directories/extensions
                    ␣need␣to␣be␣provided." << endl;
                return -2;
        }

        //Add ending '/' to directories, and leading '.'
            to extensions
        for (unsigned int i=0; i<input_dirs.size(); ++i)
            {
                if (!input_dirs[i].empty() && *(
                    input_dirs[i].rbegin()) != '/')
                    input_dirs[i] += '/';
                if (!gmm_dirs[i].empty() && *(gmm_dirs[i
                    ].rbegin()) != '/') gmm_dirs[i] += '/'
                    ;
                if (!input_exts[i].empty() && input_exts[
                    i][0] != '.') input_exts[i] = '.' +
                    input_exts[i];
                if (!gmm_exts[i].empty() && gmm_exts[i
                    ][0] != '.') gmm_exts[i] = '.' +
                    gmm_exts[i];
        }

        //advance argc and argv to skip read options
        ArgC -= optind;
        ArgV += optind;

        if (ArgC != 3)
                return -3;

        //Save name of gmm files in vector 'gmm_filenames
            '
        ifstream is(ArgV[0]);
        if (!is.good()) {
                cerr << "ERROR␣opening␣list␣of␣gmm␣files:
                    ␣" << ArgV[0] << endl;
                return -4;
        }
        string s;
        while (is >> s)
                gmm_filenames.push_back(s);
        is.close();

        //Save name of files in vector 'input_filenames'
        is.open(ArgV[1]);
        if (!is.good()) {
                cerr << "ERROR␣opening␣list␣of␣test␣files
                    :␣" << ArgV[1] << endl;
                return -5;
        }
        while (is >> s)
                input_filenames.push_back(s);
        is.close();

        //Save name of files in vector 'input_filenames'
        is.open(ArgV[2]);
        if (!is.good()) {
                cerr << "ERROR␣opening␣list␣of␣user␣
                    candidates:␣" << ArgV[2] << endl;
                return -6;
        }
        while (is >> s)
                candidates.push_back(s);
        is.close();

        return 0;
}
int main(int argc, const char *argv[]) {

        vector<Directory> input_dirs, gmm_dirs;
        vector<Ext> input_exts, gmm_exts;
        vector<string> input_filenames, gmm_filenames,
            candidates;
        int retv = read_options(argc, argv, input_dirs,
            input_exts,gmm_dirs, gmm_exts, input_filenames
            , gmm_filenames,candidates);

        if (retv != 0)
                return usage(argv[0], retv);
#if 0
        cout << "IDIR------------\n"; for (unsigned int i
            =0; i<input_dirs.size(); ++i) cout <<
            input_dirs[i] << endl;
        cout << "GDIR------------\n"; for (unsigned int i
            =0; i<gmm_dirs.size(); ++i) cout << gmm_dirs[i
            ] << endl;
        cout << "IEXT------------\n"; for (unsigned int i
            =0; i<input_exts.size(); ++i) cout <<
            input_exts[i] << endl;
        cout << "GEXT------------\n"; for (unsigned int i
            =0; i<gmm_exts.size(); ++i) cout << gmm_exts[i
            ] << endl;
        cout << ”INAM————————\n”; for (unsigned int i
            =0; i<input_filenames.size(); ++i) cout <<
            input_filenames[i] << endl;
        cout << ”GNAM————————\n”; for (unsigned int i
            =0; i<gmm_filenames.size(); ++i) cout <<
            gmm_filenames[i] << endl;

#endif

        if (input_filenames.size() != candidates.size())
            {
                cerr << ”Error:␣num_candidates␣!=␣
                    num_files\n”;
        }

        /*
    Toni: I have implemented the reading of arguments for
        multiple GMM/Features. Read GMMs
    But here I will only use the first set of GMM/vectors
        .

