パターン認識 – 課題 3

GENG Haopeng 611710008
Email: kevingenghaopeng@gmail.com

Department of Intelligent Systems, Nagoya University

2018 年 6 月 4 日

概要

今回の課題は、パーセプトロンによる学習の基礎および応用である。
さらに、3 種の認識方法を理解した上、プログラムを作成し、実行結果による評価と考察を行う。
そのほか、課題のソースコードが既に Github Repository に掲載されるため、参照あるいは実行することは可能である。

1　パーセプトロンによる学習 (基本編)

1.1　実験理論およびアルゴリズム

この課題は、パーセプトロンの収束定理を利用し、重みベクトルを修正しつつ、線形分離可能なパターンであれば、解領域内の重みベクトルに到達する。いわゆる、全ての学習パターンが正しく識別される。アルゴリズムの手順は以下のように表す：

1 重みベクトル $\boldsymbol{w}$ の初期値を決める
2 パターン集合 $\mathcal{X}$ から順番で一つのパターン ω_i を選ぶ
3 識別関数

$$g(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x}$$

によって識別を行い。正しく識別できなかった場合、次のように重みベクトルを修正する。

$$\boldsymbol{w}' = \boldsymbol{w} + \rho \boldsymbol{x} (\omega_i \in \mathcal{X}, g_i(\boldsymbol{x}) \neq \max\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), \cdots, g_n(\boldsymbol{x})\})$$

$$\boldsymbol{w}' = \boldsymbol{w} - \rho \boldsymbol{x} (\omega_j \in \mathcal{X}, g_j(\boldsymbol{x}) = \max\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), \cdots, g_n(\boldsymbol{x})\})$$

4 上の処理 [2], [3] 全てのパターンに対して繰り返す
5 $\forall \omega_i \in \mathcal{X}$, 正しく識別できたら終了、誤りがあるときは [2] に戻る

ただし、最大値が複数個出現する場合、計算順で最初に出た重みのみ修正する。

1.2　プログラムに工夫した部分

誤り識別するとき、重みベクトルの修正は以下のように

ソースコード 1　correct weight vector

```
/* calculate until converge, while converge condition is all pattern have become stable */
 while(conv(flag, LEARNING_NUM)){

   if(p >= LEARNING_NUM) p = p % LEARNING_NUM;

   for(j = 0; j < CLU_NUM; j++){
     eval[j] = multi(p_arr[p].data, w[j], CLU_NUM);
     printf("Evaluation of Pattern %d : %f\n", p, eval[j]);
   }

   /* Corr is pattern's true Class */
   /* Err is Evaluated Class, if condition is met , err == -1 */
   int corr = p_arr[p].pclass;
   int err = judge_max(eval, CLU_NUM, corr);

   printf("Right Class = %d,Error Class = %d\n",corr,err);

   if(err != -1){
     flag[p] = 1;
     for(j = 0; j < W_DIM; j++){
       w[corr][j] += rho * p_arr[p].data[j];
       w[err][j] -= rho * p_arr[p].data[j];
     }
   }

   /* Weight Condition is met, change flag to let result converge */
   else{
     flag[p] = 0;
   }

   printf("========\n");

   for(i = 0; i < CLU_NUM; i++){
     for(j = 0; j < W_DIM; j++){
       printf("%.0f ",w[i][j]);
     }
     printf("\n");
   }
   p++;
 }
```

1.3 プログラム実行例

1.3.1 重みベクトルの修正

与えられたパターンと初期重みを用い、以下のように出力される。ただし、20 回の繰り返し演算を略し、結果のみを表す。

```
$./pl learning_data.list init_weight.dat weight.dat
.
.
.
Evaluation of Pattern 1 by g(0) : 13.000000
Evaluation of Pattern 1 by g(1) : 9.000000
Evaluation of Pattern 1 by g(2) : 12.000000
Right Class = 0,Error Class = -1
========
9 2 0
2 1 5
-2 6 2
```

出力結果について、

1 各評価関数の値
2 所属すべきクラス、識別関数による識別結果
3 修正された重みベクトル

という順番で示され、ゆえに、合計 20 回の修正で、学習結果である重みベクトルは $\begin{bmatrix} 9 & 2 & 0 \\ 2 & 1 & 5 \\ -2 & 6 & 2 \end{bmatrix}$ になった。

