Feature Extraction and Classification for High Dimensional Data

Chulhee Lee & David Landgrebe

TR-EE 93-1
January 1993

Appendix B

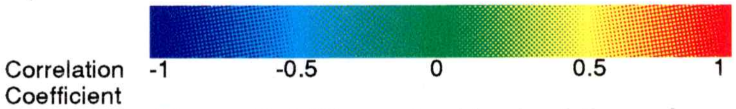

Figure B.1 The actual look of the color code.

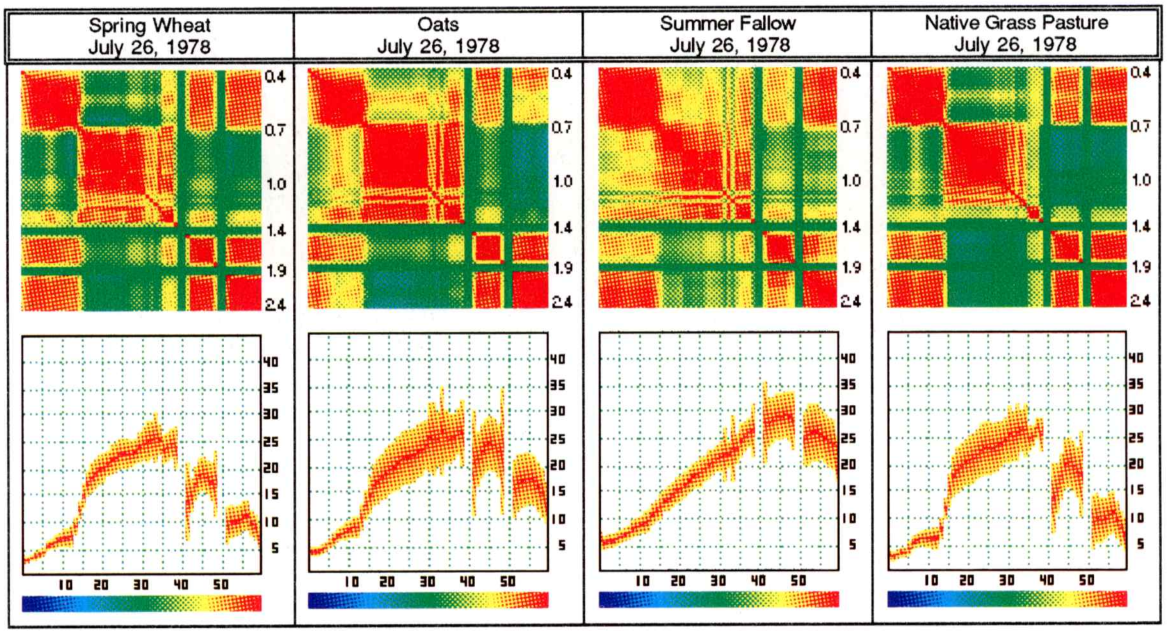

Figure B.2 Statistics images of spring wheat, oats, summer fallow, and native grass pasture on July 26, 1978.

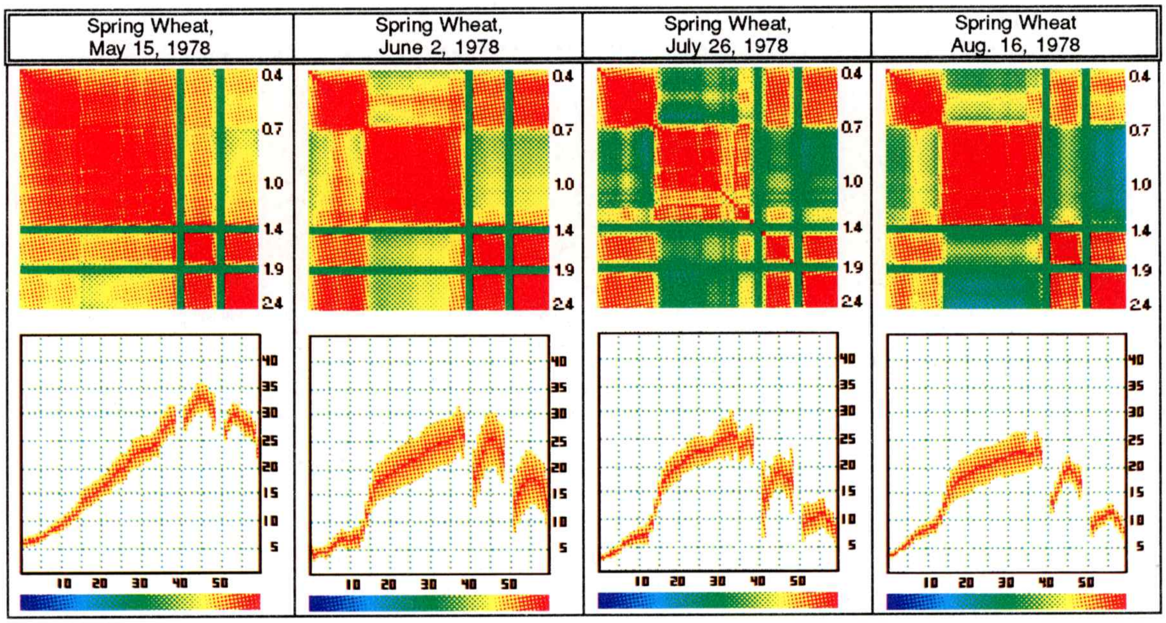

Figure B.3 Statistics images of spring wheat over 4 months period.

Appendix C
Program of Fast Likelihood Classification

```
typedef struct class_str {
            char        location[LEN_LOCATION];
            char        species[LEN_SPECIES];
            long        date;
            short       id_number;

            long        no_sample;
            long        no_training_sample;
            long        nos_tested;

            short       *cla_result;
            float       *accuracy_array;
            float       accuracies[MAX_NO_REPEAT];
            double      accuracy;

            float       *data_float1;
            float       *data_float2;

            float       *train_data;
            float       *test_data;
            float       *test_accuracy;

            short       *short_train_data;
            short       *short_test_data;

            double      *double_sumx;
            double      *double_sumxy;

            long        no_train_data;
            long        no_test_data;

            float       *parzen_result;
            double      *mean;
            double      *cov;
            double      *icov;
            double      det;
            double      log_det;
            double      square_det;

            unsigned char           *classification_np_result;
            short       *classified_as;
            float       *likelyhood_values;

} CLASS_STR;

typedef struct class_info_str {
            unsigned long           wave_length_default;

            char        name[LEN_NAME];
            char        stat_image_name[LEN_NAME];
            long        no_sample;

            float       *data_float;            /* not used */

            double      *mean;
            double      *cov;
            double      *icov;                  /* not used */
            double      det;                    /* not used */
            double      square_det;             /* not used */

            unsigned char           *half_image;
            float       *fmean;
            float       *fvar;

            long        STI_save_mode;          /* 55555     :save in file and return in memory
                                                   33333     :return in memory only
                                                   otherwise:save in file only
                                                   NOTE:STI_save_mode need to be set for every class */
            long        no_line_STI;
            long        no_col_STI;
            unsigned char           *STI;
} CLASS_INFO_STR;

/*****************************************************************/
/*******
/******* FUNCTION : sub_fast_ml                    LCH
/*******
/*******                class                    : struct(see "dfss.h")
```

```
/*******              no_class           : number of classes
/*******              actual_array_size : class->data_float2(data to be classified)
/*******                                   class->cov
/*******              no_new_channel     : number of channels (new features)
/*******              no_band_used_cla   : number of bands to be used for classification
/*******              choose_data
/*******              no_ev_used
/*******              key          0=classify all  (no truncation)
/*******                           1=classify one  (no truncation)
/*******                           2=classify one  (truncation by diff)
/**************************************************************/
sub_fast_ml(class,no_class,actual_array_size,no_new_channel,no_band_used_cla,choose_data,no_ev_used,key)
long no_new_channel,no_class,no_band_used_cla,actual_array_size,*choose_data,no_ev_used,key;
struct class_str          *class;
{
          long i,j,k;
          long para_len,max_poss_band;
          long longint;
          double *icov,*means,*detmat,*para;
          float ave_ac;

          /*****/
          /***** Absolute or Relative FML
          /*****global_fml_flag => 0:ML 1:relative fml 2:absolute fml
          /*****/
          if (global_fml_flag==1)
           {
            printf("Relative FML: Number of Truncations=%d\n",global_number_trun);
            for (i=0; i<=global_number_trun; i++)
             printf("### %d trun_at=%d trun_by=%f\n",i,global_trun_at[i],global_trun_by[i]);
           }
          else if (global_fml_flag==2)
           {
            printf("Absolute FML: Number of Truncations=%d\n",global_number_trun);
            for (i=0; i<=global_number_trun; i++)
             printf("### %d trun_at=%d trun_by=%f\n",i,global_trun_at[i],global_trun_by[i]);
           }
          else if (global_fml_flag==3)
            printf(" FML: Number of Truncations=%d\n",global_number_trun);

          /******/
          /****** allocatate memory for means, cov, detmat,para
          /******/
          longint=(long)DOUBLE*actual_array_size*no_class;
          if ((means=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for means");

          longint=(long)DOUBLE*actual_array_size*actual_array_size*no_class;
          if ((icov=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for icov");

          longint=(long)DOUBLE*no_class*actual_array_size;
          if ((detmat=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for detmat");

          para_len=actual_array_size*(actual_array_size-1);
          para_len=para_len/2+2*(actual_array_size-1)+1;
          longint=(long)DOUBLE*no_class*para_len;
          if ((para=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for para");

          /******/
          /****** make para
          /******/
          max_poss_band=no_band_used_cla;
          printf("Making parameter for FML...\n");
          for (i=0; i<no_class; i++)
          {
           for (j=0; j<no_band_used_cla; j++)
           {
```

```
       *(means+i*no_band_used_cla+j)
         = *((class+i)->mean+ j);
      }          /* for j*/

      printf("%d making parameter of %s... N=%d,\n",i,(class+i)->species,no_band_used_cla);
      make_para(no_band_used_cla,(class+i)->cov,actual_array_size,
                detmat+actual_array_size*i,para+para_len*i,&j);

      if (j<max_poss_band)
      {
       max_poss_band=j;
       printf(" Not enough training sample for %d ID:%d %s %d max_poss_band=%d\n",i,
         (class+i)->id_number,(class+i)->species,(class+i)->date,max_poss_band);
      }
     }          /* for i */

     if (global_fml_flag==3)
     {
       printf("Calculating Separability for FML3...\n");
       calculate_msc_threshold(no_class,class,actual_array_size);
     }

     no_band_used_cla=max_poss_band;
     printf("### Maximum Possible number of Features=%d\n",max_poss_band);

     /******/
     /****** do all classifications
     /******/
     printf("###    FML Started\n");
     for (i=0; i<no_class; i++)
       sub_fast_ml_per_class(i,no_class,class,no_band_used_cla,actual_array_size,
       key,para,para_len,means,detmat);

     /******/
     /****** print all classifications result
     /******/
     if (key<1)
      for (k=0; k<no_band_used_cla; k++)
      {
       for (ave_ac=i=0; i<no_class; i++)
       {
        (class+i)->accuracy=(class+i)->accuracies[k];
        for (j=0; j<no_class; j++)
         *((class+i)->cla_result+j)= *((class+i)->cla_result+k*no_class+j);
       }          /* loop i */

       ave_ac/=no_class;
       print_classification_one_result(class,no_class,k+1,choose_data,no_ev_used);
      }          /* loop k */
     else
      print_classification_one_result(class,no_class,no_band_used_cla,choose_data,no_ev_used);

     if (key<1)
      for (k=0; k<no_band_used_cla; k++)
      {
       for (ave_ac=i=0; i<no_class; i++)
        ave_ac+=(class+i)->accuracies[k];

       ave_ac/=no_class;
       printf("N=%d Average Accuracy=%.2f\n",k+1,ave_ac);
      }          /* loop k */

      free(para);
      free(icov);
      free(means);
      free(detmat);

      return;
}
```

```
/******************************************************************************/
/***
/*** FUNCTION: calculate_msc_threshold
/***
/******************************************************************************/
calculate_msc_threshold(no_class,class,no_new_channel)
long no_class,no_new_channel;
struct class_str        *class;
{
            long i,j,k,no_feature;
            double bha,threshold;

            threshold= 2*log(global_abs_fml_threshold/(1.-global_abs_fml_threshold));

            /****************************************************/
            /*******assign memory for icov
            /****************************************************/
            i=no_class*no_class*global_number_trun;
            if ((global_rmsc_threshold=(double *)malloc((unsigned)(DOUBLE*i)))==NULL)
             outerr("ERROR - Can't assign memory for global_rmsc_threshold");

            for (i=0; i<global_number_trun; i+=1)
            {
             no_feature=global_trun_at[i];
             printf("%d Calculating separability(RMSC)...N=%d Threshold=%.2f\n",
             i,no_feature,threshold);

             for (j=0; j<no_class; j++)
             {
               *(global_rmsc_threshold+no_class*no_class*i+j*no_class+j)= 20;
               for (k=0; k<j; k++)
              {
               double_bhattacharyya(&bha,(class+j)->cov,(class+j)->mean,(class+k)->cov,
               (class+k)->mean,no_feature,no_new_channel); /* Calculate bhattacharyya distance */
               *(global_rmsc_threshold+no_class*no_class*i+j*no_class+k)=
               *(global_rmsc_threshold+no_class*no_class*i+k*no_class+j)= threshold-bha;
              }
             }
            }           /* for i */
            return;
}
/*************************************************************/
/*******
/******* FUNCTION : sub_fast_ml_per_class         ICH
/*******
/*************************************************************/
sub_fast_ml_per_class(i,no_class,class,no_band_used_cla,actual_array_size,key,para,para_len,means,detmat)
struct class_str        *class;
double *para,*means,*detmat;
long i,no_class,no_band_used_cla,key,para_len,actual_array_size;
{
            long longint;
            unsigned char *result;
            long j,k;
            long no_feature,nc,correct_cl;
            double accuracy,*accuracies;

            /******/
            /****** assign memory
            /******/
            longint=(long)DOUBLE*NO_BAND_FSS;
            if ((accuracies=(double *)malloc((unsigned)longint))==NULL)
             outerr("ERROR - Can't assign memory for accuracies");

             longint=(long)CHAR*(class+i)->no_sample*no_band_used_cla;
             if ((result=(unsigned char *)malloc(longint))==NULL)
              outerr("ERROR - Can't assign memory for result 745");

            /******/
            /****** set parameters
```

```
              /******/
               no_feature=no_band_used_cla;          /*  no_feature are same */
               nc=actual_array_size;
               correct_cl=i;

              /******/
              /****** call function
              /******/
               if (key==2)                           /* 2=classify one        with truncation */
                sub_fast_ml_one_truncation(nc,no_class,no_feature,correct_cl,
                 para,para_len,means,detmat,(class+i)->data_float2,(class+i)->no_sample,
                 &((class+i)->nos_tested),result,&accuracy,actual_array_size);
               else if (key==0)        /* 0=classify all  (no truncation) */
                sub_fast_ml_all_without_truncation(nc,no_class,no_feature,correct_cl,
                para,para_len,means,detmat,(class+i)->data_float2,(class+i)->no_sample,
                &((class+i)->nos_tested),result,accuracies,actual_array_size);
               else if (key==1)        /* 1=classify one  (no truncation) */
                sub_fast_ml_one_without_truncation(nc,no_class,no_feature,correct_cl,
                para,para_len,means,detmat,(class+i)->data_float2,(class+i)->no_sample,
                &((class+i)->nos_tested),result,accuracies,actual_array_size);

              /******/
              /****** save classification results
              /******/
               if (key<1)
                for (k=0; k<no_band_used_cla; k++)
                {
                 for (j=0; j<no_class; j++)
                  *((class+i)->cla_result+k*no_class+j)=0;

                 for (j=0; j<(class+i)->no_sample; j++)
                  *( (class+i)->cla_result + k*no_class +
                   (long)*(result+k*(class+i)->no_sample+j) ) += 1;

                 (class+i)->accuracies[k]=accuracies[k];
                }          /* loop k */
               else        /* classification is done for just one feature */
                {
                 for (j=0; j<no_class; j++)
                  *((class+i)->cla_result+j)=0;

                 for (j=0; j<(class+i)->no_sample; j++)
                  *( (class+i)->cla_result + (long)*(result+j) ) += 1;

                 (class+i)->accuracy= accuracy;
                }

               free(result);
               free(accuracies);
               return;
}

/*********************************************************************************/
/***
/***    FUNCTION: sub_fast_ml_all_without_truncation
/***
/*********************************************************************************/
/*********************************************************************************/
/***
/***        <PARAMETER DESCRIPTION>
/***
/***        nc          : number of channels (MAX. 300)
/***        no_class    : number of classes
/***        no_feature  : number of channels to be used
/***        correct_cl  : correct class to be classified if available
/***        para        : parameter files. Include cov,mean,det.
