手書き数字認識の
計算機シミュレーション

2016 年 6 月

法政大学情報科学部

若原徹

目次

1　開発環境

【プログラミング言語】

C 言語を用いる。下記 URL：

http://cis.k.hosei.ac.jp/~wakahara/

の C Programming Language Lecture Notes にある講義ノートを参考に勉強すること。

特に，第 8 回講義ノート Cpro_8.htm で取り上げたプログラミングスタイルが重要である。以下に列挙するが，詳細は講義ノートを参照のこと。

［1］ 名前のつけかた
- グローバル変数にはわかりやすい名前を，ローカル変数には短い名前を。
- 関連性のあるものには関連性のある名前をつけて，統一しよう。
- 関数には能動的な名前を。
- 名前は的確に。

［2］ 式と文
- 構造がわかるようにインデントしよう。
- 自然な形の式を使おう。
- カッコを使ってあいまいさを解消しよう。
- 複雑な式は分割しよう。
- 明解に書こう。
- 副作用（++などの演算子）に注意。

［3］ 一貫性と慣用句
- インデントとブレースのスタイルを統一しよう。
- 慣用句によって一貫性を確保しよう。
- 多分岐の判定には else-if を使おう。

［4］ マクロやリテラルな数値の使い方
- 関数マクロはなるべく使うな。
- マクロの本体と引数はかっこに入れよう。
- 数値はマクロでなく定数として定義しよう。
- 整数でなく文字定数を使おう。
- オブジェクトサイズは言語に計算させよう。

【プログラミング開発環境】

Linux 上の gcc を用いることを推奨する。また，拡張子.c が付く C 言語のソースコードファイルを作成するテキストエディタには emacs が高機能で使い易い。

Linux で用いるコマンドについては，例えば，

http://rat.cis.k.hosei.ac.jp/article/linux/command.html

を参照すること。良く使うコマンド rm, cp, mv については，ホームディレクトリにあるログインシェルファイル.bashrc を

$ emacs ~/.bashrc &↵

で開いて，動作確認のオプション–i を含むエイリアスを追記しておくと良い。

```
alias rm='rm –i'
alias cp='cp –i'
alias mv='mv –i'
```

作業環境の立ち上げでは，まず，Linux でログインしたホームディレクトリの下に作業ディレクトリ patrec を

$ mkdir patrec↵

によって作成する。次に，作業ディレクトリ patrec に

$ cd patrec↵

により移動する。以降，主にこのディレクトリでプログラミング開発を行う。

ソースコードファイル sample.c のエディット・コンパイル・リンク・実行のしかたは次のようになる。

$ emacs sample.c &↵　← ソースコードファイル sample.c を作成する。
$ gcc –c sample.c↵　← –c（ハイフンシー）を付けると，コンパイルが実行されて，オブジェクトコードファイル sample.o ができる。
$ gcc sample.o↵　← リンクが実行されて，実行可能コードファイル a.out ができる。実行は　$ ./a.out↵
$ gcc sample.c↵　← コンパイル・リンクが実行されて，a.out ができる。実行は　$ ./a.out↵
$ gcc sample.c –o sample↵　← –o（ハイフンオー）の後に（任意の）実行可能コードファイル名，ここでは sample，を指定する。実行は　$ ./sample↵

数学関数を利用する場合，ヘッダファイル math.h をソースコードファイル sample.c で

#include <math.h>

でインクルードする。このソースコードファイル sample.c をコンパイル・リンクする場合は，次のようにオプション–lm（ハイフンエルエム）を必ず付けること。

$ gcc sample.c –o sample –lm↵

※Windows 上で開発する場合

統合開発環境 Visual C++ または Visual C++ Express を用いて Win32 コンソールアプリケーションを立ち上げること。

【ヘッダファイル mypgm.h とプログラミングの流れ】

下記 URL:

http://cis.k.hosei.ac.jp/~wakahara/

の C Program Source Codes の先頭にあるヘッダファイル

mypgm.h

を作業ディレクトリ patrec にダウンロードする。これから作成するプログラムには必ずこのヘッダファイルを

#include “mypgm.h”

でインクルードする。

mypgm.h には，#define を用いたマクロの定義と，外部変数の宣言および関数定義がな

されている。

```
/* Constant declaration */
#define MAX_IMAGESIZE   1024
#define MAX_BRIGHTNESS   255    /* Maximum gray level */
#define GRAYLEVEL          256      /* No. of gray levels */
#define MAX_FILENAME      256      /* Filename length limit */
#define MAX_BUFFERSIZE   256

/* Global constant declaration */
/* Image storage arrays */
unsigned char image1[MAX_IMAGESIZE][MAX_IMAGESIZE],
  image2[MAX_IMAGESIZE][MAX_IMAGESIZE];
int x_size1, y_size1,      /* width & height of image1*/
  x_size2, y_size2;        /* width & height of image2 */

/* Prototype declaration of functions */
void load_image_data( );          /* image input */
void save_image_data( );          /* image output*/
void load_image_file(char *);     /* image input */
void save_image_file(char *);     /* image output*/
```

ヘッダファイル mypgm.h を用いたプログラミングは次のような流れで行う。

[1] 入力画像格納用の 2 次元配列 image1[ ][ ]に，関数 load_image_data( )または load_image_file(filename)を用いて，ハードディスクから pgm 形式の画像を読み込む。読み込んだ画像の横および縦サイズはそれぞれ変数 x_size1, y_size1 に代入される。ただし，char filename[MAX_FILENAME]; で宣言して，適宜，ファイル名を設定する。