1.3.2 未知パターン識別

上記の重みベクトルを用い、未知パターンの識別を行った。

```
$./pl_rec weight.dat unknown.dat
Value of g[0]: 13.000000
Value of g[1]: 14.000000
Value of g[2]: 14.000000
recog result == -1
```

ゆえに、識別関数の最大値が複数個（g[1], g[2]）あるため、識別できない結果になった。

1.4 考察

1.4.1 初期ベクトルの変化による影響

$\boldsymbol{W_{init}}$ をランダムに 500 個を選び、識別結果を評価した結果は表 1 で示す。

表 1　500 個の初期重みベクトルの実験結果

識別結果	w_1	w_2	w_3	Unrec
識別数	131	84	180	105
比率	26.2%	16.8%	36.0%	21.0%

ただし、各重みベクトル $\boldsymbol{w_i}$ の要素 w_{ij} に対し, 以下の条件を満たす。

$$w_{ij} \in [-5, 5], w_{ij} \in \boldsymbol{Z}$$

よって、正しい識別率は 26.2% になった。

1.4.2 定数 ρ の変化による影響

定数 ρ を 0.1 から 1.5 まで、刻み幅を 0.1、および $\rho = 2.0, 3.0, 4.0, 5.0, 6.0$ に設定し実験した結果は表 2 のように表す。

表 2　ρ を変化させてできた実験結果

ρ	識別結果	ρ	識別結果	ρ	識別結果	ρ	識別結果	識別率
0.1	√	0.6	√	1.1	Err(w_2)	2.0	√	
0.2	√	0.7	√	1.2	√	3.0	Unrec	
0.3	√	0.8	√	1.3	√	4.0	√	75.0%
0.4	√	0.9	√	1.4	√	5.0	√	
0.5	Unrec	1.0	Unrec	1.5	Unrec	6.0	√	

実験結果により、$\rho = 0.5, 1.0, 1.5, 3.0$ の時、識別できない場合は多いため、初期重みとパターンの値は整数であるは原因であると考える。

2　プロトタイプの最近傍決定則による文字認識

2.1　実験目的および理論

この課題は、学習パターンからプロトタイプを生成する。そして、未知パターンを先用いた学習パターンではなく、各プロトタイプとの距離の計算し、一番近いプロトタイプは識別結果になる。ただし、プロトタイプの生成方法に関して、今回の課題は、重心 (各種パターンの平均値) とする。

$$\boldsymbol{P} = \frac{1}{n}\sum_{i=0}^{n}\boldsymbol{X_i}$$

ただし、生成されたプロトタイプは平均値はほとんど小数になるが、未知パターンの各次元とプロトタイプの各次元の絶対値の和とその絶対値の平方の和の大小関係は同じであることが分かった。即ち、Euclidean 距離を計算しなくとも正しい結果が得られることである。

下記の実行例と識別結果は、「距離」は各次元の差の絶対値の和として理解すれば良い。

2.2　プログラムの解説

2.2.1　プログラムの流れ

1. 学習パターンをファイルから入力し、各パターンに応じて、プロトタープを生成し保存。
2. 未知パターンを入力し保存。
3. 各プロトタイプと未知パターンの距離を計算し、ソートして距離の最小値とそれに相応しいパターンをリターンする。
4. 戻り値は結果として認められる。

2.3　プログラム実行例

pattern7-5.dat を例として、以下のように出力される。

```
$../proto_nnm proto.list unrecog_data/pattern7-5.dat
==> unrecog_data/pattern7-5.dat <==
1 1 1 1 1
1 1 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 1 0 0

==> Valuation module ==<
Distence with Prototype [2] : 6.800000
Distence with Prototype [7] : 5.600000
Distence with Prototype [9] : 11.000000

==> Result of PATTERN by Prototype Method <==
Recognition Result == 7
```

出力結果について、

1 未知パターンの二値表示
2 各プロトタイプとの距離
3 識別結果

の順番で表す。ゆえに、pattern7-5 の識別結果は 7 であることが分かった。

2.4 識別結果

プロトタイプを用いる最近傍法を用いて、3 種類の数字で各 2 文字ずつを認識した結果は表 3 のように表す。

表 3　プロトタイプを用いる最近傍法の識別結果

評価項目 / 未知パターン	各プロトタイプとの距離			識別結果 (最近傍プロトタイプ)
	Prototype2	Prototype7	Prototype9	
pattern2-5	4.4	8.4	11.0	prototype2
pattern2-5	5.2	9.6	11.8	prototype2
pattern7-5	6.8	5.6	11.0	prototype7
pattern7-6	12.4	9.2	10.2	prototype7
pattern9-5	9.6	9.2	7.4	prototype9
pattern9-6	8.8	12.0	10.2	prototype2

