```
/* nbc.c: Naive Bayesian Classifier */

#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include "nbc.h"

int No_Features, No_Classes, No_Instances;
int No_Trainers, No_Testers, Train_correct, Test_correct;
int linear[MAX_FEATURES];
int verbosity=0;
char domainName[MAX_NAME], resName[MAX_NAME];
int className[MAX_CLASSES], classIndex[MAX_CLASSES];
INSTANCE trainData[MAX_DATA], testData[MAX_DATA];
int classCount[MAX_CLASSES];
double g[MAX_CLASSES];
char trainDataFile[MAX_NAME], testDataFile[MAX_NAME];

VALCOUNT distribution[MAX_FEATURES][MAX_DATA];
int noValues[MAX_FEATURES];

FILE *logf;

struct timeval *ts_start, *ts_end;
struct timezone *tz;
int ts_time;

float
average(float *array, int size)
{
  float sum=0.0;
  int i;

  for (i=0; i < size; i++)
    sum += array[i];
  return sum / size;
} /* average */

void
initialize()
{
  int i, f;

  No_Trainers = read_train_data(); /* initializes trainData from domain.train*/
  for (i=0; i<No_Trainers; i++)   /* count classes in the train data */
    classCount[trainData[i].class]++;
  No_Testers  = read_test_data();  /* initializes testData  from domain.test */

  for (f=0; f<No_Features; f++)
    noValues[f] = 0;

  ts_start = (struct timeval * ) malloc(sizeof(struct timeval));
  ts_end = (struct timeval * ) malloc(sizeof(struct timeval));
  tz    = (struct timezone * ) malloc(sizeof(struct timezone));
} /* initialize */

parse_options(int argc, char *argv[])
{
  strcpy(domainName, argv[1]);
  if (argc >= 3 && (strcmp(argv[2], "-v")==0)) {
```

```
    if (argc == 3) verbosity = 1; /* default is the least verbosity */
    else sscanf(argv[3], "%d", &verbosity);
    open_log_file();
  }
} /* parse_options */

void
insert(INSTANCE x, FEATURE f)
{
  int i, j=0;
  int done = FALSE;
  CLASS c;

  for (j=0;( (j < noValues[f]) && (!done) ); j++) {
    if ( distribution[f][j].value >= x.value[f] )
      done = TRUE;
  }

  if ( done ) {
    if ( x.value[f] == distribution[f][j-1].value )
      distribution[f][j-1].count[x.class]++;
    else {
      /*fprintf(logf, "NEW VALUE= %5.2g  in feature %d\n", x.value[f], f); */
      for (i=noValues[f]; i>=j; i--) {
        distribution[f][i].value = distribution[f][i-1].value;
        for (c=0; c<= No_Classes; c++)
          distribution[f][i].count[c] = distribution[f][i-1].count[c];
      }
      distribution[f][j-1].value = x.value[f];
      distribution[f][j-1].count[x.class]++;
      noValues[f]++;
    }
  }
  else {
    distribution[f][j].value = x.value[f];
    distribution[f][j].count[x.class]++;
    noValues[f]++;
  }
}
void
train ()
{
  FEATURE f;
  int index;

  for (index=0; index < No_Trainers; index++) {
    printf("\n Now train instance %d of class %d .... \n", index, trainData[inde
x].class);
    for (f=0; f < No_Features; f++) {
      if (trainData[index].status[f] == KNOWN) {
        if (noValues[f] == 0)  {
          distribution[f][0].value = trainData[index].value[f];
          distribution[f][0].count[trainData[index].class] = 1;
          noValues[f] = 1;
        }
        else
          insert(trainData[index], f);
      }
    }
  }
```

```
} /* train */

float
p(VALUE x, FEATURE f, CLASS c)
{
  int i;                    /* instance index */
  float density=0.0;
  int found=FALSE;

  if (classCount[c] == 0) return(0.0);

  for (i=0; (i<noValues[f] && !found); i++)  {
    if (x == distribution[f][i].value) {
      found = TRUE;
      if ( (i==0) && (noValues[f]==1) )
        density = 1.0;
      else if ( i==0 )
        density = distribution[f][i].count[c] / (distribution[f][i+1].value-x)/
                                              classCount[c];
      else if ( i==noValues[f]-1 )
        density = distribution[f][i].count[c] / (x-distribution[f][i-1].value)/
                                              classCount[c];
      else
        density = distribution[f][i].count[c] /
                   (distribution[f][i+1].value-distribution[f][i-1].value) * 2 /
                                              classCount[c];
    }
  else if ( x < distribution[f][i].value ) {