[2] image1[ ][ ]に格納された入力画像に，目的とする画像処理を関数で定義して施す。処理結果は何らかの数値データであったり，画像であったりする。処理結果の画像を出力する場合は，2 次元配列 image2[ ][ ]に格納する。格納する画像の横および縦サイズはそれぞれ変数 x_size2, y_size2 に設定する。

[3] 出力画像格納用の 2 次元配列 image2[ ][ ]の内容をハードディスクに pgm 形式で出力する場合は，関数 save_image_data( )または save_image_file(filename)を用いる。変数 x_size2, y_size2 に正しい値が入っていないとエラーとなる。

プログラミング例としては，入力画像の反転画像を出力するプログラム inverse.c：
http://cis.k.hosei.ac.jp/~wakahara/inverse.c
を参照のこと。例えば，画像 chess.pgm を用いる場合，次のように実行する。

```
$ gcc inverse.c –o inverse↵
$ ./inverse↵
  …
  Name of input image file? (*.pgm)：chess.pgm↵  ← 拡張子.pgm の入力画像名
  …
  Name of output image file? (*.pgm)：temp.pgm↵  ← 拡張子.pgm の任意の名前
  …
$
```

【画像の閲覧】

画像ビューワーには gimp を用いる。作業ディレクトリ patrec からは，次のようにして画像を表示する。

```
$ gimp chess.pgm &↵
```

※Windows 上での画像の閲覧には次の URL:
http://www.forest.impress.co.jp/library/software/irfanview/
から画像ビューワーIrfanView を入手すると良い。

2　手書き数字データの入手

【MNIST database】

米国 National Institute of Standards and Technology (NIST)が収集・整備した手書き数字のデータベースを用いる。元は白黒の 2 値画像であったものを、位置・大きさの正規化を施し、28×28 画素のサイズで濃淡画像に変換してある。

学習用に 60,000 サンプル、テスト用に 10,000 サンプルが含まれる。それぞれ 0～9 のサンプル数は以下の通りである。

表 1　数字毎のサンプル数

数字	“0”	“1”	“2”	“3”	“4”	“5”	“6”	“7”	“8”	“9”
学習	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
テスト	980	1135	1032	1010	982	892	958	1028	974	1009

MNIST データベースの詳細は次の URL を参照のこと。本データベースを用いて行われた認識実験でのエラー率も表形式で記載されている。

http://en.wikipedia.org/wiki/MNIST_database

http://yann.lecun.com/exdb/mnist/

上記データベースは授業支援システムの教材に

mnistPgm.tgz

として用意されているのでディレクトリ patrec にダウンロードし，tar コマンドを用いて解凍する。

```
$ tar zxvf mnistPgm.tgz↵
```

これにより，ディレクトリ patrec の下に新たにディレクトリ mnistPgm が作成されて，ディレクトリ mnistPgm 内に学習用 60,000 サンプルとテスト用 10,000 サンプルがバイナリ pgm 形式の画像ファイルとして格納される。

学習用の画像サンプルには，ldg 数字_通し番号_gray.pgm の名前を付してある。通し番号は左側 0 詰めの 4 桁数字である。ldg0_0001_gray.pgm～ldg9_5949_gray.pgm までの総数 60,000 枚の画像となる。

同様に，テスト用の画像サンプルには，tdg 数字_通し番号_gray.pgm の名前を付してある。tdg0_0001_gray.pgm～tdg9_1009_gray.pgm までの総数 10,000 枚の画像となる。

各画像ファイルは 1 サンプルの手書き数字の濃淡画像を格納しており，画像サイズは縦 28，横 28 である。バイナリ pgm 形式の場合，各画素の濃淡値 0～255 は 8 ビットの 2 進数で表現されるため，1 バイト／画素となっている。文字部分がより黒く（値 0 に近い），背景部分がより白く（値 255 に近い）なっている。

画像の閲覧は次のように行う。

```
$ gimp ./mnistPgm/ldg0_0001_gray.pgm &↵
```

3　前処理

一般に，パターン認識の処理の流れは

パターンの観測　→　前処理　→　特徴抽出　→　識別処理　→　カテゴリ出力

となっている。手書き数字認識を例に取ると，「パターンの観測」は手書き数字の画像入力である。最後の「カテゴリ出力」は入力された手書き数字画像の認識結果であり，クラス名 0～9 の出力である。「特徴抽出」と「識別処理」については後述する。

「前処理」であるが，これは特徴抽出に先立って，雑音除去や平滑化処理，さらには整形操作を施す処理であり，認識精度の向上に大きく貢献する。

手書き数字認識では，整形操作の一種である「位置と大きさの正規化」が重要な前処理である。位置と大きさの正規化処理にも様々な手法があるが，ここではモーメント法に基づく手法を採用する。手書き数字画像は 2 値で入力されているとする。すなわち，文字部分が黒（0），背景部分が白（255）である。

また，2 値画像を濃淡化するには Gaussian filtering を施すことが多い。例えば，下図に示す 5×5 サイズの 2 次元ガウスフィルタの重み係数を用いて，2 値画像から濃淡画像に変換する。