    You can use data like this ...
    <vector<vector<GMM> > mgmm; mgmm.resize(3); mgmm[0] =
        vgmm;
    <vector<fmatrix> vfmat;

         */

        vector<GMM> vgmm;
        retv = read_gmms(gmm_dirs[0], gmm_exts[0],
            gmm_filenames, vgmm);
        if (retv != 0)
                return usage(argv[0], retv);



        ///Read and verify files
        for (unsigned int i=0; i<input_filenames.size();
            ++i) {
                fmatrix dat;
                string path = input_dirs[0] +
                    input_filenames[i] + input_exts[0];
                ifstream ifs(path.c_str(), ios::binary);
                if (ifs.good())
                        ifs >> dat;
                if (!ifs.good()) {
                        cerr << "Error␣reading␣data␣file:
                           ␣" << path << endl;
                        return usage(argv[0],1);
                }
                unsigned int aux;
                for(unsigned int j=0;j<gmm_filenames.size
                    ();++j){
                        if(candidates[i]==gmm_filenames[j
                           ]){aux=j; break;}
                }
                cout<<input_filenames[i]<<'\t'<<
                    candidates[i]<<'\t'<<verify(vgmm,aux,
                    dat)<<endl;
        }

        return 0;
}
```

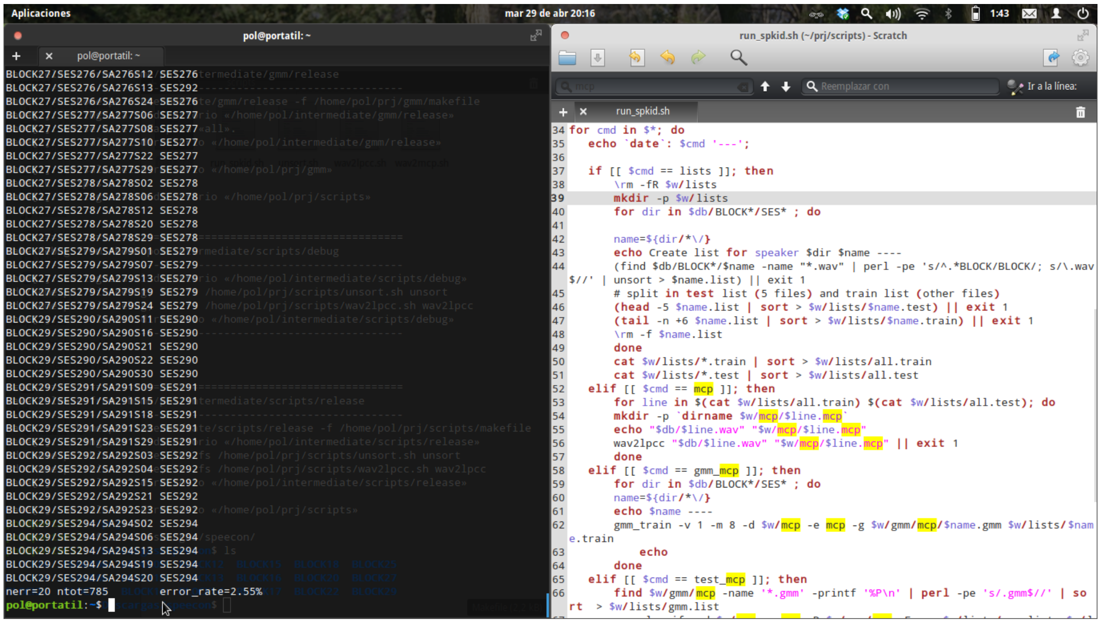
3.5 run_spkid

El script run_spkid.sh ens permetra avaluar el nostre projecte.

- la opció **lists** selecciona una part dels fitxers dels fitxers de la base de dades per a entrenament i en reserva una altre part per a la avaluació del sistema.
- la opció **mcp** computa el vector de caracteristiques de tots els audios de la base de dades, lpcc o mpcc.
- la opció **gmm_mcp** entrena una gmm per a cada locutor de la base de dades.
- la opció **background** entrena una gmm amb tots els locutors de la base de dades.
- la opció **test_mcp** testeja el nostre sistema mitjançant els audis que ens haviem reservat.
- la opció **verify** fa un test de verificació de locutors.