/***        para_len    : length of parameter file
/***        fss_td      : data to be classified td[no_sample][nc]
/***
```

```
/***         no_sample     : number of data to be classified
/***         result        : unsigned char array which will contain the results
/***                         (unsigned char result[no_sample])
/***         accuracy      : classification accuracy
/***         actual_array_size_td :
/***
/*****************************************************************************/
sub_fast_ml_all_without_truncation(nc,no_class,no_feature,correct_cl,para,para_len,mean,detmat,
fss_td,no_sample,nos,result,accuracies,actual_array_size_td)
long nc,no_class,correct_cl,no_feature,actual_array_size_td,para_len;
long no_sample,*nos;
double *para,*mean,*detmat,*accuracies;
float *fss_td;
unsigned char *result;
{
           long i,j,k,l;
           register double *tpara,*meanf;
           register float *pt2;

           long longint;
           long error[MAX_NO_REPEAT],tres,nos1;
           double tmax,*temp,*templ,dt,dt1,tm;

           for (i=0; i<no_feature; i++)
            error[i]=0;

           /******/
           /****** assign memory
           /******/
           longint=(long)DOUBLE*no_feature*no_class;
           if ((temp=(double *)malloc(longint))==NULL)
            outerr("ERROR - Can't assign memory for temp");

           /******/
           /****** classify
           /******/
            for (i=nos1=0; i<no_sample; i++,nos1++)
            {
             pt2= fss_td+actual_array_size_td*i;

             for (j=0; j<no_class; j++)
             {
               tpara= para+para_len*j;
               meanf= mean+nc*j;
               templ=temp+no_feature*j;

               tm= *pt2- *meanf;
               *(templ+0)= tm*tm* *(tpara++);     /* start point */

                       for (k=1; k<no_feature; k++)
                       {
                        tm= *(pt2+k)- *(meanf+k);
                        dt = *(templ+k-1);        /* previous value */

                        for (dt1=l=0; l<k; l++)
                         dt1 += (*(pt2+l)- *(meanf+l))* *(tpara++);

                        dt += *(tpara++)*(tm*(tm-dt1-dt1)+dt1*dt1);

                        *(templ+k)=dt;
                       } /* loop k */

             }         /* loop j */


            /******/
            /****** classify up to no_feature
            /******/
             for (k=0; k<no_feature; k++)
             {
```

```
             tmax= -1e30;
             for (j=0; j<no_class; j++)
              if (- *(temp+no_feature*j+k)- *(detmat+nc*j+k)>tmax)
              {
               tmax= - *(temp+no_feature*j+k)- *(detmat+nc*j+k);
               tres=j;
              }

                      if (correct_cl != tres)
                       (error[k])++;

                      *(result+k*no_sample+i)=(unsigned char) tres;
            }         /* loop k */

           }                                 /* loop i=no_sample */


           /******/
           /****** calculate accuracy
           /******/
           for (k=0; k<no_feature; k++)
           {
            *(accuracies+k)= 100.*(no_sample-error[k])/no_sample;
            printf("correct_cl=%d no_feature=%d accuracy=%5.1f\n",correct_cl,k+1,*(accuracies+k));
           }
           free(temp);
           *nos=nos1;

           return;
}

/*******************************************************/
/******
/****** FUNCTION: sub_fast_ml_one_truncation
/******
/*******************************************************/
sub_fast_ml_one_truncation(nc,no_class,no_feature,correct_cl,para,para_len,
mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td)
long nc,no_class,correct_cl,actual_array_size_td,para_len;
long no_sample,*nos;
double *para,*mean,*detmat,*accuracy;
float *fss_td;
unsigned char *result;
{
            if (global_fml_flag==1)            /* truncation by absolute difference */
            {
             sub_fast_ml_by_likely_diff_rel_gen(nc,no_class,no_feature,correct_cl,para,para_len,
                      mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td);
            }
            else if (global_fml_flag==2)       /* truncation by absolute region ####aFML#### */
            {
             sub_fast_ml_by_likely_diff_abs_gen(nc,no_class,no_feature,correct_cl,para,para_len,
                      mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td);
            }
            else if (global_fml_flag==3)       /* truncation by relative difference considering class
                                                  separability ####rFML#### */
            {
             sub_fast_ml_by_rel_diff_rel_gen(nc,no_class,no_feature,correct_cl,para,para_len,
                      mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td);
            }
            else
            {
             printf("global_fml_flag=%d\n",global_fml_flag);
             outerr("global_fml_flag ERROR");
            }

            return;
}

/*******************************************************/
```

```
/******
/****** FUNCTION: sub_fast_ml_by_likely_diff_rel_gen
/******
/******                  relative FML
/******
/*******************************************************/
sub_fast_ml_by_likely_diff_rel_gen(nc,no_class,no_feature,correct_cl,para,para_len,
mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td)
long nc,no_class,correct_cl,actual_array_size_td,para_len;
long no_sample,*nos;
double *para,*mean,*detmat,*accuracy;
float *fss_td;
unsigned char *result;
{

            long i,j,k,l,kl;
            register double *tpara,*meanf;
            register float *pt2;

            long longint;
            long ierr,tres;
            double tmax,*temp,*tmp_val,*temp1,dt,dt1,tm;
            double *temp7,*detmat7;

            long j1,no_keep1,no_keep2;
            long detindex,para_len_offset,*index_keeps;

            long incre,start;
            double *keeps;

            if (no_sample==0)
            {
             printf("No test sample... Just returning... Correct_cl=%d\n",correct_cl);
             return;
            }

            /************************************************/
            /***  set truncation parameter */
            /************************************************/
            global_trun_at[global_number_trun]=no_feature;
            ierr=0;

            /************************************************/
            /******* assign memory
            /************************************************/
            longint=(long)DOUBLE*no_class;
            if ((tmp_val=(double *)malloc(longint))==NULL)
             outerr("ERROR - Can't assign memory for tmp_val");

            longint=(long)DOUBLE*no_feature*no_class;
            if ((temp=(double *)malloc(longint))==NULL)
             outerr("ERROR - Can't assign memory for temp");

            longint=(long)DOUBLE*no_class;
            if ((keeps=(double *)malloc(longint))==NULL)
             outerr("ERROR - Can't assign memory for keeps");

            longint=(long)INT*no_class;
            if ((index_keeps=(long *)malloc(longint))==NULL)
             outerr("ERROR - Can't assign memory for index_keeps");

            /************************************************/
            /***  calculate discriminant functions */
            /************************************************/
            start=test_start;
            incre=test_incre;
            *nos=0;

             for (i=start; i<no_sample; i=i+incre)
             {
```

```
pt2= fss_td+actual_array_size_td*i;

/************************************************/
/***  initialization */
/************************************************/
no_keep1=no_class;

for (j=0; j<no_class; j++)          /* start point */
{
  *(index_keeps+j)=j;
  tm= *pt2- *(mean+nc*j);
  *(temp+no_feature*j)= tm*tm* *(para+para_len*j);
}          /* loop j */
para_len_offset=1;


for (k1=0; k1<global_number_trun; k1++)
{
/************************************************/
/***  calculate discriminant functions */
/************************************************/
for (j1=0,j= *(index_keeps); j1<no_keep1; j= *(index_keeps+ ++j1))
{
  tpara=para+para_len*j+para_len_offset;
  meanf=mean+nc*j;
  templ=temp+no_feature*j;

           for (k=global_trun_at[k1]; k<global_trun_at[k1+1]; k++)
           {
            tm= *(pt2+k)- *(meanf+k);
            dt = *(templ+k-1);       /* previous value */

            for (dt1=l=0; l<k; l++)
             dt1 += (*(pt2+l)- *(meanf+l))* *(tpara++);

            dt += *(tpara++)*(tm*(tm-dt1-dt1)+dt1*dt1);

            *(templ+k)=dt;
           } /* loop k */

}          /* loop j */

/************************************************/
/***  classify and truncate
/************************************************/
tmax= -1e30;
detindex= global_trun_at[k1+1]-1;
no_keep2=no_keep1;

temp7=temp+detindex;
detmat7=detmat+detindex;
for (j1=0,j= *(index_keeps); j1<no_keep2; j= *(index_keeps+ ++j1))
 if (tmax < (tmp_val[j]= - *(temp7+no_feature*j)- *(detmat7+nc*j)))
 {
  tres=j;
  tmax = tmp_val[j];
 }
if (k1+1==no_feature)
 break;

 no_keep1=0;
 tmax -= global_trun_by[k1+1];

 for (j1=0,j= *(index_keeps); j1<no_keep2; j= *(index_keeps+ ++j1))
 if (tmax < tmp_val[j])
  *(index_keeps+no_keep1++)=j;

          /* check only one class left */
 if (no_keep1==1)
  break;
```

```
            /*  para_len_offset=15*14/2+15; */
            para_len_offset= global_trun_at[k1+1]*(global_trun_at[k1+1]-1)/2+ global_trun_at[k1+1];

          }          /* loop k1 */

            if (correct_cl != tres)
                    ierr++;
            (*nos)++;

                    if (global_chi_threshold_prob<=0)     /* check threshold */
                     *(result+i)=(unsigned char) tres;
                    else if (global_chi_threshold[detindex]>tmax+ *(detmat7+nc*tres))
                     *(result+i)=(unsigned char) no_class;
                    else
                     *(result+i)=(unsigned char) tres;
          }                                   /* loop i */

          *accuracy=(*nos - ierr)*100;          /* accuracy */
          *accuracy /= *nos;
          printf("rFML nos tested:%d correct_cl:%d accuracy=%4.1f\n",*nos,correct_cl,*accuracy);

          /*******************************************************/
          /******* check saving classfication result(map)
          /*******************************************************/
          check_save_result_TM(result,no_sample,no_class);

          free(temp);
          free(index_keeps);
          free(keeps);

          return;
}


/*******************************************************/
/******
/****** FUNCTION: sub_fast_ml_by_likely_diff_abs_gen
/******
/******                 absolute FML
/******
/*******************************************************/
sub_fast_ml_by_likely_diff_abs_gen(nc,no_class,no_feature,correct_cl,para,para_len,
mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td)
long nc,no_class,correct_cl,actual_array_size_td,para_len;
long no_sample,*nos;
double *para,*mean,*detmat,*accuracy;
float *fss_td;
unsigned char *result;
{
          long i,j,k,l,k1,trun_each_state[10];
          register double *tpara,*meanf;
          register float *pt2;

          long longint;
          long ierr,tres;
          double amax,tmax,*temp,*temp1,dt,dt1,tm;
          double *temp7;

          long j1,no_keep1,no_keep2,tres_bak;
          long detindex,para_len_offset,*index_keeps;

          long flag_correct_cl_truncated;
          double *keeps;

          /*****/
          /*****  check number of samples
          /*****/
          if (no_sample==0)
          {
```

```
 printf("No test sample... Just return Correct_cl=%d\n",correct_cl);
 return;
}

/*****/
/*****  set truncation parameter
/*****/
global_trun_at[global_number_trun]=no_feature;

ierr=0;
for (i=0; i<10; i++)