0.0	0.009	0.017	0.009	0.0
0.009	0.057	0.105	0.057	0.009
0.017	0.105	0.194	0.105	0.017
0.009	0.057	0.105	0.057	0.009
0.0	0.009	0.017	0.009	0.0

図 1　2 次元ガウスフィルタの重み係数の例

☆今回扱う MNIST データベースの場合, 予め位置と大きさの正規化を施された濃淡画像として格納されている。このため，本章での前処理の説明は参考として述べるものとする。

【文字の重心の算出】

画像内の文字部分（黒画素領域）の重心(gx, gy)を求める。これを用いて，重心が画像の中心(cx, cy)にくるように文字部分を平行移動する。この操作が位置の正規化となる。また，重心を求める操作は，画像の 1 次モーメント算出に相当する。

2 値文字画像の重心の算出は次のようなコーディングになる。

```
#define BLACK 0
#define WHITE 255
…
int main() {
```

```
   …
   int x, y;
   int count;
   double gx, gy;
   …
   gx = gy = 0.0;
   count = 0;
   for (y = 0; y < y_size1; y++) {
      for (x = 0; x < x_size1; x++) {
         if (image1[y][x] == BLACK) {
            gx += x;
            gy += y;
            count++;
         }
      }
   }
   gx /= (double)count;
   gy /= (double)count;
   …
}
```

【重心周りの平均距離の算出】

2 値画像内の文字部分（黒画素領域）の重心(gx, gy)を求めて，文字を構成する各黒画素から重心までの平均距離 rad を算出する。これを用いて，文字部分を重心の周りに一様に伸縮する。この操作が大きさの正規化となる。また，文字を構成する各黒画素から重心までの平均距離 rad を求める操作は，画像の 2 次モーメント算出に相当する。

各黒画素から重心までの平均距離 rad の算出は次のようなコーディングになる。gx, gy, count の値は，文字の重心算出の際に算出されているものとする。

```
…
#include <math.h>
…
int main() {
   …
   int x, y;
   double rad;
```

```
  …
  rad = 0.0;
  for (y = 0; y < y_size1; y++) {
    for (x = 0; x < x_size1; x++) {
      if (image1[y][x] == BLACK) {
        rad += sqrt((x-gx)*(x-gx)+(y-gy)*(y-gy));
      }
    }
  }
  rad /= (double)count;
  …
}
```

【双一次補間法と逆写像を用いた位置と大きさの正規化】

入力画像 image1[ ][ ]に位置と大きさの正規化を施した画像を image2[ ][ ]に格納するものとする。このとき，次のような計算手順を用いる。

[1] image2[ ][ ]の各位置(X,Y)が，位置と大きさの正規化処理の逆写像によって image1[ ][ ]のどの位置に対応するかを計算する。その位置を(tx, ty)と記すと，一般に tx, ty は整数ではなく実数となる。

[2] 位置(tx, ty)における image1[ ][ ]の濃淡値 gray を双一次補間法によって算出する。

[3] gray の値を，閾値 127.5 により，0 もしくは 255 に 2 値化する。この 2 値化された値 val を image2[Y][X]に代入する。

上記の処理は，以下のようなコーディングになる。gx, gy, rad の値は既に算出されているとする。ただし，正規化半径 RADIUS は 6.0 とした。

```
#define RADIUS 6.0
#define TH 127.5
#define WID 28     /* width */
#define HEI 28     /* height */
#define CX (WID/2)
#define CY (HEI/2)
…
int main() {
  …
  int X, Y;
```

```
    int m, n, val;
    double tx, ty, p, q, gray;
    …
    x_size2 = x_size1;
    y_size2 = y_size1;
    for (Y = 0; Y < y_size2; Y++) {
      for (X = 0; X < x_size2; X++) {
        /* inverse mapping by position and size normalization */
        tx = rad/RADIUS * (X-CX) + gx;
        ty = rad/RADIUS * (Y-CY) + gy;
        m = (int)floor(tx);
        p = tx - m;
        n = (int)floor(ty);
        q = ty - n;
        if (m >= 0 && m+1 < x_size1 && n >= 0 && n+1 < y_size1) {
          /* bilinear interpolation */
          gray = (1.0 - q)*((1.0 - p)*image1[n][m] + p*image1[n][m+1]) +
            q*((1.0 - p)*image1[n+1][m] + p*image1[n+1][m+1]);
          /* binarization */
          val = gray >= TH ? WHITE : BLACK;
        } else {
          val = WHITE;
        }
        image2[Y][X] = (unsigned char)val;
      }
    }
    …
}
```

4　特徴抽出

パターン認識では，特徴抽出が認識性能を大きく左右する。特徴空間が定まってしまえば，その特徴空間内でのサンプルの分布に基づき，統計的パターン認識理論により強力な識別器が構成できる。

しかし，最適な特徴抽出の理論は確立していない。このため，特に文字認識研究の分野で，これまでに多くの研究者の試行錯誤により様々な特徴抽出法が検討されてきた。

ここでは，素朴ながら最も基本的な特徴としての「メッシュ特徴」，および文字線の方向分布に着目した「濃淡勾配方向分布特徴」を取り上げる。

濃淡勾配方向分布特徴は，文字認識に限らず，広くコンピュータビジョンの分野でHOG(Histograms of Oriented Gradients)特徴量と呼ばれて活用されている。

http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

なお，本章で扱う手書き数字画像は，章 3 の前処理を施された濃淡画像であるとする。実際, 今回用いる MNIST データベースから用意したファイル mnistPgm.tgz に格納されている文字画像は位置と大きさの正規化を施された濃淡画像である。