```
#!/bin/bash

# Scripting is very useful to repeat tasks, as testing
   different configuration, multiple files, etc.
# This bash script is provided as one example
# Please, adapt at your convinience
# Antonio Bonafonte, April 2013
# Set the proper value to the next variables

w=$HOME/tmp # work directory
db=$HOME/Descargas/speecon # directory with the input
    database
pavbin=$HOME/bin/release # directory with the programs

# Add the path of bin files to the path where the
    operative system looks for 'programs'
PATH=$PATH:$pavbin

CMDS="lists␣mcp␣d1c␣d2c␣gmm_mcp␣test_mcp␣finaltest"

if [[ $# < 1 ]]; then
    echo "$0␣cmd1␣[...]"
    echo "Where␣commands␣can␣be:"
    echo "␣␣␣␣lists:␣create,␣for␣each␣spk,␣training␣and␣
        devel.␣list␣of␣files"
    echo "␣␣␣␣␣␣mcp:␣feature␣extraction␣(mel␣cepstrum␣
        parameters)"
#    echo "       d1c: 1st derivative"
#    echo "       d2c: 2nd derivative"
    echo "␣␣␣gmm_mcp:␣train␣gmm␣for␣the␣mcp␣features"
    echo "␣background:␣compute␣the␣background␣model"
    echo "␣verify:␣run␣a␣verify␣test␣with␣candidates.list"
    echo "␣␣test_mcp:␣test␣GMM␣using␣only␣mcp␣features"
    exit 1
fi


for cmd in $*; do
    echo `date`: $cmd '---';

    if [[ $cmd == lists ]]; then
        \rm -fR $w/lists
        mkdir -p $w/lists
        for dir in $db/BLOCK*/SES* ; do

            name=${dir/*\/}
            echo Create list for speaker $dir $name ----
            (find $db/BLOCK*/$name -name "*.wav" | perl -
                pe 's/^.*BLOCK/BLOCK/; s/\.wav$//' | unsort
                 > $name.list) || exit 1
            # split in test list (5 files) and train list
                (other files)
            (head −5 $name.list | sort > $w/lists/$name.
                test) || exit 1
            (tail −n +6 $name.list | sort > $w/lists/$name
                .train) || exit 1
            \rm −f $name.list
        done
        cat $w/lists/*.train | sort > $w/lists/all.train
        cat $w/lists/*.test | sort > $w/lists/all.test
    elif [[ $cmd == mcp ]]; then
        for line in $(cat $w/newmcp.txt); do
            mkdir −p ‘dirname $w/mcp/$line.mcp‘
            echo ”$db/$line.wav” ”$w/mcp/$line.mcp”
            wav2mcp ”$db/$line.wav” ”$w/mcp/$line.mcp” ||
                exit 1
        done
    elif [[ $cmd == gmm_mcp ]]; then
        for dir in $db/BLOCK*/SES* ; do
            name=${dir/*\/}
            echo $name ——
            gmm_train −v 1 –m 12 −d $w/mcp −e mcp −g $w/
               gmm/mcp/$name.gmm $w/lists/$name.train
            echo
        done
   elif [[ $cmd == test_mcp ]]; then
        find $w/gmm/mcp −name ’*.gmm’ −printf ’%P\n’ |
            perl −pe ’s/.gmm$//’ | sort  > $w/lists/gmm.
            list
        gmm_classify −d $w/mcp −e mcp –D $w/gmm/mcp –E gmm
             $w/lists/gmm.list  $w/lists/all.test | tee $w/
            result.log
        perl −ne ’BEGIN {$ok=0; $err=0}
                    next unless /^.*SA(...).*SES(...).*$/;
                    if ($1 == $2) {$ok++}
                    else {$err++}
                    END {printf ”nerr=%d\tntot=%d\
                        terror_rate=%.2f%%\n”, ($err, $ok+
                        $err, 100*$err/($ok+$err))}’ $w/
                        result.log