 trun_each_state[i]=0;

/*****/
/*****  assign memory
/*****/
longint=(long)DOUBLE*no_feature*no_class;
if ((temp=(double *)malloc(longint))==NULL)
 outerr("ERROR - Can't assign memory for temp 234");

longint=(long)DOUBLE*no_class;
if ((keeps=(double *)malloc(longint))==NULL)
 outerr("ERROR - Can't assign memory for keeps 462");

longint=(long)INT*no_class;
if ((index_keeps=(long *)malloc(longint))==NULL)
 outerr("ERROR - Can't assign memory for index_keeps 62b");

/*****/
/*****  classify
/*****/
for (*nos=i=0; i<no_sample; i++)
 {
  pt2= fss_td+actual_array_size_td*i;

  /*****/
  /*****  initialization
  /*****/
  no_keep1=no_class;

  for (j=0; j<no_class; j++)          /* start point */
  {
    *(index_keeps+j)=j;
    tm= *pt2- *(mean+nc*j);
    *(temp+no_feature*j)= tm*tm* *(para+para_len*j);
  }          /* loop j */
  para_len_offset=1;


  for (k1=0; k1<global_number_trun; k1++)
  {
  /*****/
  /*****  calculate discriminant functions
  /*****/
  for (j1=0,j= *(index_keeps); j1<no_keep1; j= *(index_keeps+ ++j1))
  {
    tpara= para+para_len*j+para_len_offset;
    meanf= mean+nc*j;
    temp1= temp+no_feature*j;

             for (k=global_trun_at[k1]; k<global_trun_at[k1+1]; k++)
             {
              tm= *(pt2+k)- *(meanf+k);
              dt = *(temp1+k-1);       /* previous value */

              for (dt1=l=0; l<k; l++)
               dt1 += (*(pt2+l)- *(meanf+l))* *(tpara++);

              dt += *(tpara++)*(tm*(tm-dt1-dt1)+dt1*dt1);
```

```
            *(temp1+k)=dt;
           } /* loop k */

 }         /* loop j1 */

 /*****/
 /*****  classify and truncate
 /*****/
 amax = global_trun_by[k1+1];       /* absolute value */
 detindex= global_trun_at[k1+1]-1;
 no_keep2=no_keep1;
 no_keep1=0;
 temp7=temp+detindex;

 tres_bak=j= *(index_keeps);
 flag_correct_cl_truncated=0;
 for (no_keep1=j1=0; j1<no_keep2; j= *(index_keeps+ ++j1))
 {
  if (amax < - *(temp7+no_feature*j))           /* threshold depends only on r */
   *(index_keeps+no_keep1++)=j;

  /* count no. truncation of correct class at each stage */
  if (j==correct_cl && amax >= - *(temp7+no_feature*j))
  {
   flag_correct_cl_truncated=1;
   trun_each_state[k1]+=1;
  }
 }                       /* for no_keep1  */

 if (no_keep1<=1)        /* only one class is left */
     break;
  /*  para_len_offset=15*14/2+15; */
  para_len_offset= global_trun_at[k1+1]*(global_trun_at[k1+1]-1)/2+ global_trun_at[k1+1];

}           /* loop k1 */

 /*****/
 /***** make comparson outside the loop
 /*****/
 if (no_keep1==1)
  *(index_keeps)=tres_bak;           /* restore the first class */
 tmax= -1e30;
 for (j1=0,j= *(index_keeps); j1<no_keep2; j= *(index_keeps+ ++j1))
  if (tmax < (dt1= - *(temp7+no_feature*j)- *(detmat+nc*j+detindex)))
  {
   tres=j;
   tmax = dt1;
  }

  if (correct_cl != tres)
            ierr++;

  if (correct_cl != tres && flag_correct_cl_truncated==1)
   trun_each_state[k1]-=1;           /* though truncated, classified correctly */

  (*nos)++;

            if (global_chi_threshold_prob<=0)     /* check threshold */
             *(result+i)=(unsigned char) tres;
            else if (global_chi_threshold[detindex]>tmax+ *(detmat+nc*tres+detindex))
             *(result+i)=(unsigned char) no_class;
            else
             *(result+i)=(unsigned char) tres;

}                                   /* loop i */

*accuracy=(*nos - ierr)*100;        /* accuracy */
*accuracy /= *nos;
printf("aFML nos tested:%d correct_cl:%d accuracy=%.4f\n",*nos,correct_cl,*accuracy);
```

```
          /********************************************************/
          /******* check saving classfication result(map)
          /********************************************************/
          check_save_result_TM(result,no_sample,no_class);

          free(temp);
          free(index_keeps);
          free(keeps);

          return;
}
/********************************************************/
/******
/****** FUNCTION: sub_fast_ml_by_rel_diff_rel_gen
/******
/******                    relative FML
/******
/********************************************************/
sub_fast_ml_by_rel_diff_rel_gen(nc,no_class,no_feature,correct_cl,para,para_len,
mean,detmat,fss_td,no_sample,nos,result,accuracy,actual_array_size_td)
long nc,no_class,correct_cl,actual_array_size_td,para_len;
long no_sample,*nos;
double *para,*mean,*detmat,*accuracy;
float *fss_td;
unsigned char *result;
{
          long i,j,k,l,kl;
          register double *tpara,*meanf;
          register float *pt2;
          long longint;
          long ierr,tres;
          double tmax,*temp,*tmp_val,*temp1,dt,dt1,tm;
          double *temp7,*detmat7,*tmp_threshold;
          long j1,no_keep1,no_keep2;
          long detindex,para_len_offset,*index_keeps;
          double *keeps;

          if (no_sample==0)
          {
           printf("No test sample... Just returning... Correct_cl=%d\n",correct_cl);
           return;
          }

          /**************************************************/
          /***  set truncation parameter */
          /**************************************************/
          global_trun_at[global_number_trun]=no_feature;
          ierr=0;

          /********************************************************/
          /******* eigenvectors & eigenvalues and feaband
          /********************************************************/

          longint=(long)DOUBLE*no_class;
          if ((tmp_val=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for tmp_val");

          longint=(long)DOUBLE*no_feature*no_class;
          if ((temp=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for temp 63");

          longint=(long)DOUBLE*no_class;
          if ((keeps=(double *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for keeps 94f");

          longint=(long)INT*no_class;
          if ((index_keeps=(long *)malloc(longint))==NULL)
           outerr("ERROR - Can't assign memory for index_keeps 8j");
```

```
/*************************************************/
/***  calculate discriminant functions */
/*************************************************/
for (*nos=i=0; i<no_sample; i++)
 {
  pt2= fss_td+actual_array_size_td*i;

  /*************************************************/
  /***  initialization */
  /*************************************************/
  no_keep1=no_class;

  for (j=0; j<no_class; j++)          /* start point */
  {
    *(index_keeps+j)=j;
    tm= *pt2- *(mean+nc*j);
    *(temp+no_feature*j)= tm*tm* *(para+para_len*j);
  }          /* loop j */
  para_len_offset=1;


  for (k1=0; k1<global_number_trun; k1++)
  {
  if (k1==0)
  {
   global_sum_no_feature += global_trun_at[k1+1]*no_keep1;
   global_sum_operation += global_trun_at[k1+1]*global_trun_at[k1+1]*no_keep1;
  }
  else
  {
   global_sum_operation += (global_trun_at[k1+1]*global_trun_at[k1+1]
                            -global_trun_at[k1]*global_trun_at[k1])*no_keep1;
   global_sum_no_feature += (global_trun_at[k1+1]-global_trun_at[k1])*no_keep1;
  }

  /*************************************************/
  /***  calculate discriminant functions */
  /*************************************************/
  for (j1=0,j= *(index_keeps); j1<no_keep1; j= *(index_keeps+ ++j1))
  {
    tpara=para+para_len*j+para_len_offset;
    meanf=mean+nc*j;
    temp1=temp+no_feature*j;

             for (k=global_trun_at[k1]; k<global_trun_at[k1+1]; k++)
             {
              tm= *(pt2+k)- *(meanf+k);
              dt = *(temp1+k-1);       /* previous value */

              for (dt1=l=0; l<k; l++)
               dt1 += (*(pt2+l)- *(meanf+l))* *(tpara++);

              dt += *(tpara++)*(tm*(tm-dt1-dt1)+dt1*dt1);

              *(temp1+k)=dt;
             } /* loop k */

  }          /* loop j */

  /*************************************************/
  /***  classify and truncate
  /*************************************************/
  tmax= -1e30;
  detindex= global_trun_at[k1+1]-1;
  no_keep2=no_keep1;

  temp7=temp+detindex;
  detmat7=detmat+detindex;
  for (j1=0,j= *(index_keeps); j1<no_keep2; j= *(index_keeps+ ++j1))
   if (tmax < (tmp_val[j]= - *(temp7+no_feature*j)- *(detmat7+nc*j)))
```

```
          {
           tres=j;
           tmax = tmp_val[j];
          }
         if (k1+1==no_feature)
          break;

          no_keep1=0;
          tmp_threshold= global_rmsc_threshold+(k1*no_class+tres)*no_class;

          for (j1=0,j= *(index_keeps); j1<no_keep2; j= *(index_keeps+ ++j1))
          if (tmax - *(tmp_threshold+j) < tmp_val[j])
           *(index_keeps+no_keep1++)=j;

                    /* check only one class left */
          if (no_keep1==1)
           break;

          /*  para_len_offset=15*14/2+15; */
          para_len_offset= global_trun_at[k1+1]*(global_trun_at[k1+1]-1)/2+ global_trun_at[k1+1];

        }           /* loop k1 */

          if (correct_cl != tres)
                     ierr++;
          (*nos)++;

                    if (global_chi_threshold_prob<=0)      /* check threshold */
                     *(result+i)=(unsigned char) tres;
                    else if (global_chi_threshold[detindex]>tmax+ *(detmat7+nc*tres))
                     *(result+i)=(unsigned char) no_class;
                    else
                     *(result+i)=(unsigned char) tres;
        }                                   /* loop i */

        *accuracy=(*nos - ierr)*100;        /* accuracy */
        *accuracy /= *nos;
        printf("rFML3 nos tested:%d correct_cl:%d accuracy=%4.1f\n",*nos,correct_cl,*accuracy);

        free(temp);
        free(index_keeps);
        free(keeps);
        return;
}
```

Appendix D
Program of Decision Boundary Feature Extraction for Gaussian ML Classifier

```
/********************************************************************/
/***
/***        FUNCTION: FS_decision_boundary (36)
/***
/***        INPUT:
/***                    no_feature(long):        number of features(=N)
/***                    no_class(long):          number of classes
/***                    class_info(struct):     class informations (See Appendix C)

/***        OUTPUT:
/***                    eigenvectors(double, N by N)          eigenvectors
/***                    eigenvectors(double, N)               eigenvalues
/***                    ERROR_FLAG(*long)                     error flag. must be zero.
/***
/********************************************************************/
FS_decision_boundary(class_info,no_class,no_feature,eigen_vectors,eigen_values,ERROR_FLAG)
long *ERROR_FLAG;
long no_class,no_feature;
struct class_info_str *class_info;
double *eigen_vectors,*eigen_values;
{
            double *edbfm,*EV_edbfm,*E_value_edbfm,*edbfm_all;
            char fname[100];
            long i,j,k,class_index[2];
            double accuracy;
            long longint;

            /*******/
            /******* assign memory
            /*******/
            longint=no_feature*no_feature*FS_DOUBLE;

            edbfm=(double *)SI_create_memory(longint);
            if (edbfm==FS_NULL)
             *ERROR_FLAG=507;

            edbfm_all=(double *)SI_create_memory(longint);
            if (edbfm_all==FS_NULL)
             *ERROR_FLAG=508;

            EV_edbfm=(double *)SI_create_memory(longint);
            if (EV_edbfm==FS_NULL)
             *ERROR_FLAG=509;

            E_value_edbfm=(double *)SI_create_memory(longint);
            if (E_value_edbfm==FS_NULL)
             *ERROR_FLAG=510;

            /*******/
            /******* check error
            /*******/
            if (*ERROR_FLAG>0)
            {
             SI_free_memory((char **)&E_value_edbfm);
             SI_free_memory((char **)&EV_edbfm);
             SI_free_memory((char **)&edbfm_all);
             SI_free_memory((char **)&edbfm);

             return;
            }
```

```
        /*******/
        /******* find edbfm of each pair of classes