4.1　メッシュ特徴

手書き数字画像を複数のブロック領域に分割し，各ブロック領域に含まれる画素の濃淡値の平均を算出する。このブロック毎の平均濃淡値をブロック数分だけ並べたベクトルをメッシュ特徴と呼ぶ。

具体的には，縦 28 画素×横 28 画素の手書き数字画像を 2×2 画素の正方ブロック領域に分割することにより，縦 14×横 14 の総数 196 個のブロック領域を生成する。各ブロックの平均濃淡値を並べた特徴ベクトルの次元数は 14×14＝196 となる。

メッシュ特徴 fv_mesh[196]の抽出処理は次のようなコーディングになる。

```
#define WID 28     /* width */
#define HEI 28     /* height */
#define X_BLOCK 2
#define Y_BLOCK 2
#define DIM ((WID/X_BLOCK)*(HEI/Y_BLOCK))
…
int main() {
  …
  int x, y, k;
  double fv_mesh[DIM];
  …
  for (k = 0; k < DIM; k++) {
```

```
    fv_mesh[k] = 0.0;
  }
  /* input of one image file */
  load_image_data( );
  …
  for (y = 0; y < y_size1; y++) {
    for (x = 0; x < x_size1; x++) {
      k = y/Y_BLOCK*(WID/X_BLOCK)+x/X_BLOCK;
      fv_mesh[k] += (double)image1[y][x];
    }
  }
  for (k = 0; k < DIM; k++) {
    fv_mesh[k] /= (double)X_BLOCK*Y_BLOCK;
  }
  …
}
```

手書き数字の各サンプル画像について，抽出したメッシュ特徴 fv_mesh[196]を double 型データとして，名前を付けてファイル出力しておくとよい。

そのために，まず，出力ファイル格納用のディレクトリ fv_mesh を作業ディレクトリ patrec の下に作成しておく。

```
$ mkdir fv_mesh↵
```

以下，学習用サンプル画像で数字が digit，通し番号が serial の画像から抽出したメッシュ特徴 fv_mesh[196]をファイル出力するためのコーディングを示す。

```
…
FILE *fp;
char filename[256];
int digit, serial;
static int l_pat[10] = {5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949};

…
for (digit = 0; digit < 10; digit++) {
  for (serial = 1; serial <= l_pat[digit]; serial++) {
    /* input of one image file */
```

```
      sprintf(filename, "./mnistPgm/ldg%d_%04d_gray.pgm", digit, serial);
      load_image_file(filename);
      /* extraction of mesh feature */
      …
      sprintf(filename, "./fv_mesh/lfv%d_%04d", digit, serial);
      fp = fopen(filename, "wb");
      fwrite(fv_mesh, sizeof(fv_mesh), 1, fp);
      fclose(fp);
      …
   }
}
```

MNIST データベースの全サンプルについて，上記のメッシュ特徴のファイル出力を行うと，ディレクトリ fv_mesh 内に lfv0_0001～lfv9_5949（総数 60,000 個）と tfv0_0001～tfv9_1009（総数 10,000 個）の特徴データファイルが作成される。double 型データを格納したバイナリファイルであり，各々のファイルサイズは 196×8=1568 バイトとなる。

4.2　濃淡勾配方向分布特徴

手書き数字画像を複数のブロック領域に分割し，各ブロック領域に含まれる画素位置での濃淡勾配方向を 45° 刻みで 8 方向に量子化して計数し，ブロック毎に 8 方向別の出現比率を算出する。この 8 方向別濃淡勾配方向の出現比率をブロック数分だけ並べたベクトルを濃淡勾配方向分布特徴と呼ぶ。

具体的には，縦 28 画素×横 28 画素の手書き数字画像を 4×4 画素の正方ブロック領域に分割することにより，縦 7×横 7 の総数 49 個のブロック領域を生成する。各ブロックでの 8 方向別濃淡勾配方向の出現比率を並べた特徴ベクトルの次元数は 7×7×8＝392 となる。

濃淡勾配の算出には Roberts cross operator を用いる。詳細は下記 URL を参照のこと。

http://en.wikipedia.org/wiki/Roberts_cross

具体的には次式を用い，画素(x, y)での濃淡勾配$\theta(x, y)$を算出する。なお，画素(x, y)での濃淡値を$g(x, y)$と表す。

$$\Delta u = g(x, y) - g(x + 1, y + 1),$$
$$\Delta v = g(x + 1, y) - g(x, y + 1),$$
$$\theta(x, y) = \tan^{-1}\left(\frac{\Delta u}{\Delta v}\right) - \frac{\pi}{4}.$$

$\theta(x, y)$は 8 方向に量子化し，整数 0～7 で表す。ただし，$\Delta u = \Delta v = 0$の場合は，方向なしとして，-1を与えることとする。