    if ( i==0 )
      density = distribution[f][i].count[c] / (distribution[f][i].value-x) /
                                              classCount[c];
    else if ( i!=noValues[f]-1 )
      density = (distribution[f][i].count[c]+distribution[f][i-1].count[c]) /
                 (distribution[f][i].value-distribution[f][i-1].value) * 2 /
                                              classCount[c];
  }
  else if ( (x > distribution[f][i].value) && (i==noValues[f]-1) )
    density = distribution[f][i].count[c] / (x-distribution[f][i].value) /
                                              classCount[c];
  }

/*  if (verbosity>2)
    fprintf(logf, "c: %d, f: %d density: %g\n", className[c], f,density);
  if (verbosity>3)
    fprintf(logf, "   (lsc:%d, eqc:%d, slc:%d, lrg_smaller:%g, sml_larger:%g)\n"
, lsc, eqc, slc, lrg_smaller, sml_larger);   */

  return(density * 100);
} /* p */

float
nbc ()
{
  int i;
  FEATURE f;
  CLASS c, prediction;
  int correct_count=0;
  float test_acc=0;

  gettimeofday(ts_start, tz);
```

```
  for (i=0; i< No_Testers; i++) {
    prediction = 0;
    if (verbosity > 0) fprintf(logf,"\n");
    if (verbosity > 2) {
      fprintf (logf, "Test instance no: %d\n", i);
      for (f=0; f< No_Features; f++)
        fprintf(logf, "%g ", testData[i].value[f]);
      fprintf(logf, "\n");
      fflush(logf);
    }

    for (c=0; c <= No_Classes; c++) {
      g[c] = (float)classCount[c] / No_Trainers;
      for (f=0; f< No_Features; f++)
        if (testData[i].status[f] == KNOWN)
          g[c] = g[c] * p(testData[i].value[f], f, c);
      if (g[c] > g[prediction]) prediction = c; /* find c with max g[c] */
    }

    if (prediction == testData[i].class) correct_count++;

    if (verbosity > 1) {
      for (c=0; c <= No_Classes; c++)
        fprintf(logf, "g[%d]:%g ", className[c], g[c]);
      fprintf(logf, "\n");
    }
    if (verbosity > 0) {
      fprintf(logf, "%d: prediction: %d actual: %d\n",
              i, className[prediction], className[testData[i].class]);
      fflush(logf);
    }
  }

  gettimeofday(ts_end, tz);

  test_acc = (float)correct_count/No_Testers;
  if (verbosity>0)
    fprintf (logf, "Test accuracy: %g\n", test_acc);

  return (test_acc);
} /* nbc */

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

  FILE *fopen(), *resultf;
  float accuracy;
  int c, i, f;

  if (argc == 1) {
    printf("nbc domain -v verbosity\n");
    exit(1);
  }
  else
    parse_options(argc,argv);

  read_info();
```

```
  initialize();

  if (verbosity > 0) {
    fprintf(logf,"No_Features: %d\n", No_Features);
    fprintf(logf,"No_Classes: %d\n", No_Classes);
    fflush(logf);
  }
  if (verbosity > 3) {
    fprintf(logf, "train data:\n");
    log_data(trainData, No_Trainers);
    fprintf(logf, "test data:\n");
    log_data(testData, No_Testers);
  }
  if (verbosity > 1) {
    fprintf(logf,"No_Trainers: %d\n", No_Trainers);
    fprintf(logf,"No_Testers: %d\n", No_Testers);
    for (c=0; c <= No_Classes; c++)
      fprintf(logf, "classCount[%d]:%d ", className[c], classCount[c]);
    fprintf(logf, "\n");
    fflush(logf);
  }

  train ();
  for (f=0; f < No_Features; f++) {
    fprintf(logf, "\nFEATURE %d has %d distinct values .........................
.. \n", f, noValues[f]);
    for (i=0; i<noValues[f]; i++) {
      for (c=1; c<=No_Classes; c++)
        fprintf(logf, "Feature %d has %d of value %5.2g in CLASS %d\n", f, distr
ibution[f][i].count[c], distribution[f][i].value, c);
      fprintf(logf, "\n");
    }
  }

  accuracy = nbc();

  /* Accuracy and timing results */

  ts_time = (ts_end->tv_sec * 1000 - ts_start->tv_sec * 1000) +
    (ts_end->tv_usec / 1000 - ts_start->tv_usec / 1000);

  sprintf(resName,"%s.res", domainName);
  resultf = fopen(resName, "w");
  fprintf (resultf, "Accuracy: %5.3f Time: %d msec.\n",
           accuracy, ts_time);
  if (verbosity>0)
  fprintf (logf, "Accuracy: %5.3f\%% Testing: %d msec.\n",
           100 * accuracy, ts_time);
  fclose(resultf);
  if (verbosity>0) fclose(logf);
  return 0;
}
```