パターン認識 – 課題 5

GENG Haopeng 611710008
Email: kevingenghaopeng@gmail.com

Department of Intelligent Systems, Nagoya University

2018 年 7 月 18 日

概要

今回の課題は、決定木手法で特徴量の特徴空間を分割することによって、分類木や回帰木を生成する。そして、得られた識別能力の低い識別器を用い、Adaboost 手法で強識別器を構築し、識別結果を評価する。

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、解説が備え、参照あるいは実行することは可能である。

1 決定木による識別

1.1 実験理論

決定株というのは、とある次元の特徴量を注目し、一定な閾値を定め、それを踏まえてパターンを枝分け、つまりパターンを分離する識別器である。ただし、今回の実験では、閾値の評価基準は Gini 不純度である。

Gini 不純度というのは，1 つの集合内に，異なるクラスのデータがどれだけ混在しているかを表す数値。計算式は:

$$\sum_{i=1}^{C} \frac{n_i}{N}(1 - \frac{n_i}{N}) = 1 - \sum_{i=1}^{C}(\frac{n_i}{N})^2$$

ただし，C はクラス数，n_i は，そのクラス i のサンプル数である。

さらに、今回の実験は二分木であるため、この 2 分割による Gini 不純度の総和を計算する。

$$G_{total} = \frac{p_{left}}{N} G_{left} + \frac{p_{right}}{N} G_{right}$$

各取り得る閾値から、最も Gini 不純度の小さいのは分離度は一番良いのである。なお、各次元において、Gini 不純度の最小である次元上の分離性能は一番高いとも言える。学習のアルゴリズムは以下のように表す：

1 学習パターンの各次元の特徴量でソートする。
2 ソートした特徴量で、取り得る閾値を決定する。
3 取り得る閾値を一個づつ代入し、Gini 不純度を計算する。今時点の Gini 不純度は、全て計算された Gini 不純度の中で一番低いのではないかを判断する。
   – 3.1 一番低いの場合、現時点の閾値を決定木の閾値と認め、更新を行う。
   – 3.2 そうでない場合、従来の閾値を認める。
4 全ての次元の閾値とそれに応じる Gini 不純度をソートし、一番 Gini 不純度の低い次元を判定次元と認め、その次元上の閾値もパターン間の分離境界と認める。

なお、今回の実験条件は表 1 のように表す。

表 1. 実験条件

学習パターン数	評価パターン数	パターン次元数	クラスター数
8	2	3	2

1.2 プログラムに工夫した部分

1 各分割の Gini 不純度の計算の手順は以下のように示す。

ソースコード 1. Gini impurity

```
double gini_imp(double lower, double upper, Node *arr, Pattern *ptn, int n, int class){
int i;
int count_all = 0;
int count_right = 0;
double g = 0;
/* Count Patttern Numbers under threshold and count whose pclass is the same as var <class>*/
for(i = 0; i < n; i++){
  if(lower <= arr[i].value && arr[i].value <= upper){
    count_all ++;
    if(ptn[arr[i].index].pclass == class){
      count_right ++;
    }
  }
}

if(count_all == 0)
  return 0;
else{
  double p = (count_right * 1.0 ) / (count_all * 1.0);
  g = p * (1 - p);
  return g;
}
}
```

2 なお、今回のパターンは二分割である、片方のクラスが分かれば、もう一方のクラスはその反転であることとする。

ソースコード 2. Judge Class under Threshold

```
int judge_class(Node *p, Pattern *ptn, double thre, int n){
  int i = 0;
  int cnt = 0;// number of patterns under threshold
  int cnt_l1 = 0;//number of patterns under threshold && class == 1
  int cnt_r1 = 0;//number of patterns above threshold && class == 1

  int res;
  for(i = 0; i < n; i++){
    if(p[i].value < thre){
      cnt ++;
      if(ptn[p[i].index].pclass == 1) cnt_l1 ++;
    }
    else if(ptn[p[i].index].pclass == 1 && p[i].value > thre) cnt_r1 ++;
  }
  if((cnt_l1 * 1.0 / cnt * 1.0) >= (cnt_r1 * 1.0) / ((n - cnt) * 1.0)) res = 1;
  else res = 2;
  return res;
}
```