        /*******/
        for (i=0; i<no_feature*no_feature; i++)          /* initialize */
         edbfm_all[i]=0;

        for (i=0; i<no_class; i++)
         for (j=0; j<i; j++)
         {
          class_index[0]=i;
          class_index[1]=j;

          FS_sub_find_edbfm_2_class(class_info,class_index,2,no_feature,edbfm,&accuracy,ERROR_FLAG);


          FS_optimize_2_class(class_info,class_index,2,no_feature,edbfm,accuracy,ERROR_FLAG);

          for (k=0; k<no_feature*no_feature; k++)
           edbfm_all[k]+=edbfm[k];
         }

        /*******/
        /******* calculate eigenvectors and eigenvalues of edbfm_all
        /*******/
        for (j=0; j<no_feature*no_feature; j++)
         E_value_edbfm[j]=edbfm_all[j];
        deigen(no_feature,E_value_edbfm,2,&i,EV_edbfm,no_feature);

        /*******/
        /******* copy eigen_vectors and eigen_value
        /*******/
        for (i=0; i<no_feature; i++)
         for (j=0; j<no_feature; j++)
         {
           eigen_values[i]=E_value_edbfm[i*no_feature+i];
           eigen_vectors[i*no_feature+j]=EV_edbfm[i*no_feature+j];
         }

        /*******/
        /******* free memory
        /*******/
        SI_free_memory((char **)&E_value_edbfm);
        SI_free_memory((char **)&EV_edbfm);
        SI_free_memory((char **)&edbfm_all);
        SI_free_memory((char **)&edbfm);

        return;
}
/*******************************************************/
/*******
/*******      FUNCTION: FS_optimize_2_class
/*******
/*******************************************************/
FS_optimize_2_class(class_info,class_index,no_class,no_new_channel,edbfm,accuracy,
ERROR_FLAG)
long *ERROR_FLAG;
struct class_info_str *class_info;
long *class_index,no_new_channel,no_class;
double *edbfm,accuracy;
{
        register double *fmean,*icov;
        long i2,i,j,k,a,b,err[2];
        double       class_min[2],log_det,op_threshold;
        long error=0;
        long longint;

        double *mean,*EV,*eval,*icov_all,*cov,*ficov,mah[2],t1,t3;
        double det[2];
        float *td,*dfloat[2],acc[2];
```

```
long total_sample;

/*******/
/******* init threshold
/*******/
op_threshold=0.97;

/*******/
/******* assign memory & calculate eigenvalue of edbfm
/*******/
EV=(double *)SI_create_memory((unsigned long)no_new_channel*no_new_channel*FS_DOUBLE);
if (EV==FS_NULL)
 *ERROR_FLAG=511;

eval=(double *)SI_create_memory((unsigned long)no_new_channel*no_new_channel*FS_DOUBLE);
if (eval==FS_NULL)
 *ERROR_FLAG=512;

if (*ERROR_FLAG==0)
{
 for (i=0; i<no_new_channel*no_new_channel; i++)
  eval[i]=edbfm[i];

 deigen(no_new_channel,eval,2,&i,EV,no_new_channel);
}

icov_all=(double *)
SI_create_memory((unsigned long)no_class*no_new_channel*no_new_channel*FS_DOUBLE);
if (icov_all==FS_NULL)
 *ERROR_FLAG=513;

cov=(double *)
SI_create_memory((unsigned long)no_class*no_new_channel*no_new_channel*FS_DOUBLE);
if (cov==FS_NULL)
 *ERROR_FLAG=514;

mean=(double *)SI_create_memory((unsigned long)no_class*no_new_channel*FS_DOUBLE);
if (mean==FS_NULL)
 *ERROR_FLAG=515;

/*******/
/******* assign memory & copy data
/*******/
if (*ERROR_FLAG==0)       /* continue if no error occurred */
for (total_sample=i=0; i<no_class; i++)
{
 total_sample+= (class_info+class_index[i])->no_sample;

 longint=(long)FS_FLOAT*no_new_channel*(class_info+class_index[i])->no_sample;
 dfloat[i]=(float *)SI_create_memory((unsigned)longint);
 if (dfloat==FS_NULL)
 {
  *ERROR_FLAG=516;
  break;
 }

 FS_sub_linear_transform(dfloat[i],(class_info+class_index[i])->data_float,
 EV,no_new_channel,no_new_channel,(class_info+class_index[i])->no_sample);

 cal_stat_fdata(dfloat[i],no_new_channel,
  (class_info+class_index[i])->no_sample,
 mean+i*no_new_channel,cov+i*no_new_channel*no_new_channel,no_new_channel);
}

if (*ERROR_FLAG==0) /*################ continue if no error occurred */
{

/*******/
/******* classsify
/*******/
for (i2=1; i2<=no_new_channel; i2++)
{
 for (j=0; j<no_class*no_new_channel*no_new_channel; j++)
  icov_all[j]=cov[j];
 for (i=0; i<no_class; i++)
 {
 FS_inverse_log_det(icov_all+no_new_channel*no_new_channel*i,i2,
 no_new_channel,det+i,&log_det);

 det[i]=log(det[i]);
 }

err[0]=err[1]=0;
for (i=0; i<no_class; i++)
{
 /*******/
 /******* do classification
 /*******/
 for (j=0; j<(class_info+class_index[i])->no_sample; j++)
 {
  td=dfloat[i]+j*no_new_channel;
  for (k=0; k<no_class; k++)
  {
             fmean=mean+no_new_channel*k;
             icov=icov_all+no_new_channel*no_new_channel*k;

             for (mah[k]=a=0; a<i2; a++)
             {
                          ficov = icov+no_new_channel*a;
         t3= *(td+a)- *(fmean+ a);

         for (t1=b=0; b<a; b++)
          t1 -= (*(td+ b)- *(fmean+ b))* *(ficov+ b);
         mah[k] += t3*(t1+t1-t3* *(ficov+ a));
    }
  }          /* for k */

             /*******/
             /******* check error
             /*******/
    if (mah[i]-det[i]<mah[1-i]-det[1-i])
     err[i]++;
  }          /* for  j */
 }           /* for  i */

error=err[0]+err[1];

if (100.*(total_sample-error)/total_sample>=op_threshold*accuracy)
 break;
} /* i2 */

/*******/
/******* Making new EDBFM
/*******/
for (i=0; i<no_new_channel*no_new_channel; i++)
 edbfm[i]=0;

for (i=0; i<i2; i++)
{
 for (j=0; j<no_new_channel; j++)
  for (k=0; k<no_new_channel; k++)
   *(edbfm+j*no_new_channel+k) += *(EV+j*no_new_channel+i)*
                                  *(EV+k*no_new_channel+i)* eval[i*no_new_channel+i];
}

} /*################################################ if (*ERROR_FLAG==0) */

/*******/
/******* free menory
/*******/
```

```
        SI_free_memory((char **)&dfloat[1]);
        SI_free_memory((char **)&dfloat[0]);

        SI_free_memory((char **)&mean);
        SI_free_memory((char **)&cov);
        SI_free_memory((char **)&icov_all);
        SI_free_memory((char **)&eval);
        SI_free_memory((char **)&EV);

        return;
}

/********************************************************/
/*******
/*******        FUNCTION: FS_sub_find_edbfm_2_class
/*******
/********************************************************/
typedef struct cl_res_info_str {
        long            counter;
        double          threshold2;
        float           *classified;
        long            *classified_as;
        short           *sflag;
        long            no_passed_mah_dis;
        float           *mah_dis;
        long            no_passed_mah_dis_the_other;
        float           *mah_dis_the_other;
} CL_RES_INFO_STR;
FS_sub_find_edbfm_2_class(class_info,class_index,no_class,no_new_channel,edbfm,accuracy,
ERROR_FLAG)
long *ERROR_FLAG;
struct class_info_str *class_info;
long *class_index,no_new_channel,no_class;
double *edbfm,*accuracy;
{
        struct cl_res_info_str cl_res[2];
        register double *fmean,*icov;
        long i,j,k,l,a,b,cnt_tbl[5],point_array_cnt,flag,err[2];
        double          *point1_array,*point2_array,dmin,tmp,class_min[2],log_det;
        float *tp1,*tp2;
        long imin,error=0,minimum;
        long longint;

        double *icov_all,*ficov,mah[2],t1,t3;
        double threshold2,threshold_prob2,threshold,threshold_prob=.995,det[2];
        float *td,fmax;
        long total_sample;

        /*******/
        /******* assign memory
        /*******/
        for (total_sample=i=0; i<no_class; i++)
         total_sample+=(class_info+class_index[i])->no_sample;

        longint=(long)FS_DOUBLE*no_new_channel*total_sample;
        point1_array=(double *)SI_create_memory((unsigned)longint);
        if (point1_array==FS_NULL)
         *ERROR_FLAG=517;

        point2_array=(double *)SI_create_memory((unsigned)longint);
        if (point2_array==FS_NULL)
         *ERROR_FLAG=518;

        longint=(long)sizeof(long)*no_new_channel*total_sample;
        (cl_res+0)->classified_as=(long *)SI_create_memory((unsigned)longint);
        if ((cl_res+0)->classified_as==FS_NULL)
         *ERROR_FLAG=519;

        (cl_res+1)->classified_as=(long *)SI_create_memory((unsigned)longint);
        if ((cl_res+1)->classified_as==FS_NULL)
```

```
 *ERROR_FLAG=520;

longint=(long)FS_FLOAT*total_sample;
(cl_res+0)->mah_dis=(float *)SI_create_memory((unsigned)longint);
if ((cl_res+0)->mah_dis==FS_NULL)
 *ERROR_FLAG=521;

(cl_res+1)->mah_dis=(float *)SI_create_memory((unsigned)longint);
if ((cl_res+1)->mah_dis==FS_NULL)
 *ERROR_FLAG=522;

(cl_res+0)->mah_dis_the_other=(float *)SI_create_memory((unsigned)longint);
if ((cl_res+0)->mah_dis_the_other==FS_NULL)
 *ERROR_FLAG=523;

(cl_res+1)->mah_dis_the_other=(float *)SI_create_memory((unsigned)longint);
if ((cl_res+1)->mah_dis_the_other==FS_NULL)
 *ERROR_FLAG=524;

longint=(long)sizeof(short)*total_sample;
(cl_res+0)->sflag=(short *)SI_create_memory((unsigned)longint);
if ((cl_res+0)->sflag==FS_NULL)
 *ERROR_FLAG=525;

(cl_res+1)->sflag=(short *)SI_create_memory((unsigned)longint);
if ((cl_res+1)->sflag==FS_NULL)
 *ERROR_FLAG=526;

icov_all=(double *)
SI_create_memory((unsigned long)no_class*no_new_channel*no_new_channel*FS_DOUBLE);
if (icov_all==FS_NULL)
 *ERROR_FLAG=527;

if(*ERROR_FLAG==0) /*################ continue if no error occurred */
{
/*******/
/******* calculate inverse of cov
/*******/
for (i=0; i<no_class; i++)
{
 for (j=0; j<no_new_channel*no_new_channel; j++)
  icov_all[no_new_channel*no_new_channel*i+j]= (class_info+class_index[i])->cov[j];
 FS_inverse_log_det(icov_all+no_new_channel*no_new_channel*i,no_new_channel,
 no_new_channel,det+i,&log_det);

 det[i]=log(det[i]);
}

/*******/
/******* fine threshold to remove outliers (self)
/*******/
threshold_prob=.95;
sub_find_r_threshold(no_new_channel,&threshold,threshold_prob,(double)0.1);
threshold *= -threshold;

/*******/
/******* fine threshold to remove outliers (the other class)
/*******/
threshold_prob2=.95;
sub_find_r_threshold(no_new_channel,&threshold2,threshold_prob2,(double)0.1);
threshold2 *= -threshold2;

/*******/
/******* classsify
/*******/
err[0]=err[1]=(cl_res+0)->counter=(cl_res+1)->counter=0;
for (i=0; i<no_class; i++)
{
 /*******/
 /******* do classification
```

```
/*******/
for (j=0; j<(class_info+class_index[i])->no_sample; j++)
{
 td=(class_info+class_index[i])->data_float+j*no_new_channel;

 for (k=0; k<no_class; k++)
 {
           fmean=(class_info+class_index[k])->mean;
           icov=icov_all+no_new_channel*no_new_channel*k;

           for (mah[k]=a=0; a<no_new_channel; a++)
           {
                        ficov = icov+no_new_channel*a;
       t3= *(td+a)- *(fmean+ a);

       for (t1=b=0; b<a; b++)
        t1 -= (*(td+ b)- *(fmean+ b))* *(ficov+ b);
       mah[k] += t3*(t1+t1-t3* *(ficov+ a));
   }

   if (i==k)
   {
    (cl_res+i)->mah_dis[j]=mah[k];
    (cl_res+i)->mah_dis_the_other[j]=mah[1-k];
   }
   else
   {
    (cl_res+i)->mah_dis[j]=mah[1-k];
    (cl_res+i)->mah_dis_the_other[j]=mah[k];
   }

 }         /* for k */

           /*******/
           /******* check error
           /*******/
   if (mah[i]-det[i]<mah[1-i]-det[1-i])
   {
    err[i]++;
            (cl_res+i)->classified_as[j]=1-i;
   }
   else
   {
            (cl_res+i)->classified_as[j]=i;
           }
 }         /* for  j */
}          /* for  i */

error=err[0]+err[1];

*accuracy=100.*(total_sample-error)/total_sample;

/*******/
/******* count number of samples passing the threshold test
/*******/
for (i=0; i<2; i++)
{
 (cl_res+i)->no_passed_mah_dis_the_other=(cl_res+i)->no_passed_mah_dis=0;
 fmax= -1e30;

 for (j=0; j<(class_info+class_index[i])->no_sample; j++)      /* 333 */
 if ((cl_res+i)->classified_as[j]==i && (cl_res+i)->mah_dis[j]>threshold)
 {
  if ((cl_res+i)->mah_dis[j]>threshold)
   (cl_res+i)->no_passed_mah_dis+=1;
  if ((cl_res+i)->mah_dis_the_other[j]>threshold2)
   (cl_res+i)->no_passed_mah_dis_the_other+=1;
  if ((cl_res+i)->mah_dis_the_other[j]>fmax)
   fmax=(cl_res+i)->mah_dis_the_other[j];
 }
 /*******/
 /******* Too Few Samples ?