濃淡勾配方向分布特徴 fv_dir[49][8]の抽出処理は次のようなコーディングになる。

```
…
#include <math.h>
…
#define WID 28     /* width */
#define HEI 28     /* height */
#define X_BLOCK 4
#define Y_BLOCK 4
#define BLOCKS ((WID/X_BLOCK)*(HEI/Y_BLOCK))
#define DIRECTION 8
#define PI 3.141592654
…
/* prototype declaration of functions */
int roberts8(int y, int x);
…
int main() {
  …
```

```
    int x, y, k;
    int code;
    double fv_dir[BLOCKS][DIRECTION];
    …
    for (k = 0; k < BLOCKS; k++) {
        for (code = 0; code < DIRECTION; code++) {
            fv_dir[k][code] = 0.0;
        }
    }
    for (y = 0; y < y_size1-1; y++) {
        for (x = 0; x < x_size1-1; x++) {
            code = roberts8(y, x);
            if (code != -1) {
                k = y/Y_BLOCK*(WID/X_BLOCK)+x/X_BLOCK;
                fv_dir[k][code] += 1.0;
            }
        }
    }
    for (k = 0; k < BLOCKS; k++) {
        for (code = 0; code < DIRECTION; code++) {
            fv_dir[k][code] /= (double)X_BLOCK*Y_BLOCK;
        }
    }
    …
}

/* definition of functions */
int roberts8(int y, int x)
/* 8-directional gradient at (x, y) */
{
    double delta_RD, delta_LD;
    double norm, angle;
    int code;

    delta_RD = image1[y][x+1] - image1[y+1][x];
    delta_LD = image1[y][x] - image1[y+1][x+1];
```

```
    norm = sqrt(delta_RD*delta_RD + delta_LD*delta_LD);

    if (norm == 0.0) {
        code = -1;
        return code;    /* no gradient */
    }
    if (delta_RD == 0.0) {
        if (delta_LD > 0) code = 3;
        else code = 7;
    } else {
        angle = atan2(delta_LD, delta_RD);
        if (angle > 7.0/8.0*PI) code = 5;
        else if (angle > 5.0/8.0*PI) code = 4;
        else if (angle > 3.0/8.0*PI) code = 3;
        else if (angle > 1.0/8.0*PI) code = 2;
        else if (angle > -1.0/8.0*PI) code = 1;
        else if (angle > -3.0/8.0*PI) code = 0;
        else if (angle > -5.0/8.0*PI) code = 7;
        else if (angle > -7.0/8.0*PI) code = 6;
        else code = 5;
    }
    return code;
}
```

手書き数字の各サンプル画像について，抽出した濃淡勾配方向分布特徴 fv_dir[16][8]を double 型データとして，名前を付けてファイル出力しておくとよい。

そのために, まず, 出力ファイル格納用のディレクトリ fv_dir を作業ディレクトリ patrec の下に作成しておく。

```
$ mkdir fv_dir↵
```

以下，学習用サンプル画像で数字が digit，通し番号が serial の画像から抽出した濃淡勾配方向分布特徴 fv_dir[49][8]をファイル出力するためのコーディングを示す。

```
…
FILE *fp;
char filename[256];
int digit, serial;
```

```
static int l_pat[10] = {5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949};
…
for (digit = 0; digit < 10; digit++) {
   for (serial = 1; serial <= l_pat[digit]; serial++) {
      /* input of one image file */
      sprintf(filename, "./mnistPgm/ldg%d_%04d_gray.pgm", digit, serial);
      load_image_file(filename);
      /* extraction of histogram of oriented gradients feature */
      …
      sprintf(filename, "./fv_dir/lfv%d_%04d", digit, serial);
      fp = fopen(filename, "wb");
      fwrite(fv_dir, sizeof(fv_dir), 1, fp);
      fclose(fp);
      …
   }
}
```

MNIST データベースの全サンプルについて，上記の濃淡勾配方向分布特徴のファイル出力を行うと，ディレクトリ fv_dir 内に lfv0_0001～lfv9_5949（総数 60,000 個）と tfv0_0001～tfv9_1009（総数 10,000 個）の特徴データファイルが作成される。double 型データを格納したバイナリファイルであり，各々のファイルサイズは 7×7×8×8=3136 バイトとなる。

5　識別器の構成

学習データを用いて特徴空間での各クラスのサンプルの分布を求めて，その統計的ないし確率的振舞いを分析し，テストサンプルに対して誤り確率最小となるクラス名を出力する識別器を構成するのが，ベイズ理論に基づく統計的パターン認識の基礎的考え方である。

学習サンプル数が十分にある場合，経験則としてはクラス当たりのサンプル数が特徴次元数の 2 桁以上ある場合，統計的パターン認識に基づく識別器は極めて有力である。近年良く利用されるのが，次の 3 種類の識別器である。

[1] 各クラスの特徴空間での分布が正規分布であると仮定して，ベイズ理論に基づく最適 2 次識別関数から導かれる「擬似ベイズ識別関数」

[2] 入力層に特徴ベクトル成分ないし原データを並べて出力層に各クラスの尤度を並べる feedforward 型の多層ネットワークで，中間層ユニットの活性化関数に sigmoid 関数などを用いて非線形な決定境界を誤差逆伝播法で学習する「2 層ニューラルネット」さらには特徴抽出機能も実現し，勾配消失問題や過学習を克服する「深層学習」

[3] カーネル関数を用いて元の特徴空間から高次元（しばしば無限大）の特徴空間に写像し，マージン最大となる超平面境界を凸 2 次計画法で決定する「サポートベクターマシン」

上記[1]では，擬似ベイズ識別関数の近似として定式化される「改良投影距離」も良く用いられる。

また，上記[3]のサポートベクターマシンの実装を試みる場合は次の URL：
http://www.csie.ntu.edu.tw/~cjlin/libsvm/
が参考になる。