   elif [[ $cmd == background ]]; then
     gmm_train −v 1 –m 12 −d $w/mcp −e mcp −g $w/gmm/mcp/
         background.gmm $w/lists/all.train

 elif [[ $cmd == verify ]]; then
```

```
        gmm_verify -d $w/mcp -e mcp -D $w/gmm/mcp -E gmm $w/
            lists/gmm.list  $w/verif_files.txt $w/verif_target
            .txt | tee $w/resultverify.log

    elif [[ $cmd == final_test ]]; then
        echo "To be implemented ..."
    else
        echo "undefined command $cmd" && exit 1
    fi
done

exit 0
```



```
BLOCK27/SES276/SA276S12 SES276
BLOCK27/SES276/SA276S13 SES292
BLOCK27/SES276/SA276S24 SES276
BLOCK27/SES277/SA277S06 SES277
BLOCK27/SES277/SA277S08 SES277
BLOCK27/SES277/SA277S10 SES277
BLOCK27/SES277/SA277S22 SES277
BLOCK27/SES277/SA277S29 SES277
BLOCK27/SES278/SA278S02 SES278
BLOCK27/SES278/SA278S06 SES278
BLOCK27/SES278/SA278S12 SES278
BLOCK27/SES278/SA278S20 SES278
BLOCK27/SES278/SA278S29 SES278
BLOCK27/SES279/SA279S01 SES279
BLOCK27/SES279/SA279S07 SES279
BLOCK27/SES279/SA279S13 SES279
BLOCK27/SES279/SA279S19 SES279
BLOCK27/SES279/SA279S24 SES279
BLOCK29/SES290/SA290S11 SES290
BLOCK29/SES290/SA290S16 SES290
BLOCK29/SES290/SA290S21 SES290
BLOCK29/SES290/SA290S22 SES290
BLOCK29/SES290/SA290S30 SES290
BLOCK29/SES291/SA291S09 SES291
BLOCK29/SES291/SA291S15 SES291
BLOCK29/SES291/SA291S18 SES291
BLOCK29/SES291/SA291S23 SES291
BLOCK29/SES291/SA291S29 SES291
BLOCK29/SES292/SA292S03 SES292
BLOCK29/SES292/SA292S04 SES292
BLOCK29/SES292/SA292S15 SES292
BLOCK29/SES292/SA292S21 SES292
BLOCK29/SES292/SA292S23 SES292
BLOCK29/SES294/SA294S02 SES294
BLOCK29/SES294/SA294S06 SES294
BLOCK29/SES294/SA294S13 SES294
BLOCK29/SES294/SA294S19 SES294
BLOCK29/SES294/SA294S20 SES294
nerr=20 ntot=785        error_rate=2.55%
pol@portatil:~$
```

```
for cmd in $*; do
   echo `date`: $cmd '---';

   if [[ $cmd == lists ]]; then
       \rm -fR $w/lists
       mkdir -p $w/lists
       for dir in $db/BLOCK*/SES* ; do

