

```

399 void cHMM::trainMS(std::vector< std::vector<Sample_3d> > trainingset){
400
401     boost::numeric::ublas::matrix<double> A_up(numStati, numStati);
402     boost::numeric::ublas::matrix<double> A_down(numStati, numStati);
403
404     std::vector<Gaussian_3d_mixture> gaussians_up;
405     std::vector<Gaussian_3d_mixture> gaussians_down;
406
407     // inizializza A_up, A_down
408     for(int i=0; i<numStati; i++){
409         for(int j=0; j<numStati; j++){
410             A_up(i,j) = 0;
411             A_down(i,j) = 0;
412         }
413     }
414
415     // inizializza gaussians_up, gaussians_down con tutti zero
416     for(int k=0; k<numStati; k++){
417
418         Gaussian_3d_mixture* g1 = new Gaussian_3d_mixture(nGauss, true);
419         Gaussian_3d_mixture* g2 = new Gaussian_3d_mixture(nGauss, true);
420
421         gaussians_up.push_back( *g1 );
422         gaussians_down.push_back( *g2 );
423
424     }
425
426     for(int data=0; data<trainingset.size(); data++){
427
428         std::vector<Sample_3d> current = trainingset.at(data);
429         double scale[current.size()];
430         boost::numeric::ublas::matrix<double> alpha(numStati, current.size());
431         boost::numeric::ublas::matrix<double> beta(numStati, current.size());
432         forwardProc_scale(current, alpha, scale);
433         backwardProc(current, beta, scale);
434
435         // debug
436         /*
437         std::cout<< "scale:" << std::endl;
438         for(int d=0; d<current.size(); d++)
439             std::cout<<scale[d]<<std::endl;
440         */
441
442         double P = getProbabilityFromScale(scale, current.size());
443
444         //debug
445         std::cout<<"Probabilità: "<<P<<std::endl;
446
447         // aggiornamento pi
448         if(isErgodic){
449             for(int i=0; i<numStati; i++){
450                 pi[i] = alpha(i,1) * beta(i,1) / P;
451             }
452
453             // aggiornamento A
454             for(int i=0; i<numStati; i++){
455
456                 for(int j=0; j<numStati; j++){
457
458                     double up = 0;
459                     double down = 0;
460
461                     for(int t=0; t<current.size()-2; t++){
462
463                         up += alpha(i,t) * A(i,j) * B(j,current.at(t+1)) * beta(j,t+1);
464                         down += alpha(i,t) * beta(j,t);
465
466                     }//t
467
468                     A_up(i,j) += up; // / P;
469                     A_down(i,j) += down; // / P;
470
471                 }//j
472             }//i
473
474             //debug
475             //std::cout<<A_up<<std::endl;
476             //std::cout<<A_down<<std::endl;
477
478             // aggiornamento parametri gaussiane
479             // gamma[t][j][k]
480             double **gamma;
481             gamma = (double ***)malloc(current.size() * sizeof(double **));
482             for (int t = 0; t < current.size(); t++)
483             {
484                 gamma[t] = (double **)malloc(numStati * sizeof(double *));
485                 for (int j = 0; j < numStati; j++)
486                 {
487                     gamma[t][j] = (double *)malloc(mixture_vect.at(j).howmany * sizeof(double));
488                 }
489             }
490
491             // calcolo gamma
492             for(int t=0; t<current.size(); t++){ // ciclo sui sample della gesture
493
494                 double sum = 0;
495                 for(int j=0; j<numStati; j++){
496                     sum += alpha(j,t)*beta(j,t);
497                 }
498
499                 for(int j=0; j<numStati; j++){ // ciclo sugli stati
500
501                     for(int k=0; k<mixture_vect.at(j).howmany; k++){ // ciclo sulle componenti della mixture dello stato corrente
502
503                         gamma[t][j][k] = alpha(j,t) * beta(j,t) * mixture_vect.at(j).weight[k] *
504                             mixture_vect.at(j).components.at(k).pdf_3d(current.at(t)) /
505                             ( sum * B(j, current.at(t)) );
506
507                     }
508