表 3 によって、pattern9-6 以外の場合は正しく識別される。識別率は $5 \div 6 \approx 83.3\%$ である。

3 考察

3.1 プログラムの頑丈性

K 近傍法に応じて、K をいくつかの値を用い考察した結果は表 4 のように表す。

表 4　K 近傍法の考察結果

評価結果 K の値 / 未知パターン	$K=1$	$K=3$	$K=5$	$K=7$	$K=9$	正確率
pattern2-5	√	√	√	√	√	100%
pattern2-6	√	√	√	√	√	100%
pattern7-5	√	√	√	Err(2)	√	80%
pattern7-5	√	√	Rej(7,9)	Rej(7,9)	Err(9)	40%
pattern9-5	√	√	√	√	√	100%
pattern9-6	Err(2)	√	√	Err(2)	Err(2)	40%
正確率	83.3%	100%	83.3%	50%	67.7%	76.7%

上記の表 4 によると、以下のことが分かった。

1 $K=3$ の時識別率が一番良い、$K=7$ の時識別率が一番悪い。
　(i) 必ずしも K が大きいほうが良い。K の選択法は評価結果にかなり影響を与える。

2 各未知パターンに応じて、pattern2-5, pattern2-6, pattern9-5 の識別率は相当に良いが、pattern7-5 と pattern9-6 の識別率が不十分である。誤識別率の高いパターンにおいて:
　(i)pattern7-5 では、K 近傍の中に 7 と 9 が出現率が一緒の場合が多くて、数字 7 か数字 9 か識別できなくなる。

(ii) 一方、pattern9-6 では、数字 2 と誤識別される場合が多い。いわゆる、数字 2 と相似である。

実際 K 近傍法を応用する時、K 値を決め方の一つは、K-fold Cross Validation(K-分割交差検証) という方法がよく使われている [1]。

3.2 プログラムの互換性

パターンデータの保存方法に関して、以下の構造体の利用した。

```
typedef struct {
  double **data; /*Character Datas (2 Demention Variable Length Array)*/
  int width; /*Width of Data matrix*/
  int height; /*Height of Data Matrix*/
  int pattern;/*Pattern type*/
} character_data;
```

構造体の中に、パターンである二値画像をデータの可変長配列、配列の横幅、縦幅、パターン種類で保存される。それぞれの要素を各学習ファイルから獲得できるような処理を行い、データ種類やデータ量が数多くなる時もう対応できるようになる。

さらに、パターンが数字でなく文字の場合、[pattern] の変数型を変更すれば良い、他の種類のパターンも対応できるようになる。

3.3 プログラムの欠点

gprof コマンドを用いてプログラムの関数呼び出し回数を解析した結果。プログラムの時間複雑度は $O(n^2)$ 数量級である。

だが、今回の課題では、全ての学習データを一元構造体配列に格納するとこになったが、距離を計算した後どの学習パターンを値を得るなどの処理が重くなる。一つの提案として、全ての学習データを連結リストの構造で保存すれば、距離を参照し、相応しい学習パターンを得る処理がポインターで実現すれば、時間複雑度と後の拡張プログラム作業量も減らせると考えられる。

4 パターン認識の使用例

4.1 音声認識の分野

人間が話す音声が機械に認識される一連の処理には、自然音声の音声学特徴を抽出し、音声モデルに応用されるのは一般的である。よく使われる特徴量は：

1 線形予測係数
2 ケプストラム
3 MFCC(メル周波数ケプストラム係数)

である。これらの特徴量を利用し、実際の文字配列と対応する音声モデルが生成可能になる [2]。

参考文献

[1] Refaeilzadeh P, Tang L, Liu H. Cross-validation[M]//Encyclopedia of database systems. Springer US, 2009: 532-538.

[2] 今井聖. (1995). 音声認識. 情報 電子入門シリーズ 16.