       name=${dir/*\/}
       echo Create list for speaker $dir $name ----
       (find $db/BLOCK*/$name -name "*.wav" | perl -pe 's/^.*BLOCK/BLOCK/; s/\.wav$//' | unsort > $name.list) || exit 1
       # split in test list (5 files) and train list (other files)
       (head -5 $name.list | sort > $w/lists/$name.test) || exit 1
       (tail -n +6 $name.list | sort > $w/lists/$name.train) || exit 1
       \rm -f $name.list
       done
       cat $w/lists/*.train | sort > $w/lists/all.train
       cat $w/lists/*.test | sort > $w/lists/all.test
   elif [[ $cmd == mcp ]]; then
       for line in $(cat $w/lists/all.train) $(cat $w/lists/all.test); do
       mkdir -p `dirname $w/mcp/$line.mcp`
       echo "$db/$line.wav" "$w/mcp/$line.mcp"
       wav2lpcc "$db/$line.wav" "$w/mcp/$line.mcp" || exit 1
       done
   elif [[ $cmd == gmm_mcp ]]; then
       for dir in $db/BLOCK*/SES* ; do
       name=${dir/*\/}
       echo $name ----
       gmm_train -v 1 -m 8 -d $w/mcp -e mcp -g $w/gmm/mcp/$name.gmm $w/lists/$name.train
           echo
       done
   elif [[ $cmd == test_mcp ]]; then
       find $w/gmm/mcp -name '*.gmm' -printf '%P\n' | perl -pe 's/.gmm$//' | sort  > $w/lists/gmm.list
```

Figure 1: eficacia del sistema amb coeficients lpc

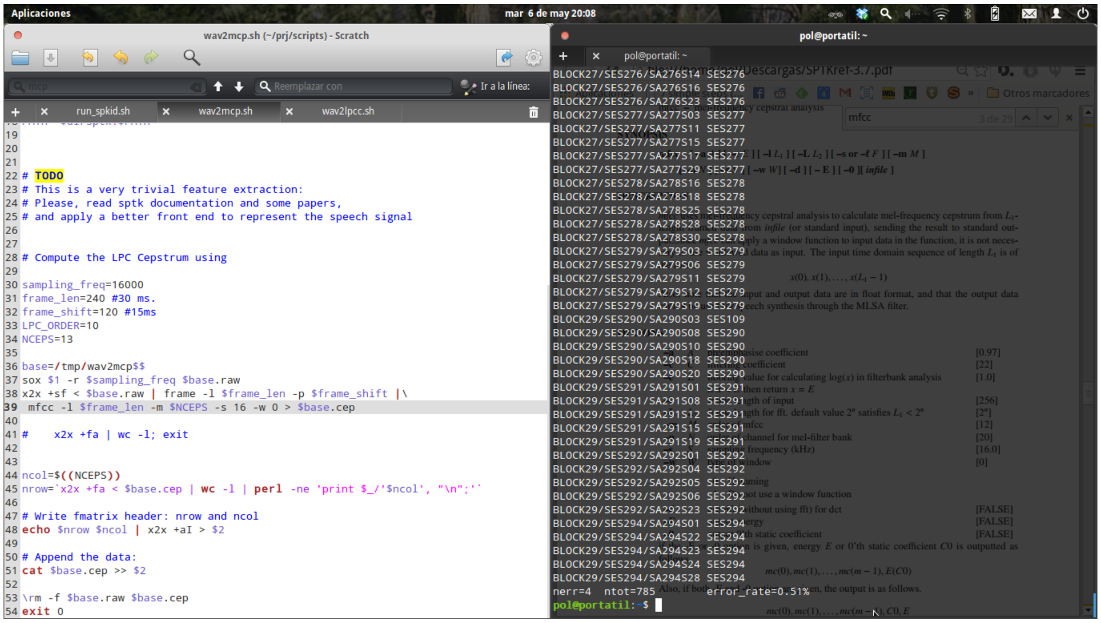
4 Conclusions

En el apartat de classificació de locutor el nostre sistema ha estat clarament un exit,hem apreciat una millora en el nostre sistema al usar mel frequency cepstrums. En les nostres proves hem vist que el nombre de coeficients optim era de 13, mes enllà no hem vist una millora significativa. A més hem suposat que cada coeficient tindria una variança associada i per tant una única gaussiana, suposit que mes tard hem vist que era erroni, sent un parametre clau del sistema.

En quant a la verficació hem obtingut els seguents resultats:

```
=======================================
THR: 5.35862779999998
Missed:     343/374=0.9171
FalseAlarm: 0/11981=0.0000
----------------------------------------------
==> CostDetection: 91
=======================================
```

Els resultats en aquesta aplicació no son tant bons, no obstant aixo es causat en part per la curta llongitud dels audios de verificació, molt més curts que en els que hem fet servir per classificació.