ここでは，上記 3 種類とは異なる，次の 2 種類の識別器を取り上げる。

1 つ目は，最も素朴な識別器であり，テストサンプルと各クラスの平均ベクトルとの距離を調べて，最小距離値を持つクラスに分類する「最近傍平均分類」（Nearest-mean classification）である。

2 つ目は，学習サンプルの特徴ベクトルをすべて登録しておき，テストサンプルとの距離の小さい順に k 個の学習サンプルを選択し，それら k 個の中でのクラスの多数決を行う「k-NN 分類」（k-nearest-neighbor classification）である。

5.1　最近傍平均分類

最近傍分類の処理の流れは次のようになる。

① 学習データを用いて，特徴空間での各クラスのサンプルの分布から各クラスの平均ベクトルを算出する。手書き数字認識の場合，カテゴリ 0～9 の分類であり，クラス数は 10 である。特徴空間の次元数を DIM とすると，算出された 10 個の平均ベクトルは double 型の 2 次元配列 lfv_mean[10][DIM]に格納することができる。

② テストデータから認識すべきサンプルを一つずつ順に入力する。テストサンプルから抽出された特徴ベクトルを tfv[DIM]とする。

③ 特徴ベクトル tfv[DIM]と各クラス k の平均ベクトル lfv_mean[k][DIM]とのパターン間距離，例えば，ユークリッド距離，を算出して，それら 10 個の値を小さい順にソートする。

④ 最小距離を持つクラス名を当該テストサンプルの認識結果として出力する。

上記の最近傍平均分類の処理は次のようなコーディングになる。

平均ベクトルの格納には，2 次元配列を用いずに，double **lfv_mean; の宣言により，動的なメモリ確保を利用している。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
…
#define N_CLASSES 10
#define DIM 196
…
/* prototype declaration of functions */
double calc_Euc_dist(double *, double *, int);
void bsort_ascend(double *, int *, int);
…
int main() {
  FILE *fp;
  char filename[256];
  int i, j, k;
  int l_digit, l_serial;
  int t_digit, t_serial;
  double lfv[DIM];
  double tfv[DIM];
  double **lfv_mean;
  double dist[N_CLASSES];
  int label[N_CLASSES];
  …
  static int l_pat[10] = {5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949};
  static int t_pat[10] = {980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009};
```

```
…
/* dynamic memory allocation */
lfv_mean = (double **)malloc(sizeof(double *) * N_CLASSES);
for (k = 0; k < N_CLASSES; k++) {
   lfv_mean[k] = (double *)malloc(sizeof(double) * DIM);
}
…
/* input of learning feature vectors and generation of mean vectors */
/* initialization of lfv_mean */
for (k = 0; k < N_CLASSES; k++) {
   for (i = 0; i < DIM; i++) {
      lfv_mean[k][i] = 0.0;
   }
}
for (l_digit = 0; l_digit < N_CLASSES; l_digit++) {
   for (l_serial = 1; l_serial <= l_pat[l_digit]; l_serial++) {
      sprintf(filename, "./fv_mesh/lfv%d_%04d", l_digit, l_serial);
      fp = fopen(filename, "rb");
      fread(lfv, sizeof(lfv), 1, fp);
      fclose(fp);
      for (i = 0; i < DIM; i++) {
         lfv_mean[l_digit][i] += lfv[i];
      }
   }
}
for (k = 0; k < N_CLASSES; k++) {
   for (i = 0; i < DIM; i++) {
      lfv_mean[k][i] /= l_pat[k];
   }
}
/* input of one test feature vector */
while (1) {
   do {
      printf("¥ndigit (exit = -1) = ");
      scanf("%d", &t_digit);
      if (t_digit == -1 || t_digit < 0 || t_digit >= 10) exit(1);
```

```
        printf("serial = ");
        scanf("%d", &t_serial);
      } while (t_serial < 1 || t_serial > t_pat[t_digit]);
      sprintf(filename, "./fv_mesh/tfv%d_%04d", t_digit, t_serial);
      fp = fopen(filename, "rb");
      fread(tfv, sizeof(tfv), 1, fp);
      fclose(fp);
      /* calculation of Euclidean distances between input and mean feature vectors */
      for (k = 0; k < N_CLASSES; k++) {
        dist[k] = calc_Euc_dist(tfv, lfv_mean[k], DIM);
      }
      /* sorting of distances in the ascending order */
      for (k = 0; k < N_CLASSES; k++) {
        label[k] = k;
      }
      bsort_ascend(dist, label, N_CLASSES);
      …
      /* display of recognition result */
      …
    }

    /* release of memory allocation */
    for (k = 0; k < N_CLASSES; k++) {
      free(lfv_mean[k]);
    }
    free(lfv_mean);
    …
}

/* definition of functions */
double calc_Euc_dist(double *fv1, double *fv2, int dim)
/* calculation of Euclidean distance between fv1 and fv2 */
{
    int k;
    double dist;
```