 /*******/
 (cl_res+i)->threshold2=threshold2;
 minimum=5;
 if ((cl_res+i)->no_passed_mah_dis_the_other<minimum)
 {

  for (j=0; j<(class_info+class_index[i])->no_sample; j++)
   (cl_res+i)->sflag[j]=0;

  for (k=0; k<minimum; k++)
  {
   for (fmax= -1e30,j=0; j<(class_info+class_index[i])->no_sample; j++)
    if ((cl_res+i)->mah_dis_the_other[j]>fmax &&
        (cl_res+i)->classified_as[j]==i && (cl_res+i)->mah_dis[j]>threshold &&
        (cl_res+i)->sflag[j]==0) /* 333 */
    {
     fmax=(cl_res+i)->mah_dis_the_other[j];
             (cl_res+i)->sflag[j]=1;
    }
   }
  (cl_res+i)->threshold2=fmax;
 }
}

/*******/
/******* find the closest sample in the other group
/*******/
point_array_cnt=0;
for (i=0; i<2; i++)
 for (j=0; j<(class_info+class_index[i])->no_sample; j++)
  if ((cl_res+i)->classified_as[j]==i && (cl_res+i)->mah_dis[j]>threshold)
  {
    dmin=1e30;
    tp1=(class_info+class_index[i])->data_float+j*no_new_channel;
    for (flag=k=0; k<(class_info+class_index[1-i])->no_sample; k++)
     if (((cl_res+1-i)->classified_as[k]==(1-i)) &&
         (cl_res+1-i)->mah_dis[k]>threshold &&
         (cl_res+1-i)->mah_dis_the_other[k]>(cl_res+1-i)->threshold2) /* 333 */
     {
      tp2=(class_info+class_index[1-i])->data_float+k*no_new_channel;

              /*******/
              /******* Euclidean distance
              /*******/
      for (tmp=l=0; l<no_new_channel; l++)
       tmp+= (*(tp1+l) - *(tp2+l))*(*(tp1+l) - *(tp2+l));

      if (tmp<dmin)
      {
       dmin=tmp;
       imin=k;
      }
      flag=1;
     }

    if (flag==1)           /* save pair */
    {
     tp2=(class_info+class_index[1-i])->data_float+imin*no_new_channel;
     for (l=0; l<no_new_channel; l++)
     {
      *(point1_array+point_array_cnt*no_new_channel+l)= *(tp1+l);
      *(point2_array+point_array_cnt*no_new_channel+l)= *(tp2+l);

     }
     point_array_cnt++;
            }           /* if (flag==1) */
   }        /* for j */
```

```
            /*******/
            /******* find the point on decision boundary and calculate edbfm
            /*******/
            FS_sol_bnd_line((class_info+class_index[0])->mean,(class_info+class_index[1])->mean,
            (class_info+class_index[0])->cov,(class_info+class_index[1])->cov,
            point1_array,point2_array,point_array_cnt,no_new_channel,no_new_channel,edbfm,
            ERROR_FLAG);

            } /*################################################ if (*ERROR_FLAG==0) */

            /*******/
            /******* free memory
            /*******/
            SI_free_memory((char **)&icov_all);
            SI_free_memory((char **)&(cl_res+1)->sflag);
            SI_free_memory((char **)&(cl_res+0)->sflag);
            SI_free_memory((char **)&(cl_res+1)->mah_dis_the_other);
            SI_free_memory((char **)&(cl_res+0)->mah_dis_the_other);
            SI_free_memory((char **)&(cl_res+1)->mah_dis);
            SI_free_memory((char **)&(cl_res+0)->mah_dis);
            SI_free_memory((char **)&(cl_res+1)->classified_as);
            SI_free_memory((char **)&(cl_res+0)->classified_as);
            SI_free_memory((char **)&point2_array);
            SI_free_memory((char **)&point1_array);

            return;
}

/*******************************************************/
/*******
/*******    FUNCTION: FS_sol_bnd_line
/*******
/*******************************************************/
FS_sol_bnd_line(mean1,mean2,cov1,cov2,point1_array,point2_array,no_points,
dim,act_dim,edbfm,ERROR_FLAG)
double *mean1,*mean2,*cov1,*cov2;
long        dim,act_dim;
double      *point1_array,*point2_array,*edbfm;
long *ERROR_FLAG;
{
            double *icov1,*icov2,*icov_diff,*mean_icov_diff;
            double det1,det2,c,threshold=0,msm1,msm2;
            long i,j,k,rejected=0;
            double *normal,log_det1,log_det2;

            /*******/
            /******* initialize & check no of points
            /*******/
            for (i=0; i<dim*dim; i++)
              *(edbfm+i)=0;

            if (no_points<=0)
             return;

            /*******/
            /******* assign memory
            /*******/
            normal=(double *)SI_create_memory((unsigned long)dim*FS_DOUBLE);
            if (normal==FS_NULL)
             *ERROR_FLAG=528;

            icov1=(double *)SI_create_memory((unsigned long)dim*dim*FS_DOUBLE);
            if (icov1==FS_NULL)
             *ERROR_FLAG=529;

            icov2=(double *)SI_create_memory((unsigned long)dim*dim*FS_DOUBLE);
            if (icov2==FS_NULL)
             *ERROR_FLAG=530;
             icov_diff=(double *)SI_create_memory((unsigned long)dim*dim*FS_DOUBLE);
             if (icov_diff==FS_NULL)
              *ERROR_FLAG=530;

             mean_icov_diff=(double *)SI_create_memory((unsigned long)dim*FS_DOUBLE);
             if (mean_icov_diff==FS_NULL)
              *ERROR_FLAG=531;

             if (*ERROR_FLAG==0) /*############### continue if no error occurred */
             {
             /*******/
             /******* calculate icov_diff
             /*******/
             for (i=0; i<dim; i++)
              for (j=0; j<dim; j++)
              {
               *(icov1+i*dim+j)= *(cov1+i*act_dim+j);
               *(icov2+i*dim+j)= *(cov2+i*act_dim+j);
              }
             FS_inverse_log_det(icov1,dim,dim,&det1,&log_det1);
             FS_inverse_log_det(icov2,dim,dim,&det2,&log_det2);

             for (i=0; i<dim; i++)
              for (j=0; j<dim; j++)
               *(icov_diff+i*dim+j)= *(icov1+i*dim+j) - *(icov2+i*dim+j);


             /*******/
             /******* calculate mean_icov_diff
             /*******/
             for (i=0; i<dim; i++)
              for (*(mean_icov_diff+i)=j=0; j<dim; j++)
               *(mean_icov_diff+i) += *(mean1+j)* *(icov1+i*dim+j) - *(mean2+j)* *(icov2+i*dim+j);


             /*******/
             /******* calculate c
             /*******/
             for (msm1=msm2=i=0; i<dim; i++)
              for (j=0; j<dim; j++)
              {
               msm1+= *(mean1+i)* *(icov1+i*dim+j)* *(mean1+j);
               msm2+= *(mean2+i)* *(icov2+i*dim+j)* *(mean2+j);
              }
             c=0.5*((msm1-msm2)+log(fabs(det1/det2)));


             /*******/
             /******* calculate effective decision boudary feature matrix
             /*******/
             for (i=0; i<no_points; i++)
             {
              FS_sol_bnd_line_2(mean_icov_diff,icov_diff,c,point1_array+dim*i,point2_array+dim*i,
              dim,dim,normal,threshold,i,ERROR_FLAG);

              if (*ERROR_FLAG>0)
               break;

if (normal[0]==0 && normal[1]==0)
 rejected++;

              for (j=0; j<dim; j++)
               for (k=0; k<dim; k++)
                *(edbfm+j*dim+k) += *(normal+j)* *(normal+k);
             }

             /*******/
             /******* normalize edbfm
             /*******/
             for (j=0; j<dim; j++)
             for (k=0; k<dim; k++)
              *(edbfm+j*dim+k) /= no_points;
            } /*################################################## if (*ERROR_FLAG==0) */

            /*******/
            /******* free memory
            /*******/
            SI_free_memory((char **)&mean_icov_diff);
            SI_free_memory((char **)&icov_diff);
            SI_free_memory((char **)&icov2);
            SI_free_memory((char **)&icov1);
            SI_free_memory((char **)&normal);

            return;
}
/*********************************************************/
/*******
/*******     FUNCTION: sol_bnd_line_2
/*******
/*********************************************************/
FS_sol_bnd_line_2(mean_icov_diff,icov_diff,c,point1,point2,dim,act_dim,normal,threshold,
point_array_index,ERROR_FLAG)
double *icov_diff,*mean_icov_diff,c,threshold;
double      *point1,*point2,*normal;
long dim,act_dim,point_array_index,*ERROR_FLAG;
{
            double c2,a,b,u,u1,u2,root,*ficov,*V,v2[2];
            long i=act_dim, j=point_array_index;
            double t1,t2,t3;

            /*******/
            /******* assign memory for difference vector V and calcuate V
            /*******/
            if ((V=(double *)SI_create_memory((unsigned long)dim*FS_DOUBLE))==FS_NULL)
            {
             *ERROR_FLAG=532;
             return;
            }

            for (i=0; i<dim; i++)
             *(V+i)= *(point2+i) - *(point1+i);

            /*******/
            /******* calculate a, b, c2
            /*******/
            for (a=b=c2=i=0; i<dim; i++)
            {
             ficov = icov_diff+dim*i;

             for (t1=t3=j=0; j<i; j++)
             {
              t1 += *(V+j)* *(ficov+ j);
              t3 += *(point1+j)* *(ficov+ j);
             }

             a += *(V+i)*(t1+t1+ *(V+i)* *(ficov+ i));
             c2 += *(point1+i)*(t3+t3+ *(point1+i)* *(ficov+ i));

             for (t2=j=0; j<dim; j++)
              t2 += *(ficov+ j)* *(V+j);
             b += *(point1+i)*t2;

            }           /* for i */
            a/=2;
            c2/=2;

            for (i=0; i<dim; i++)
            {
             b -= *(mean_icov_diff+i)* *(V+i);
             c2 -= *(mean_icov_diff+i)* *(point1+i);
```

```
}
c2+=c;


/*******/
/******* calculate u
/*******/
if (fabs(a)<1e-6*fabs(b))
{
 if (b==0)
  outerr("ERROR no solution... E5626");

 u=threshold-c2/b;
}
else
{
 if ((b*b-4*a*(c2-threshold))<0)
 {

  printf("ERROR    imagenary root...IGNORE\n");

  for (i=0; i<dim; i++)
   *(normal+i)= 0;

  SI_free_memory((char **)&V);
  return;
 }

 root=sqrt(b*b-4*a*(c2-threshold));
 u1=(-b+root)/2/a;
 u2=(-b-root)/2/a;
 if ((u1>=0) && (u1<=1))
  u=u1;
 else if ((u2>=0) && (u2<=1))
  u=u2;
 else
 {

  printf("ERROR    no solution between two points...IGNORE\n");

  for (i=0; i<dim; i++)
   *(normal+i)= 0;

  SI_free_memory((char **)&V);
  return;
 }
}

/*******/
/******* find intersection point(normal[i]) and calculate h(X)
/*******/
for (i=0; i<dim; i++)
  *(normal+i)= u* *(V+i) + *(point1+i);
cal_h_X(&t2,normal,icov_diff,mean_icov_diff,dim,c);

if (fabs(t2)>1e-7)
 {

  for (i=0; i<dim; i++)
   *(normal+i)= 0;

  SI_free_memory((char **)&V);
  return;
 }

/*******/
/******* find normal vector & normalize
/*******/
for (t1=i=0; i<dim; i++)
{
```

```
             V[i]=0;
             for (j=0; j<dim; j++)
              V[i]+= *(icov_diff+dim*i+j)*normal[j];
             V[i]-= mean_icov_diff[i];
             t1+= V[i]*V[i];
            }

            t1=sqrt(t1);
            for (i=0; i<dim; i++)
              normal[i]=V[i]/t1;

            /*******/
            /******* free memory
            /*******/
            SI_free_memory((char **)&V);
            return;
}


/************/
/***
/*** FUNCTION: FS_sub_linear_transform
/***
/************/
FS_sub_linear_transform(fdata_after,fdata_ori,matrix,matarix_dim,
no_new_channel,no_sample)
float *fdata_ori,*fdata_after;
double *matrix;
long no_new_channel,no_sample,matarix_dim;
{
            long l,j,k;

            for (j=0; j<no_sample; j++)
             for (k=0; k<no_new_channel; k++)
              for (fdata_after[j*no_new_channel+k]=l=0; l<no_new_channel; l++)
                        fdata_after[j*no_new_channel+k] += *(fdata_ori+j*no_new_channel+l) *
                       *(matrix+l*matarix_dim+k);
            return;
}
/*******************************************************/
/*******     FUNCTION: cal_h_X
/*******************************************************/
cal_h_X(h_X,normal,icov_diff,mean_icov_diff,dim,c)
double *h_X,*normal,*icov_diff,*mean_icov_diff,c;
int dim;
{
            int i,j;
            double t1,t3,*ficov;

            for (t3=c,*h_X=i=0; i<dim; i++)
            {
             ficov = icov_diff+dim*i;

             for (t1=j=0; j<i; j++)
              t1 += normal[j]* *(ficov+ j);

             *h_X += normal[i]*(t1+t1+ normal[i]* *(ficov+ i));
             t3 -= *(mean_icov_diff+i)* normal[i];
            }
            *h_X= *h_X/2+t3;
}
```

Appendix E
Program of Decision Boundary Feature Extraction for Parzen Density Estimator

```
/*****************************************************************/
/***
/***
/***        FUNCTION: fea_sel_parzen_by_DBFM (31)
/***
/***        class: struct (See Appendix C)
/***
/*****************************************************************/
fea_sel_parzen_by_DBFM(class,no_class,no_new_channel)
struct class_str        *class;
int no_class,no_new_channel;
{
            double *edbfm_all,*edbfm,*E_value_edbfm,*EV_edbfm;
            int i,j,k,l,*class_index,rank1,rank2,no_feature,data_type,save_mode;
            double cum,sum;
            float tacc,*over_accuracy,*weighted_accuracy;
            long longint;
            char tmp[200];
            char DB_kernel[10];

            float *ndata;
            int *nindex,total_sam,flag_tr_te,correct_cl_flag,no_sam_selected;

            /*******/
            /******* read parameters
            /*******/
            read_flag_file("flag.correct",&G_use_correct_only);
            read_flag_file("flag.np.random",&G_np_ramdon);
            G_parzen_h_size=G_def_h_size;
            printf("G_use_correct_only=%d G_np_ramdon=%d\n",G_use_correct_only,G_np_ramdon);

            read_flag_file_double("flag.portion",&cum);
            if (cum>0)
             G_DBPZ_portion=cum;
            read_flag_file_double("flag.incre",&cum);
            if (cum>0)
             G_DBPZ_incre=cum;

             printf (" G_DBPZ_portion=%.2f G_parzen_h_size=%.1f G_DBPZ_incre=%.2f\n",
             G_DBPZ_portion,G_parzen_h_size,G_DBPZ_incre);

            /*******/
            /******* make file index and inverse cov
            /*******/
            if (G_fs_parzen_kernel==1)
             strcpy(DB_kernel,"DB2");
            else
             strcpy(DB_kernel,"DB");

            cal_icov_class(class,no_class,no_new_channel);

            printf("\nfea_sel_parzen_by_DBFM...%d\n",input_parameter.result_file_index);

            /*******/
            /******* assign memory
            /*******/
            longint=no_new_channel*no_new_channel*DOUBLE;
            edbfm_all=(double *)malloc(longint);
            edbfm=(double *)malloc(longint);
            EV_edbfm=(double *)malloc(longint);
```

```
/***
/*** Assign memory and classify by parzen
/***/
class_index=(int *)malloc((long)INT*no_class);
for (i=0; i<no_class; i++)
 class_index[i]=i;