```

```

509
510     }
511
512
513     // aggiornamento pesi misture: mixture_vect.at(STAT0).weight[MISTURA]
514     for(int j=0; j<numStati; j++){
515
516         int n_mix = mixture_vect.at(j).howmany;
517
518         for(int k=0; k<n_mix; k++){
519
520             double up = 0, down = 0;
521
522             for(int t=0; t<current.size(); t++){
523
524                 up += gamma[t][j][k];
525
526                 for(int m=0; m<n_mix; m++){
527
528                     down += gamma[t][j][m];
529                 }
530             }
531
532             gaussians_up.at(j).weight[k] += up; // / P;
533             gaussians_down.at(j).weight[k] += down; // / P;
534
535             //mixture_vect.at(j).weight[k] = up / down;
536
537         }
538     }
539
540
541     // aggiornamento medie
542     for(int j=0; j<numStati; j++){
543
544         int n_mix = mixture_vect.at(j).howmany;
545
546         for(int k=0; k<n_mix; k++){
547
548             double down = 0;
549
550             for(int t=0; t<current.size(); t++){
551
552                 down += gamma[t][j][k];
553             }
554
555             for(int n=0; n<3; n++){
556
557                 double up = 0;
558
559                 for(int t=0; t<current.size(); t++){
560
561                     up += gamma[t][j][k] * current.at(t)[n];
562                 }
563
564                 gaussians_up.at(j).components.at(k).mean[n] += up; // / P;
565                 gaussians_down.at(j).components.at(k).mean[n] += down; // / P;
566
567                 //mixture_vect.at(j).components.at(k).mean[n] = up / down;
568             }
569         }
570     }
571
572 }
573
574
575 // aggiornamento covarianze
576 for(int j=0; j<numStati; j++){
577
578     int n_mix = mixture_vect.at(j).howmany;
579
580     for(int k=0; k<n_mix; k++){
581
582         double down = 0;
583
584         for(int t=0; t<current.size(); t++){
585
586             down += gamma[t][j][k];
587         }
588
589         for(int n=0; n<3; n++){
590
591             double up = 0;
592
593             for(int t=0; t<current.size(); t++){
594
595                 up += gamma[t][j][k] *
596                     (current.at(t)[n] - mixture_vect.at(j).components.at(k).mean[n]) *
597                     (current.at(t)[n] - mixture_vect.at(j).components.at(k).mean[n]);
598             }
599
600             // nota: si aggiornano solo le covarianze sulla diagonale, in quanto lavoriamo
601             // con matrici di covarianza diagonali
602             gaussians_up.at(j).components.at(k).cov(n,n) += up; // / P;
603             gaussians_down.at(j).components.at(k).cov(n,n) += down; // / P;
604
605             //mixture_vect.at(j).components.at(k).cov(n,n) = up / down;
606
607         } //n
608     } //k
609 } //j
610
611 // può servire check su covarianze, cioè se cov(i,i) < K, allora cov(i,i) = K
612
613
614
615
616 // libera la memoria
617 for (int t = 0; t < current.size(); t++){
618     for (int j = 0; j < numStati; j++){

```

```

619         free(gamma[t][j]);
620     }
621     free(gamma[t]);
622 }
623 free(gamma);
624
625 }//data
626
627 // aggiorna le matrici A e tutti i parametri delle gaussiane
628 for(int i=0; i<numStati; i++){ // cicla sugli stati
629
630     // aggiorna la matrice A
631     for(int j=0; j<numStati; j++)
632         A(i,j) = A_up(i,j) / A_down(i,j);
633
634     // aggiorna i parametri delle gaussiane
635     for(int k=0; k<mixture_vect.at(i).howmany; k++){
636
637         // aggiorna i pesi
638         mixture_vect.at(i).weight[k] = gaussians_up.at(i).weight[k] / gaussians_down.at(i).weight[k];
639
640         for(int n=0; n<3; n++){ // cicla sulle dimensioni della gaussiana (3D)
641
642             // aggiorna le medie
643             mixture_vect.at(i).components.at(k).mean[n] = gaussians_up.at(i).components.at(k).mean[n] / gaussians_down.at(i).components.at(k).mean[n];
644
645             // aggiorna le covarianze
646             // nota: si aggiornano solo quelle sulla diagonale, in quanto si lavora, per ipotesi, con matrici diagonali
647             mixture_vect.at(i).components.at(k).cov(n,n) = gaussians_up.at(i).components.at(k).cov(n,n) / gaussians_down.at(i).components.at(k).cov(n,n);
648         }
649     }
650 }
651
652 }
653
654 //debug
655 std::cout<<"Fine trainMS"<<std::endl;
656 }

```