```
    dist = 0.0;
    for (k = 0; k < dim; k++) {
        dist += (fv1[k]-fv2[k])*(fv1[k]-fv2[k]);
    }
    return sqrt(dist);
}

void bsort_ascend(double *array, int *label, int dim)
/* bubble sorting in the ascending order */
{
    int i, j, k;
    double work;

    for (i = 0; i < dim-1; i++) {
        for (j = i + 1; j < dim; j++) {
            if (array[i] > array[j]) {
                work = array[i];
                array[i] = array[j];
                array[j] = work;
                k = label[i];
                label[i] = label[j];
                label[j] = k;
            }
        }
    }
}
```

5.2　k-NN 分類

k-NN 分類についての概説は下記 URL を参照のこと。

http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

k-NN 分類の処理の流れは次のようになる。

① 全学習サンプルをテンプレートとして登録する。ここでは，学習データの全特徴ベクトルを登録することに相当する。それらを集合{ lfv[DIM] }と記す。

② テストデータから認識すべきサンプルを一つずつ順に入力する。テストサンプルから抽出された特徴ベクトルを tfv[DIM]とする。

③ テストサンプルの特徴ベクトル tfv[DIM]と集合{ lfv[DIM] }内の個々のテンプレートとのパターン間距離を算出して，値の小さい順にソートし，上位 k 個のテンプレートを選択する。

④ それら k 個のテンプレートでクラスの多数決を行い，最大個数のテンプレートを含むクラスを当該テストサンプルの認識結果として出力する。

上記の k-NN 分類の処理は次のようなコーディングになる。1 つ 1 つのテストサンプルの認識毎に，全テンプレートの読み込みが必要となる。また，上記③のソーティングでは，実際は，上位 k 個のみ求めればよい。

```
#include <stdlib.h>
#include <math.h>
…
#define N_CLASSES 10
#define DIM 196
#define MAX_CANDIDATES 60000
#define KNN 3
…
/* prototype declaration of functions */
double calc_Euc_dist(double *, double *, int);
void quicksort(double *, int *, int, int);
void dswap(double *, int, int);
void iswap(int *, int, int);

…
int main() {
  FILE *fp;
```

```
char filename[256];
int i, j, k;
double lfv[DIM];
double tfv[DIM];
int l_digit, l_serial;
int l_samples;
int t_digit, t_serial;
int hit[N_CLASSES];
double *dist;
int **sample_id;
int *label;
…
static int l_pat[10] = {5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949};
static int t_pat[10] = {980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009};
…
/* dynamic memory allocation */
dist = (double *)malloc(sizeof(double) * MAX_CANDIDATES);
label = (int *)malloc(sizeof(int) * MAX_CANDIDATES);
sample_id = (int **)malloc(sizeof(int *) * 2);
for (k = 0; k < 2; k++) {
   sample_id[k] = (int *)malloc(sizeof(int) * MAX_CANDIDATES);
}
…
/* input of one test feature vector */
while (1) {
   do {
      printf("¥ndigit (exit = -1) = ");
      scanf("%d", &t_digit);
      if (t_digit == -1 || t_digit < 0 || t_digit >= 10) exit(1);
      printf("serial = ");
      scanf("%d", &t_serial);
   } while (t_serial < 1 || t_serial > t_pat[t_digit]);
   sprintf(filename, "./fv_mesh/tfv%d_%04d", t_digit, t_serial);
   fp = fopen(filename, "rb");
   fread(tfv, sizeof(tfv), 1, fp);
   fclose(fp);
```

```
  /* k-NN classification starts */
  l_samples = 0;
  /* input of all learning feature vectors one by one */
  for (l_digit = 0; l_digit < N_CLASSES; l_digit++) {
    for (l_serial = 1; l_serial <= l_pat[l_digit]; l_serial++) {
      sample_id[0][l_samples] = l_digit;
      sample_id[1][l_samples] = l_serial;
      /* input of one learning feature vector */
      …
      /* calculation of Euclidean distances between two feature vectors */
      dist[l_samples] = calc_Euc_dist(tfv, lfv, DIM);
      l_samples++;
    }
  }
  /* sorting of distances in the ascending order */
  for (k = 0; k < l_samples; k++) {
    label[k] = k;
  }
  quicksort(dist, label, 0, l_samples-1);
  …
  /* counting of class occurrences in KNN candidates */
  for (k = 0; k < N_CLASSES; k++) {
    hit[k] = 0;
  }
  for (k = 0; k < KNN; k++) {
    i = label[k];
    j = sample_id[0][i];
    hit[j]++;
  }
  /* majority voting in hit[ ] */
  …
  /* display of recognition result */
  …
}
/* release of memory allocation */
for (k = 0; k < 2; k++) {
```

```
    free(sample_id[k]);
  }
  free(sample_id);
  free(label);
  free(dist);
  …
}

/* definition of functions */
double calc_Euc_dist(double *fv1, double *fv2, int dim)
/* calculation of Euclidean distance between fv1 and fv2 */
{
  int k;
  double dist;

  dist = 0.0;
  for (k = 0; k < dim; k++) {
    dist += (fv1[k]-fv2[k])*(fv1[k]-fv2[k]);
  }
  return sqrt(dist);
}


void quicksort(double *array, int *label, int left, int right)
{
  int i, last;

  if (left >= right) return;
  dswap(array, left, (left+right)/2);
  iswap(label, left, (left+right)/2);
  last = left;
  for (i = left+1; i <= right; i++) {
    if (array[i] < array[left]) {
      ++last;
      dswap(array, last, i);
      iswap(label, last, i);
```