assign_class_parzen_result(class,no_class);


/*******/
/******* Classify training data
/*******/
data_type=1; /* training data 542 */
printf("## Classify training data...\n");
sub_parzen_classifier(class,no_class,class_index,no_new_channel,no_new_channel,
&tacc,data_type);

/* make confusion matrix */
make_classification_result_table(class,no_class,no_new_channel,data_type);

/*******/
/******* find edbfm
/*******/
E_value_edbfm=edbfm_all;
i= -1;

if (i<0)
{
/*******/
/******* calculate edbfm
/*******/
for (i=0; i<no_new_channel*no_new_channel; i++)
 edbfm_all[i]=0;

for (i=0; i<no_class; i++)
 for (j=0; j<i; j++)
 {
  class_index[0]=i;
  class_index[1]=j;

  printf("i=%d j=%d class_index [%d-%d]\n",i,j,class_index[0],class_index[1]);
  sub_find_edbfm_2_class_by_parzen(class,no_class,class_index,no_new_channel,edbfm);

  for (k=0; k<no_new_channel*no_new_channel; k++)
   edbfm_all[k]+=edbfm[k];
 }

E_value_edbfm=edbfm_all;
deigen(no_new_channel,E_value_edbfm,2,&i,EV_edbfm,no_new_channel);

normalize_column(EV_edbfm,no_new_channel,no_new_channel);
} /* not read_edbfm */

/*******/
/******* extimate rank
/*******/
estimate_rank(&rank1,&rank2,no_new_channel,E_value_edbfm);
for (sum=i=0; i<no_new_channel; i++)
 sum+=E_value_edbfm[i*no_new_channel+i];
printf("Total: Rank1(>90 percent & <2 percent)=%d Rank2(>95 percent)=%d sum=%.1f\n",
rank1,rank2,sum);

/*******/
/******* calculate new features and stat
/*******/
cal_new_fea_class_float2(class,no_class,no_new_channel,EV_edbfm,no_new_channel);
calculate_new_mean_cov(no_class,no_new_channel,class);

cal_icov_class(class,no_class,no_new_channel);
```

```
           /*******/
           /******* save eigenvalues
           /*******/
           for (i=0; i<no_new_channel; i++)
            G_E_value_edbfm[i]=E_value_edbfm[i+no_new_channel*i];
           G_no_new_channel=no_new_channel;

           /*******/
           /******* classify using features selected by DBFM
           /*******/
           over_accuracy=(float *)malloc((long)FLOAT*no_new_channel);
           weighted_accuracy=(float *)malloc((long)FLOAT*no_new_channel);

           for (i=0; i<no_class; i++)
            class_index[i]=i;
           for (no_feature=1; no_feature<=no_new_channel; no_feature++)
           {
            parzen_classifier(class,no_class,class_index,no_new_channel,no_feature,&tacc);
            make_classification_result_table(class,no_class,no_feature,0);

            over_accuracy[no_feature-1]=weighted_accuracy[no_feature-1]=tacc;
            save_result(over_accuracy,weighted_accuracy,no_feature);
           }

           /*******/
           /******* print result
           /*******/
           printf ("fea_sel_parzen_by_DBFM G_DBPZ_portion=%.2f G_parzen_h_size=%.1f G_DBPZ_incre=%.2f\n",
           G_DBPZ_portion,G_parzen_h_size,G_DBPZ_incre);
           for (i=0; i<no_new_channel; i++)
            if (i==i/5*5)
             printf (" %.1f\n",weighted_accuracy[i]);
            else
             printf ("%.1f\n",weighted_accuracy[i]);

           free(edbfm_all);
}

/************/
/***
/*** FUNCTION: parzen_classifier
/***
/************/
parzen_classifier(class,no_class,class_index,no_new_channel,no_feature,tacc)
struct class_str          *class;
int no_new_channel,no_class,no_feature,*class_index;
float *tacc;
{
           int data_type=0; /* test data */
           sub_parzen_classifier(class,no_class,class_index,no_new_channel,no_feature,
           tacc,data_type);

}
/************/
/*** FUNCTION: sub_parzen_classifier
/************/
sub_parzen_classifier(class,no_class,class_index,no_new_channel,no_feature,tacc,data_type)
struct class_str          *class;
int no_new_channel,no_class,no_feature,*class_index,data_type;
float *tacc;
{
           register float *reg_fl1,*reg_fl2;
           register double *rcov,*rcov1;
           int i,j,k,ktmp,ind_class,cnt_class,ind_sample,a,b;
           int nos_tr,nos_test,no_err,tres,tnos,terr,bit_map_index;

           float x,acc,acc1;
           double dis,prob,tmax,t1,t2,t3,t5;
```

```
            printf("Parzen classifier h=%.1f G_DBPZ_incre=%.1f\n",
            G_parzen_window_size,G_DBPZ_incre);


            /***/
            /***  loop for each class */
            /***/
            for (tnos=terr=cnt_class=0; cnt_class<no_class; cnt_class++)
            {
             ind_class=class_index[cnt_class];
             find_bit_map_index((class+ind_class)->id_number,&bit_map_index);

            printf("cnt_class=%d ind_class=%d id_number=%d bit_map_index=%d\n",
            cnt_class,ind_class,(class+ind_class)->id_number,bit_map_index);

             /***/
             /***  loop for each sample in each class */
             /***/
             for (no_err=nos_test=ind_sample=0; ind_sample<(class+ind_class)->no_sample; ind_sample++)
             if (G_bitpmap_flag==0 ||
                 (int)*((G_ms_bit_map+bit_map_index)->bit_map+ind_sample)==data_type)
             {
              nos_test++;
              tmax= -1e30;
              reg_fl1=(class+ind_class)->data_float2+ind_sample*no_new_channel;

              /***/
              /***  calculate classwise probability */
              /***/
              for (i=0; i<no_class; i++)
              {
               for (prob=nos_tr=j=0; j<(class+i)->no_sample; j++)
               if (G_bitpmap_flag==0 || (int)*((G_ms_bit_map+i)->bit_map+j)==1)
               if (i!=ind_class || ind_sample!=j)            /* exclude self training sample */
               {
                nos_tr++;
                reg_fl2=(class+i)->data_float2+j*no_new_channel;

              /***/
              /***  Gaussian kernel:identity matrix */
              /***/
if (G_fs_parzen_kernel==0)
{
                for (dis=k=0; k<no_feature; k++)
                {
                 x=reg_fl2[k]-reg_fl1[k];
                 dis+=x*x;
                }
                prob+=exp(-dis/(2*G_parzen_h_size*G_parzen_h_size));
}
              /***/
              /***  Gaussian kernel:individual covariance */
              /***/
if (G_fs_parzen_kernel==1)
{
                       for (dis=a=0; a<no_feature; a++)
                       {
                                   rcov1 = (class+i)->icov+no_new_channel*a;
                   t3= *(reg_fl2+a)- *(reg_fl1+ a);

                   for (t1=b=0; b<a; b++)
                    t1 += (*(reg_fl2+ b)- *(reg_fl1+ b))* *(rcov1+ b);
                   dis += t3*(t1+t1+t3* *(rcov1+ a));
                }
               prob+=   (class+i)->square_det*exp(-dis/2./
                       G_parzen_window_size/G_parzen_window_size+L_parzen_scale);
}

              /***/
              /***  Uniform kernel */
              /***/
if (G_fs_parzen_kernel==2)
{
                for (dis=k=0; k<no_feature; k++)
                {
                 x=reg_f12[k]-reg_f11[k];
                 dis+=x*x;
                }
L_dis_cnt++;
L_dis_ave=((L_dis_cnt-1.)*L_dis_ave+dis)/L_dis_cnt;
                if (dis<G_parzen_h_size)
                 prob+=1;
}
               }        /* if G_bitpmap_flag==0 || */

               prob/=nos_tr;

                (class+ind_class)->parzen_result[ind_sample*no_class+i]=prob;           /* save prob */

                if (prob>tmax)
                {
                 tmax=prob;
                 tres=i;
                }
              }          /* for  i */

              /* save classification result for each sample */
              ktmp=ind_sample;
               (class+ind_class)->classification_np_result[ktmp]=(unsigned char)tres;

              if (tres!=ind_class)
               no_err++;
             }          /*  for ind_sample */

             tnos+=nos_test;
             terr+=no_err;

             acc=100.*(nos_test-no_err)/nos_test;
             printf("N=%d CLASS:%d nos=%3d err=%3d Acc=%.1f(%3.1f) G_parzen_h_size=%.1f\n",
             no_feature,ind_class,nos_test,no_err,acc,100.-acc,G_parzen_h_size);
            }

             *tacc=100.*(tnos-terr)/tnos;
             printf("\nAVERAGE OF ALL CLASS N=%d nos=%d err=%d Acc=%.1f(%.1f) G_parzen_h_size=%.1f\n\n",
             no_feature,tnos,terr,*tacc,100.- *tacc,G_parzen_h_size);
}

/*******************************************************/
/*******
/*******     FUNCTION: sub_find_edbfm_2_class_by_parzen
/*******
/*******************************************************/
sub_find_edbfm_2_class_by_parzen(class,no_total_class,class_index,no_new_channel,edbfm)
struct class_str          *class;
int *class_index,no_new_channel,no_total_class;
double *edbfm;
{
             sub_find_edbfm_2_class_by_parzen_2(class,no_total_class,class_index,no_new_channel,
             edbfm);
}

/*******************************************************/
/*******     FUNCTION: sub_find_edbfm_2_class_by_parzen_2
/*******************************************************/
sub_find_edbfm_2_class_by_parzen_2(class,no_total_class,class_index,no_new_channel,edbfm)
struct class_str          *class;
int *class_index,no_new_channel,no_total_class;
double *edbfm;
{
```

```
int i,j,k,l,a,b,key_bit,cnt_tbl[5],point_array_cnt,bit_map_index[2],flag,nos,tnos=0;
double      dmin,class_min[2];
register float *rfp1,*rfp2;
int *result,imin,ierr[2],inos[2],class2=2;
float *vectors[2],*float_points,*min_fp,*td,tmin,ftmp;
long longint;

double t1,t2,t3,*hX_float_data,*sd,ave_sd;
int rank1,rank2,total_error=0,total_sample,data_type=1        /* train data */;

/*******/
/******* assign memory & init
/*******/
vectors[0]=(float *)malloc((unsigned long)no_new_channel*FLOAT);
vectors[1]=(float *)malloc((unsigned long)no_new_channel*FLOAT);
sd=(double *)malloc((unsigned long)DOUBLE*no_new_channel);

longint=(class+class_index[0])->no_sample+(class+class_index[1])->no_sample;
if ((hX_float_data=(double *)malloc((unsigned long)DOUBLE*longint))==NULL)
  outerr("ERR dkeio3");

float_points=(float *)malloc((unsigned long)FLOAT*no_new_channel*longint);
point_array_cnt=0;

for (j=0; j<no_new_channel*no_new_channel; j++)
  edbfm[j] =0;

for (i=0; i<class2; i++)
  find_bit_map_index((class+class_index[i])->id_number,bit_map_index+i);

/***/
/*** cal average sd
/***/
for (ave_sd=i=0; i<no_new_channel; i++)
{
 for (sd[i]=j=0; j<2; j++)
  sd[i]+=sqrt((class+class_index[j])->cov[i*no_new_channel+i]);
 ave_sd+=sd[i]/no_new_channel;
}

/*******/
/******* find pairs
/*******/
for (i=0; i<class2; i++)
{
 for (nos=j=0; j<(class+class_index[i])->no_sample; j++)
 {
  if ((G_bitpmap_flag==0 ||
        (int)(G_ms_bit_map+bit_map_index[i])->bit_map[j]==data_type) /* train */
                         &&
               (*((class+class_index[i])->parzen_result+j*no_total_class+class_index[i])>
                *((class+class_index[i])->parzen_result+j*no_total_class+class_index[1-i])))
 {
             /*******/
             /******* find closest sample
             /*******/
   tmin=1e30;
   if (G_np_ramdon==1)
   {
    for (; ;)
    {
     k=random();
     k=k-k/(class+class_index[1-i])->no_sample*(class+class_index[1-i])->no_sample;
    if ((G_bitpmap_flag==0 ||
          (int)*((G_ms_bit_map+bit_map_index[1-i])->bit_map+k)==data_type) /* test data */
             &&
             (*((class+class_index[1-i])->parzen_result+k*no_total_class+class_index[i])
             < *((class+class_index[1-i])->parzen_result+k*no_total_class+class_index[1-i])))
      break;
    }
```

```
               imin=k;
               min_fp=(class+class_index[1-i])->data_float2+k*no_new_channel;
               rfp1=(class+class_index[i])->data_float2+j*no_new_channel;
              }        /* if (G_np_ramdon==1) */
              else
              {
              for (k=0; k<(class+class_index[1-i])->no_sample; k++)
               if ((G_bitpmap_flag==0 ||
                     (int)*((G_ms_bit_map+bit_map_index[1-i])->bit_map+k)==data_type) /* test data */
                       &&
                       (*((class+class_index[1-i])->parzen_result+k*no_total_class+class_index[i])
                       < *((class+class_index[1-i])->parzen_result+k*no_total_class+class_index[1-i])))
               {
                rfp1=(class+class_index[i])->data_float2+j*no_new_channel;
                rfp2=(class+class_index[1-i])->data_float2+k*no_new_channel;
                for (ftmp=1=0; 1<no_new_channel; 1++)
                 ftmp+= (rfp1[1]-rfp2[1])*(rfp1[1]-rfp2[1]);

                if (ftmp<tmin)
                {
                 tmin=ftmp;
                 imin=k;
                 min_fp=rfp2;
                }
               }
               } /* else */

              if (i==0)
                       find_point_on_DB(class,class_index,float_points+no_new_channel*point_array_cnt,
                       no_new_channel,rfp1,min_fp,hX_float_data+point_array_cnt,ave_sd);
              else
                       find_point_on_DB(class,class_index,float_points+no_new_channel*point_array_cnt,
                       no_new_channel,min_fp,rfp1,hX_float_data+point_array_cnt,ave_sd);

              point_array_cnt++;
             }         /* if after for j */
            } /* for j */
           }           /* for i */

           cal_grad(class,class_index,float_points,point_array_cnt,no_new_channel,sd,
           hX_float_data,edbfm);
}

/************/
/***
/*** FUNCTION: cal_grad
/***
/************/
int cnt_cal_grad_1=0;
cal_grad(class,class_index,float_points,no_points,no_new_channel,sd,hX_float_data,edbfm)
struct class_str         *class;
int no_new_channel,*class_index;
float *float_points;
double *sd,*hX_float_data,*edbfm;
{
           float *float_incre;
           int i,j,k,result_class_index,over_flow_error,ignore_flag,ignored=0;
           double ratio,sum,*normal,bak;

           bak=G_parzen_h_size;

           if (input_parameter.result_file_index==44)
           {
                       read_flag_file_double("flag.grad_h_size",&G_grad_h_size);

                       if (G_grad_h_size>0)
                        G_parzen_h_size=G_grad_h_size;
           }
```

```
            ERR_cnt_zero_prob_bak= ERR_cnt_zero_prob;

            float_incre=(float *)malloc((unsigned long)FLOAT*(no_new_channel+1));
            normal=(double *)malloc((unsigned long)DOUBLE*(no_new_channel+1));

            for (i=0; i<no_points; i++)
            {

              for (k=0; k<no_new_channel; k++)