```
# TODO
# This is a very trivial feature extraction:
# Please, read sptk documentation and some papers,
# and apply a better front end to represent the speech signal


# Compute the LPC Cepstrum using

sampling_freq=16000
frame_len=240 #30 ms.
frame_shift=120 #15ms
LPC_ORDER=10
NCEPS=13

base=/tmp/wav2mcp$$
sox $1 -r $sampling_freq $base.raw
x2x +sf < $base.raw | frame -l $frame_len -p $frame_shift |\
 mfcc -l $frame_len -m $NCEPS -s 16 -w 0 > $base.cep

#    x2x +fa | wc -l; exit


ncol=$((NCEPS))
nrow=`x2x +fa < $base.cep | wc -l | perl -ne 'print $_/'$ncol', "\n";'`

# Write fmatrix header: nrow and ncol
echo $nrow $ncol | x2x +aI > $2

# Append the data:
cat $base.cep >> $2

\rm -f $base.raw $base.cep
exit 0
```

```
BLOCK27/SES276/SA276S14  SES276
BLOCK27/SES276/SA276S16  SES276
BLOCK27/SES276/SA276S23  SES276
BLOCK27/SES277/SA277S03  SES277
BLOCK27/SES277/SA277S11  SES277
BLOCK27/SES277/SA277S15  SES277
BLOCK27/SES277/SA277S17  SES277
BLOCK27/SES277/SA277S29  SES277
BLOCK27/SES278/SA278S16  SES278
BLOCK27/SES278/SA278S18  SES278
BLOCK27/SES278/SA278S25  SES278
BLOCK27/SES278/SA278S28  SES278
BLOCK27/SES278/SA278S30  SES278
BLOCK27/SES279/SA279S03  SES279
BLOCK27/SES279/SA279S06  SES279
BLOCK27/SES279/SA279S11  SES279
BLOCK27/SES279/SA279S12  SES279
BLOCK27/SES279/SA279S19  SES279
BLOCK29/SES290/SA290S03  SES109
BLOCK29/SES290/SA290S08  SES290
BLOCK29/SES290/SA290S10  SES290
BLOCK29/SES290/SA290S18  SES290
BLOCK29/SES290/SA290S20  SES290
BLOCK29/SES291/SA291S01  SES291
BLOCK29/SES291/SA291S08  SES291
BLOCK29/SES291/SA291S12  SES291
BLOCK29/SES291/SA291S15  SES291
BLOCK29/SES291/SA291S19  SES291
BLOCK29/SES292/SA292S01  SES292
BLOCK29/SES292/SA292S04  SES292
BLOCK29/SES292/SA292S05  SES292
BLOCK29/SES292/SA292S06  SES292
BLOCK29/SES292/SA292S23  SES292
BLOCK29/SES294/SA294S01  SES294
BLOCK29/SES294/SA294S22  SES294
BLOCK29/SES294/SA294S23  SES294
BLOCK29/SES294/SA294S24  SES294
BLOCK29/SES294/SA294S28  SES294
nerr=4  ntot=785         error_rate=0.51%
pol@portatil:~$ 
```

Figure 2: eficacia del sistema amb coeficients mpc

De totes maneres podriem millorar el sistema inicialitzant les gaussianes amb el model de background en comptes de amb vq, ja que d'aquesta manera es té en compte el model de background en cada una de les gmm i els usuaris legitims en surten beneficiats.
Descartem el ús de cadenes de markov per aquesta aplicació, no obstant podriem fer servir un model més sofisticat que permetin un corpus de entrenament mes elevat, com poden ser les xarxes neuronals.