```
      }
   }
   dswap(array, left, last);
   iswap(label, left, last);
   quicksort(array, label, left, last-1);
   quicksort(array, label, last+1, right);
}


void dswap(double *array, int i, int j)
{
   double temp;

   temp = array[i];
   array[i] = array[j];
   array[j] = temp;
}


void iswap(int *array, int i, int j)
{
   int temp;

   temp = array[i];
   array[i] = array[j];
   array[j] = temp;
}
```

6　認識実験

認識実験では，学習段階→テスト段階の順に実験を進める。

学習段階では，学習用データ ldg0_0001_gray.pgm～ldg9_5949_gray.pgm を用いて特徴抽出と識別器の構築を行う。続いて, テスト段階では, テスト用データ tdg0_0001_gray.pgm～tdg9_1009_gray.pgm から 1 つずつ特徴抽出してから識別器で分類して 1 位から 10 位までの候補クラス名を出力する。

ここでは，原画像の読み込みから行うのではく，既に章 4 で作成した全サンプルの特徴ベクトル群 lfv0_0001～lfv9_5949 および tfv0_0001～tfv9_1009 を読み込んで認識実験を行う。

実験結果の整理では，次のような項目を評価する。

[1] 認識率：正解が第 1 位候補となった率

[2] 累積分類率：正解が上位第 k 位候補までに含まれた率（第 1 位累積分類率＝正解率）
（ただし，k-NN 分類では第 1 位候補しか求まらないので累積分類率は評価できない。）

[3] Confusion matrix（混同行列）: 行方向に入力サンプルのクラス名，列方向に出力の第 1 候補クラス名を記し，各セルに出現サンプル数を挿入したもの

以下では，認識実験を進める上での基本的な処理のいくつかについてコーディングのしかたを示す。これらを参考に認識実験のプログラムを完成すること。

【特徴データファイルの連続読み込み】

個々のファイルの読み込みではファイル名を指定する必要がある。これをキーボードから入力するのでなく，プログラミングでファイル名を指定するには次のように行う。ただし，学習用データおよびテスト用データに含まれる各数字のサンプル数を予め配列に格納しておく。

```
#include <stdio.h>
#include "mypgm.h"
…
#define DIM 196
…
int main() {
  FILE *fp;
  char filename[256];
  int digit, serial;
  double fv[DIM];   /* mesh feature */
  …
```

```
  static int l_pat[10] = {5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949};
  static int t_pat[10] = {980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009};

  for (digit = 0; digit <= 9; digit++) {
    for (serial = 1; serial <= l_pat[digit]; serial++) {
      /* input of learning mesh feature vector */
      sprintf(filename, "./fv_mesh/lfv%d_%04d", digit, serial);
      fp = fopen(filename, "rb");
      fread(fv, sizeof(fv), 1, fp);
      fclose(fp);
      …
    }
  }
  …
}
```

【累積分類率の計算】

テストでは，テスト用データの特徴データファイル tfv0_0001～tfv9_1009 を 1 つずつ読み込んで識別器で分類して 1 位から 10 位までの候補クラス名を出力する。

その際，正解が第何位に出現したかを調べて集計すると，累積分類率を評価することができる。

次のようなコーディングになる。ここでは，テスト用データの特徴データファイルを読み込んで認識実験を行う場合について示す。ただし，識別器は最近傍平均分類とする。

```
#include <stdio.h>
#include "mypgm.h"
…
#define N_CLASSES 10
#define DIM 196
…
int main() {
  …
  FILE *fp;
  char filename[256];
  int digit, serial;
  int t_total;
```

```
int k;
int rank;
double fv[DIM];
double dist[N_CLASSES];
int label[N_CLASSES];
double cum_rate[N_CLASSES+1][N_CLASSES] = {0.0};
…
static int l_pat[10] = {5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949};
static int t_pat[10] = {980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009};
…
t_total = 0;
for (digit = 0; digit <= 9; digit++) {
   t_total += t_pat[digit];
   for (serial = 1; serial <= t_pat[digit]; serial++) {
      /* input of mesh feature vector */
      sprintf(filename, "./fv_mesh/tfv%d_%04d", digit, serial);
      fp = fopen(filename, "rb");
      fread(fv, sizeof(fv), 1, fp);
      fclose(fp);
      …
      /* calculation of inter-pattern distances by nearest mean classification */
      …
      /* sorting of distances in the ascending order */
      for (k = 0; k < N_CLASSES; k++) {
         label[k] = k;
      }
      bsort_ascend(dist, label, N_CLASSES);
      /* check where the correct answer appears */
      for (k = 0; k < N_CLASSES; k++) {
         if (label[k] == digit) {
            rank = k;
            break;
         }
      }
      cum_rate[digit][rank] += 1.0;
      cum_rate[N_CLASSES][rank] += 1.0;
```

```
        }
        /* calculation of cumulative classfication rates for each digit */
        for (k = 0; k < N_CLASSES; k++) {
            cum_rate[digit][k] *= 100.0/t_pat[digit];
        }
        for (k = 1; k < N_CLASSES; k++) {
            cum_rate[digit][k] += cum_rate[digit][k-1];
        }
    }
    /* calculation of cumulative classfication rates for all digits */
    for (k = 0; k < N_CLASSES; k++) {
            cum_rate[N_CLASSES][k] *= 100.0/t_total;
    }
    for (k = 1; k < N_CLASSES; k++) {
        cum_rate[N_CLASSES][k] += cum_rate[N_CLASSES][k-1];
    }
    …
}
```

完