               float_incre[k]=float_points[no_new_channel*i+k];
              parzen_classifier_one_sample(class,class_index,no_new_channel,no_new_channel,
              float_incre,&result_class_index,&ratio,&over_flow_error);
              if (fabs((hX_float_data[i]-ratio)/ratio)>0.001)
              {
               hX_float_data[i]=ratio;
              }


             for (ignore_flag=sum=j=0; j<no_new_channel; j++)
             {
              for (k=0; k<no_new_channel; k++)
               float_incre[k]=float_points[no_new_channel*i+k];

              float_incre[j]+=sd[j]*G_DBPZ_incre;

              ERR_cnt_zero_prob_com[0]= (char)71;
              parzen_classifier_one_sample(class,class_index,no_new_channel,no_new_channel,
              float_incre,&result_class_index,&ratio,&over_flow_error);

              if (1>0 ||over_flow_error==0)       /* exclude zero probability */
              {
               normal[j]=(log(ratio)-log(hX_float_data[i]))/(sd[j]*G_DBPZ_incre);
               sum+=normal[j]*normal[j];
              }
              else
              {
               if (ignored==0)
                ignored++;
               ignore_flag=1;
               printf("cal_grad IGNORED...i=%d j=%d sum=%f no_points=%d\n",i,j,sum,no_points);
              }
             } /* for j*/

             sum=sqrt(sum);
             if (sum>0 && ignore_flag==0)
             {
              for (j=0; j<no_new_channel; j++)
               normal[j]/=sum;

              for (j=0; j<no_new_channel; j++)
               for (k=0; k<no_new_channel; k++)
                edbfm[j*no_new_channel+k]+=normal[j]*normal[k];
             }
            }

            G_parzen_h_size=bak;
            free(normal);
            free(float_incre);
}

/************/
/***
/*** FUNCTION: find_point_on_DB
/***
/************/
int                         local_repeat=0;
int                         local_counter=0;

find_point_on_DB(class,class_index,float_point_on_DB,no_new_channel,
float_point1,float_point2,hX_float_data,ave_sd)
```

```
struct class_str        *class;
int no_new_channel,*class_index;
float *float_point_on_DB,*float_point1,*float_point2;
double *hX_float_data,ave_sd;
{
            int i,result_class_index,over_flow_error;
            float err,threshold=G_DBPZ_portion*ave_sd,*new_point;
            double ratio;


            threshold=G_DBPZ_portion*ave_sd;
            for (err=i=0; i<no_new_channel; i++)
            {
             err+=(float_point1[i]-float_point2[i])*(float_point1[i]-float_point2[i]);
             float_point_on_DB[i]=(float_point1[i]+float_point2[i])/2;
            }


            if (err<threshold && local_repeat>0)
            {
             local_repeat=0;
             return;
            }

            new_point=(float *)malloc((unsigned long)FLOAT*no_new_channel);         /* assign memory */

            ERR_cnt_zero_prob_com[0]= (char)"70";
            parzen_classifier_one_sample(class,class_index,no_new_channel,no_new_channel,
            float_point_on_DB,&result_class_index,&ratio,&over_flow_error);

            *hX_float_data=ratio;

            local_repeat++;

            for (i=0; i<no_new_channel; i++)
             new_point[i]=float_point_on_DB[i];

            if (result_class_index==class_index[0])
             find_point_on_DB(class,class_index,float_point_on_DB,no_new_channel,
             new_point,float_point2,hX_float_data,ave_sd);
            else
             find_point_on_DB(class,class_index,float_point_on_DB,no_new_channel,
             float_point1,new_point,hX_float_data,ave_sd);

            free(new_point);

}

/***/
/***
/***        FUNCTION: cal_icov_class
/***/
cal_icov_class(class,no_class,no_new_channel)
struct class_str        *class;
int no_new_channel,no_class;
{
            int i,j;
            double dmax= -1e30;

            if (G_fs_parzen_kernel!=1)
             return;

            for (i=0; i<no_class; i++)
            {
             for (j=0; j<no_new_channel*no_new_channel; j++)
              (class+i)->icov[j]=(class+i)->cov[j];

             dinverse((class+i)->icov,no_new_channel,no_new_channel,&(class+i)->det);
             if ((class+i)->det==0)
              printf("ERROR class:%d DET=%g\n",i,(class+i)->det);
            (class+i)->square_det=1./sqrt((class+i)->det);
            printf("%d DET=%g 1/square_det=%g\n",i,(class+i)->det,(class+i)->square_det);
            if ((class+i)->square_det>dmax)
             dmax=(class+i)->square_det;
           }
           for (i=0; i<no_class; i++)
            (class+i)->square_det/=dmax;
}

/******/
/*******
/******* FUNCTION: assign_class_parzen_result
/*******
/******/
assign_class_parzen_result(class,no_class)
struct class_str        *class;
int no_class;
{
           long longint,i,j;

           for (i=0; i<no_class; i++)
            {
             longint=(long)FLOAT*(class+i)->no_sample*no_class;
             (class+i)->parzen_result=(float *)malloc((unsigned)longint);

             for (j=0; j<(class+i)->no_sample*no_class; j++)
              (class+i)->parzen_result[j]= -1234567;

            }          /* for i */

}
/*******************************************************/
/*******     FUNCTION: estimate_rank
/*******************************************************/
estimate_rank(rank1,rank2,no_new_channel,evalue)
int *rank1,*rank2,no_new_channel;
double *evalue;
{
           int k;
           double sum,cum;

             for (sum=k=0; k<no_new_channel; k++)
              sum+=evalue[k+no_new_channel*k];
             for (*rank1= *rank2=cum=k=0; k<no_new_channel; k++)
             {
              cum+=evalue[k+no_new_channel*k];
              if (*rank1==0 && cum>0.90*sum && evalue[k+no_new_channel*k] < 0.02*sum)
               *rank1=k+1;
              if (*rank2==0 && cum>0.95*sum)
               *rank2=k+1;
             }
}
/*******************************************************/
/*******
/*******     FUNCTION: cal_new_fea_class_float2
/*******
/*******************************************************/
cal_new_fea_class_float2(class,no_class,no_new_channel,EV,dim_EV)
struct class_str        *class;
int no_new_channel,dim_EV,no_class;
double *EV;
{
           int i,j,k,l;

           for (i=0; i<no_class; i++)
             for (j=0; j<(class+i)->no_sample; j++)
              for (k=0; k<no_new_channel; k++)
               *((class+i)->data_float1+j*no_new_channel+k)=
               *((class+i)->data_float2+j*no_new_channel+k);
            for (i=0; i<no_class; i++)
            {
              printf("Calculating new features %d...\n",i);
              for (j=0; j<(class+i)->no_sample; j++)
               for (k=0; k<no_new_channel; k++)
               {
                *((class+i)->data_float2+j*no_new_channel+k)=0;

                for (l=0; l<dim_EV; l++)
                          *((class+i)->data_float2+j*no_new_channel+k) +=
                        *((class+i)->data_float1+j*no_new_channel+l) *
                        *(EV+l*dim_EV+k);
               }
             }
}
```

Appendix F
Program of Decision Boundary Feature Extraction for Neural Networks

```
typedef struct weight_array_str {
                double          *weight;
                double          *out;          /* out[M], M:no neurons of self stage */
                double          *delta;
                double          *error;
                double          *bias_weight;

                int             input_dim;
                int             out_dim;
} WEIGHT_ARRAY_STR;             /* no_stage needed */

typedef struct in_data_str {
                double          *data;
                double          target_0;
                double          target_1;
                double          learing_rate;

                int             class;

                short           *kNN;
                short           classified_as_parzen;

                short           classified_as;
} IN_DATA_STR;


int             L_no_stage,*L_no_neuron_stage;
struct weight_array_str *L_weight_array;

main(argc,argv)
int argc;
char *argv[];
{
                char cn[100],nn[100];
                char training_file_name[100],test_file_name[100];
                struct weight_array_str *weight_array;
                struct in_data_str *tr_data;
                struct in_data_str *te_data;
                struct class_str *class;

                long longint;
                int no_stage,*no_neuron_stage,tr_data_dim,out_dim,no_tr_data,no_err;
                int             te_data_dim,no_te_data;

                int i,j,k,no_class,no_new_channel,no_err_by_max;

                double *edbfm_all,*edbfm,*E_value_edbfm,*EV_edbfm;
                int *class_index;
                char fname[200];

                if( (ofp = fopen("nndb.DB","w")) == NULL )
                 outerr("ERROR 3572gh : no stat file");
                fclose(ofp);
                if( (ofp = fopen("nndb.DB2","w")) == NULL )
                 outerr("ERROR 3572gh : no stat file");
                fclose(ofp);

                /*******/
                /******* global variable */
                /*******/
                read_nn_flag_file("nflag.hidden_neuron",&G_no_hidden_neuron);
                if (G_no_hidden_neuron<1)
                 G_no_hidden_neuron=3;

                read_nn_flag_file("nflag.no_feature",&G_no_feature);

                printf("G_no_hidden_neuron=%d G_no_feature=%d\n",G_no_hidden_neuron,G_no_feature);
```

```
/*******/
/******* read input-parameter */
/*******/
if (INT> 2 && argc>1)
 strcpy(training_file_name,argv[1]);
else
{
 printf("nueral train file name =>\n");
 scanf("%s",cn);
 strcpy(training_file_name,cn);
}

if (INT> 2 && argc>2)
 strcpy(test_file_name,argv[2]);
else
{
 printf("nueral test file name =>\n");
 scanf("%s",cn);
 strcpy(test_file_name,cn);
}

if (INT> 2 && argc>3)
 strcpy(G_weight_bias_fname,argv[3]);
else
{
 printf("nueral test file name =>\n");
 scanf("%s",cn);
 strcpy(G_weight_bias_fname,cn);
}

printf("training_file_name=%s\n",training_file_name);
printf("test_file_name=%s\n",test_file_name);
printf("G_weight_bias_fname=%s\n",G_weight_bias_fname);

/*******/
/******* assign memory for weight_array */
/*******/
no_stage=3;  /*  counting input stage */
longint=sizeof(WEIGHT_ARRAY_STR)*(no_stage+1);
weight_array=(WEIGHT_ARRAY_STR *)malloc((unsigned)longint);
no_neuron_stage=(int *)malloc((unsigned)INT*(no_stage+1));

L_no_neuron_stage=no_neuron_stage;
L_no_stage=no_stage;
L_weight_array=weight_array;

/*******/
/******* assign memory for in_data and read data into in_data (train) */
/*******/
read_info_nn_file(training_file_name,&no_tr_data,&out_dim,&tr_data_dim);
G_no_training=no_tr_data;

longint=sizeof(IN_DATA_STR)*no_tr_data;
tr_data=(IN_DATA_STR *)malloc((unsigned)longint);
longint=DOUBLE*tr_data_dim;
for (i=0; i<no_tr_data; i++)
  (tr_data+i)->data=(double *)malloc((unsigned)longint);

read_nn_file(training_file_name,&no_tr_data,&out_dim,&tr_data_dim,tr_data);
printf("File Name=%s no_tr_data=%d tr_data_dim=%d no_class=%d\n",
training_file_name,no_tr_data,tr_data_dim,out_dim);

/*******/
/******* assign memory for in_data and read data into in_data (test) */
/*******/
read_info_nn_file(test_file_name,&no_te_data,&out_dim,&te_data_dim);

longint=sizeof(IN_DATA_STR)*no_te_data;
te_data=(IN_DATA_STR *)malloc((unsigned)longint);
```

```
longint=DOUBLE*te_data_dim;
for (i=0; i<no_te_data; i++)
 (te_data+i)->data=(double *)malloc((unsigned)longint);

read_nn_file(test_file_name,&no_te_data,&out_dim,&te_data_dim,te_data);
printf("File Name=%s no_te_data=%d te_data_dim=%d no_class=%d\n",
training_file_name,no_te_data,te_data_dim,out_dim);

if (te_data_dim!=tr_data_dim)
 outerr("te_data_dim!=tr_data_dim");

G_original_in_dim=te_data_dim;
if (G_no_feature>0)
 te_data_dim=tr_data_dim=G_no_feature;
else
 G_no_feature=te_data_dim;


/*******/
/******* Assign memory
/*******/
no_new_channel=tr_data_dim;

longint=no_new_channel*no_new_channel*DOUBLE;
edbfm_all=(double *)malloc(longint);
edbfm=(double *)malloc(longint);
EV_edbfm=(double *)malloc(longint);

/*******/
/******* assign no_stage  */
/*******/
for (i=0; i<no_stage; i++)
 no_neuron_stage[i]=tr_data_dim*G_no_hidden_neuron;

no_neuron_stage[0]=tr_data_dim;
no_neuron_stage[1]=tr_data_dim*G_no_hidden_neuron;
no_neuron_stage[no_stage]=no_neuron_stage[no_stage-1]=out_dim;


/*******/
/******* initialize network and read weight and bias */
/*******/
make_neural_network_init(weight_array,no_stage,no_neuron_stage);
printf("Reading weight_bias=%s\n",G_weight_bias_fname);
read_weight_bias(G_weight_bias_fname,weight_array,no_stage,&iter_start);
printf("iter_start=%d\n",iter_start);


/*******/
/******* classify training data
/*******/
printf("Classifying...\n");
classify_neural_network(weight_array,tr_data,tr_data_dim,no_tr_data,no_stage,
no_neuron_stage,&no_err,&no_err_by_max);
printf("Train classification result err=%.1f(%d/%d)\n",100.*no_err/no_tr_data,
no_err,no_tr_data);

/*******/
/******* cal_L_ave_sd_data
/*******/
cal_L_ave_sd_data(tr_data,tr_data_dim,no_tr_data,&L_ave_sd,L_sd);

/*******/
/******* assign memory
/*******/
no_class=out_dim;
class_index=(int *)malloc((long)INT*no_class);
for (i=0; i<no_class; i++)
 class_index[i]=0;
```

```
            class=(CLASS_STR *)malloc((unsigned)sizeof(CLASS_STR)*no_class);
            for (i=0; i<no_class; i++)
            {
             (class+i)->data_float2=(float *)malloc(no_new_channel*FLOAT*no_tr_data);
             (class+i)->no_sample=0;
            }

            /*******/
            /******* assign only correctly classified samples
            /*******/
            for (i=0; i<no_tr_data; i++)
            {
             k=(tr_data+i)->classified_as;
             if (k==(tr_data+i)->class)
             {
              for (j=0; j<no_new_channel; j++)
                (class+k)->data_float2[(class+k)->no_sample*no_new_channel+j]=(tr_data+i)->data[j];
              (class+k)->no_sample +=1;
             }
            }
            for (i=0; i<no_class; i++)
             printf("no of samples (class %d) =%d\n",i,(class+i)->no_sample);


            /*******/
            /******* find edbfm
            /*******/
            for (i=0; i<no_new_channel*no_new_channel; i++)
             edbfm_all[i]=0;

            for (i=0; i<no_class; i++)
             for (j=0; j<i; j++)
             {
              class_index[0]=i;
              class_index[1]=j;
printf("i=%d j=%d class_index [%d-%d]\n",i,j,class_index[0],class_index[1]);

              sub_find_edbfm_2_class_by_nn(class,class_index,no_new_channel,edbfm);

              for (k=0; k<no_new_channel*no_new_channel; k++)
               edbfm_all[k]+=edbfm[k];
             }

            E_value_edbfm=edbfm_all;
            my_deigen(no_new_channel,E_value_edbfm,2,&i,EV_edbfm,no_new_channel);

            sprintf(fname,"EV%s\0\n",training_file_name);
            save_edbfm_NN(fname,EV_edbfm,E_value_edbfm,no_new_channel);

            /*******/
            /******* calculate new features and stat
            /*******/
            cal_new_feature_nn(no_new_channel,tr_data,no_tr_data,EV_edbfm,no_new_channel);
            cal_new_feature_nn(no_new_channel,te_data,no_te_data,EV_edbfm,no_new_channel);

            printf("After cal_new_feature_nn\n");
            cal_L_ave_sd_data(tr_data,tr_data_dim,no_tr_data,&L_ave_sd,L_sd);


            /*******/
            /******* save files with new features
            /*******/
            sprintf(fname,"n%s\0\n",test_file_name);
            write_nn_file(fname,no_te_data,te_data_dim,te_data);

            sprintf(fname,"n%s\0\n",training_file_name);
            write_nn_file(fname,no_tr_data,tr_data_dim,tr_data);

            return;
```

```
}
/************/
/***
/*** FUNCTION: write_nn_file
/***
/************/
write_nn_file(fname,no_data,no_channel,in_data)
char *fname;
int no_data,no_channel;
struct in_data_str *in_data;
{
            int i,j,no_bit=1;


printf("write_nn_file fname=%s\n",fname);
            if( (ifp = fopen(fname,"w")) == NULL )
              outerr("sdfgjkl3tcf09");

            fprintf(ifp,"%d\n",no_data);
            fprintf(ifp,"%d\n",no_channel);
            fprintf(ifp,"%d\n",no_bit);           /* no of bit */

            for (i=0; i<no_data; i++)
            {
             fprintf(ifp,"%d\t",(in_data+i)->class);

             for (j=0; j<no_channel; j++)
              fprintf(ifp,"%f\t",(in_data+i)->data[j]);
             fprintf(ifp,"\n");
            }
            fclose(ifp);
            return;
}
/*******************************************************/
/*******
/*******    FUNCTION: sub_find_edbfm_2_class_by_nn
/*******
/*******************************************************/
sub_find_edbfm_2_class_by_nn(class,class_index,no_new_channel,edbfm)
struct class_str          *class;
int *class_index,no_new_channel;
double *edbfm;
{
            int i,j,k,l,cnt_tbl[5],point_array_cnt,bit_map_index[2];
            double       class_min[2];
            register float *rfp1,*rfp2;
            int ierr[2],inos[2],class2=2,cla_res_new_pnt;
            float *float_points_on_DB,*min_fp,tmin,ftmp;
            long longint;


            /*******/
            /******* assign memory & init
            /*******/
            longint=(class+class_index[0])->no_sample+(class+class_index[1])->no_sample;
            float_points_on_DB=(float *)malloc((unsigned)FLOAT*no_new_channel*longint);
            point_array_cnt=0;

            for (j=0; j<no_new_channel*no_new_channel; j++)
              edbfm[j] =0;

            /*******/
            /******* find pairs
            /*******/
            for (i=0; i<class2; i++)
            {
             for (j=0; j<(class+class_index[i])->no_sample; j++)
              {
                        /*******/
                        /******* find nearest sample classified as the other class
                        /*******/
                tmin=1e30;
                for (k=0; k<(class+class_index[1-i])->no_sample; k++)
                {
                 rfp1=(class+class_index[i])->data_float2+j*no_new_channel;
                 rfp2=(class+class_index[1-i])->data_float2+k*no_new_channel;
                 for (ftmp=l=0; l<no_new_channel; l++)
                  ftmp+= (rfp1[l]-rfp2[l])*(rfp1[l]-rfp2[l]);

                 if (ftmp<tmin)
                 {
                  tmin=ftmp;
                  min_fp=rfp2;
                 }
                }

               /*******/
               /******* find point on the decision boundary
               /*******/
               if (i==0)
                        find_point_on_nn_DB(class,class_index,
                        float_points_on_DB+no_new_channel*point_array_cnt,
                        no_new_channel,rfp1,min_fp,&cla_res_new_pnt);
               else
                        find_point_on_nn_DB(class,class_index,
                        float_points_on_DB+no_new_channel*point_array_cnt,
                        no_new_channel,min_fp,rfp1,&cla_res_new_pnt);

               point_array_cnt++;
              }         /* if after for j */
            }           /* for i */

            if (no_new_channel<5)
            {
             if( (ofp = fopen("stat.nndb","w")) == NULL )
              outerr("ERROR 45te");
             for (j=0; j<point_array_cnt; j++)
             {
               for (k=0; k<no_new_channel; k++)
                fprintf(ofp,"%.2f\t",float_points_on_DB[no_new_channel*j+k]);
               fprintf(ofp,"\n");
             }
             fclose(ofp);
            }


            save_DB_point(float_points_on_DB,no_new_channel,point_array_cnt);

            cal_grad_NN(class_index,float_points_on_DB,point_array_cnt,no_new_channel,edbfm);

            free(float_points_on_DB);
            return;
}
/************/
/***
/*** FUNCTION: cal_grad_NN
/***
/************/
save_DB_point(float_points,no_new_channel,no_points)
int no_new_channel,no_points;
float *float_points;
{
            int i,j;

            if( (ofp = fopen("nndb.DB","a")) == NULL )
             outerr("ERROR 3572gh : no stat file");

            for (i=0; i<no_points; i++)
            {
             for (j=0; j<no_new_channel; j++)
```

```
              fprintf(ofp,"%.2f\t",float_points[i*no_new_channel+j]);
             fprintf(ofp,"\n");
            }

            fclose(ofp);

            if( (ofp = fopen("nndb.DB2","a")) == NULL )
             outerr("ERROR 3572gh : no stat file");

            for (i=0; i<no_points; i+=2)
            {
             for (j=0; j<no_new_channel; j++)
              fprintf(ofp,"%.2f\t",float_points[i*no_new_channel+j]);
             fprintf(ofp,"\n");
            }

            fclose(ofp);
}

/************/
/***
/*** FUNCTION: cal_grad_NN
/***
/************/
int cnt_cal_grad_1=0;
cal_grad_NN(class_index,float_points,no_points,no_new_channel,edbfm)
int no_new_channel,*class_index,no_points;
float *float_points;
double *edbfm;
{
            float *float_incre;
            int i,j,k,result,largest,ignore_flag;
            double ratio,sum,*normal,ratio1,size_increment_grad=0.3;

            float_incre=(float *)malloc((unsigned)FLOAT*(no_new_channel+1));
            normal=(double *)malloc((unsigned)DOUBLE*(no_new_channel+1));

            for (i=0; i<no_points; i++)
            {

              classify_neural_network_one_sample(L_weight_array,float_points+no_new_channel*i,
              no_new_channel,L_no_stage,L_no_neuron_stage,&result,&largest,
              class_index[0],class_index[1],&ratio1);

             for (ignore_flag=sum=j=0; j<no_new_channel; j++)
             {
              for (k=0; k<no_new_channel; k++)
               float_incre[k]=float_points[no_new_channel*i+k];

              float_incre[j]+=L_sd[j]*size_increment_grad;

              classify_neural_network_one_sample(L_weight_array,float_incre,
              no_new_channel,L_no_stage,L_no_neuron_stage,&result,&largest,
              class_index[0],class_index[1],&ratio);

              normal[j]=(ratio-ratio1)/(L_sd[j]*size_increment_grad);
               sum+=normal[j]*normal[j];

             } /* for j*/

             sum=sqrt(sum);
             if (sum>0 && ignore_flag==0)
             {
              for (j=0; j<no_new_channel; j++)
               normal[j]/=sum;

              for (j=0; j<no_new_channel; j++)
               for (k=0; k<no_new_channel; k++)
                edbfm[j*no_new_channel+k]+=normal[j]*normal[k];
             }
```

```
          }

          free(normal);
          free(float_incre);
          return;
}
/************/
/***
/*** FUNCTION: find_point_on_nn_DB
/***
/************/
int                         local_repeat=0;
int                         local_counter=0;
float portion=0.01;
find_point_on_nn_DB(class,class_index,float_point_on_DB,no_new_channel,
float_point1,float_point2)
struct class_str          *class;
int no_new_channel,*class_index;
float *float_point_on_DB,*float_point1,*float_point2;
{
          int i,result,largest;
          float err,threshold=portion*L_ave_sd,*new_point;
          double ratio;

          threshold=portion*L_ave_sd;
          for (err=i=0; i<no_new_channel; i++)
          {
           err+=(float_point1[i]-float_point2[i])*(float_point1[i]-float_point2[i]);
           float_point_on_DB[i]=(float_point1[i]+float_point2[i])/2;
          }

          if (err<threshold && local_repeat>0)
          {
           local_repeat=0;
           return;
          }

          new_point=(float *)malloc((unsigned)FLOAT*no_new_channel);     /* assign memory */

          local_repeat++;
          classify_neural_network_one_sample(L_weight_array,float_point_on_DB,no_new_channel,
          L_no_stage,L_no_neuron_stage,&result,&largest,class_index[0],class_index[1],
          &ratio);

          for (i=0; i<no_new_channel; i++)
           new_point[i]=float_point_on_DB[i];

          if (ratio>1)
           find_point_on_nn_DB(class,class_index,float_point_on_DB,no_new_channel,
           new_point,float_point2);
          else
           find_point_on_nn_DB(class,class_index,float_point_on_DB,no_new_channel,
           float_point1,new_point);

          free(new_point);

          return;
}
/************/
/***
/*** FUNCTION: cal_new_feature_nn
/***
/************/
cal_new_feature_nn(no_new_channel,in_data,no_in_data,EV,dim_EV)
struct in_data_str *in_data;
double *EV;
int no_new_channel,dim_EV,no_in_data;
{
          int j,k,l;
          double      tmp[240];
```

```
             for (j=0; j<no_in_data; j++)
             {
              for (k=0; k<no_new_channel; k++)
               tmp[k]=(in_data+j)->data[k];

              for (k=0; k<no_new_channel; k++)
               {
                (in_data+j)->data[k]=0;

                for (l=0; l<dim_EV; l++)
                          (in_data+j)->data[k] += tmp[l] * *(EV+l*dim_EV+k);
               }
             }
             return;
}

/*******/
/*******
/******* FUNCTION: cal_stat_in_data
/*******
/*******/
cal_stat_in_data(in_data,no_channel,no_sample,mean,cov)
struct in_data_str *in_data;
double *cov,*mean;
int no_channel,no_sample;
{
             long i,j,k,l;


             /*******/
             /******* initialization */
             /*******/
    for (i=0; i<no_channel; i++)
               for (j=0; j<no_channel; j++)
                 *(mean+i)= *(cov+i*no_channel+j)=0;

             /*******/
             /*******calculate mean & covariance */
             /*******/

               for (l=0; l<no_sample; l++)
                for (j=0; j<no_channel; j++)       /* calculate covariance */
                {
                          *(mean+j) += (in_data+l)->data[j];

                 for (k=0; k<=j; k++)
                           *(cov+j*no_channel+k) += (in_data+l)->data[j]*(in_data+l)->data[k];
                }         /* for j */


             for (j=0; j<no_channel; j++)          /* calculate mean */
                          *(mean+j) /= no_sample;


             for (j=0; j<no_channel; j++)          /* calculate covariance */
              for (k=0; k<=j; k++)
               {
                 *(cov+j*no_channel+k) /= no_sample-1;
                 *(cov+j*no_channel+k) -= *(mean+j) * *(mean+k) * no_sample /(no_sample-1);
                 *(cov+k*no_channel+j)= *(cov+j*no_channel+k);
               }
             return;
}

/************/
/***
/*** FUNCTION: classify_neural_network_one_sample
/***
/************/
```

```
classify_neural_network_one_sample(weight_array,fdata,in_data_dim,no_stage,
no_neuron_stage,result,largest,class1,class2,ratio_class1_2)
int no_stage,*no_neuron_stage,in_data_dim,*result,*largest,class1,class2;
struct weight_array_str *weight_array;
double *ratio_class1_2;
float *fdata;
{
            char comment[40];
            double *in,*out,max= -1e10;
            int i,j,k,cnt,res;


            /***/
            /***  assign memory
            /***/
            for (j=i=0; i<=no_stage; i++)
             if (j<no_neuron_stage[i])
              j=no_neuron_stage[i];
            in=(double *)malloc((unsigned) j*DOUBLE);
            out=(double *)malloc((unsigned) j*DOUBLE);

            /***/
            /***  classify
            /***/
             for (j=0; j<in_data_dim; j++)
              (weight_array+0)->out[j]=in[j]=fdata[j];

             /***/
             /***  calculate out
             /***/
             for (j=1; j<no_stage; j++)
             {
              cal_out_array(in, (weight_array+j)->input_dim, (weight_array+j)->out, (weight_array+j)->out_dim,
              (weight_array+j)->weight, (weight_array+j)->bias_weight);

              /***/
              /***  copy to in for next stage
              /***/
              for (k=0; k<(weight_array+j)->out_dim; k++)
               in[k]=(weight_array+j)->out[k];
             }

             /***/
             /***  threshold and check error
             /***/
             cnt=0;

             for (j=0; j<no_neuron_stage[no_stage-1]; j++)
             {
              if ((weight_array+no_stage-1)->out[j]>=0.5)
              {
               cnt++;
               res=j;
              }

              if ((weight_array+no_stage-1)->out[j]>max)
              {
               max=(weight_array+no_stage-1)->out[j];
               *largest=j;
              }
             }
             if (cnt!=1)              /* otherwise reject */
              res= -1;

            *result=res;
            *ratio_class1_2=(weight_array+no_stage-1)->out[class1]/
            (weight_array+no_stage-1)->out[class2];

            free(out);
            free(in);
```

```
                return;
        }
```