1.3 プログラム実行例

1.3.1 各次元において閾値決定

3 次元、2 クラス、8 個の学習パターンを入力として、出力結果は以下のように表す。

ソースコード 3. Recognition

```
feat_index clsss threshold mini_gini
0 2 1.500000 0.187500
1 1 2.500000 0.187500
2 1 2.500000 0.100000
```

結果により、各次元において、閾値はそれぞれ 1.5、2.5、2.5 で、各次元の不純度はそれぞれ 0.1875、0.1875、0.1 であることがわかった。

1.3.2 特徴量の選択および未知パターン識別

未知パターン $(1,2,1)->w1, (3,3,2)->w2$ を代入し、出力は以下のようである。

ソースコード 4. Recognition

```
Sorted Forest
2 1   2.500000 0.100000
0 2   1.500000 0.187500
1 1   2.500000 0.187500

 Judge Dimention [2]
p_arr[0] --> Cluster 1
p_arr[1] --> Cluster 1
```

識別結果から、最も gini 不純度の低い次元は Dim2 であるため、それを評価標準として用いる。

その結果、二つのパターンともクラス 1 と識別され、正解率が 50.0% であることがわかった。識別パターンは Dim2 上の差別がそれほど大きくないのは誤識別に導いた結果と想定している。

なお、他の次元においては、あえて正しい識別結果になれるということがわかった。ゆえに、決定株のような識別機の識別能力が低いということが推定できるようになった。

2 AdaBoost による識別

2.1 プログラム実行例

2.2 実験理論

上記の実験によって、一つだけの識別株の識別能力が低いことがわかった。ゆえに、複数個の弱識別器を利用し、より良い性能の高い識別器を構築するのは今回実験の目標である。

パターンの数と識別器の数が少ないため、既知サンプルの重みを踏まえてリサンプリングを行う。新しいサンプルを利用し、いくつかすでにできた識別器の性能を計り、その性能によって各識別器に重みをつける。さらに、複数個の弱識別器を利用し、重み付きの多数決で、未知パターンの種類を判別する。具体的にいうと、Adaboost という手法で、識別器の誤ったサンプルの重みを増やすことにより、次の識別器の重み決定に影響を与える。その結果、識別能力の高い識別器の信頼性が高くなり、全体的な識別率が上がれることがわかった。

ただし、今回の実験において、弱識別器は課題 5.1 で、二次元でできた二つの決定株を利用する。

表 2. iris 実験条件

学習パターン数	評価パターン数	パターン次元数	クラスター数	リサンプリング数
8	2	2	2	100

2.3 プログラムに工夫した部分

1 重み付きサンプリングの処理手順は以下のように示す。

ソースコード 5. Gini impurity

```
void resamp(Samp_Node *s, Samp_Node res, int len, double rate){
  int i;
  int m;
  if(rate <= s[0].cum_w){
    for(m = 0; m < Dim;  m++) res.data[m] = s[0].data[m];
  }
  else{
    for(i = 1; i < len; i++){
      if(s[i - 1].cum_w < rate && rate <= s[i].cum_w) {
        for(m = 0; m < Dim;  m++) { res.data[m] = s[i].data[m]; }
        break;
      }
    }
  }
}
}
```

2 Adaboost の主な手順：

- とある次元の特徴量によってソートする。
- 閾値を用いて誤り率を算出する。
- 誤り率を用いて識別器の重みを算出する。
- 誤り率を用いて、各パターンが誤識別かどうかによって、パターンごとに重みを修正する。

ソースコード 6. Judge Class under Threshold

```
for(m = 0; m < LEARNING_NUM; m++){
      array[m].value = p_arr[m].data[n];
      array[m].index = p_arr[m].pclass;
    }
    qsort(array, LEARNING_NUM, sizeof(Node), comp_array); // Sort
    for(m = 0; m < LEARNING_NUM; m++){
      if(array[m].value < forest[n].threshold){
        if(array[m].index != forest[n].class){
          error ++;
          eps += weight[m];
        }
      }
      else if(array[m].value >= forest[n].threshold){
        if(array[m].index == forest[n].class){
          error ++;
          eps += weight[m];
        }
      }
      printf("error_%d_eps_%f\n", error, eps);
    }
    h_w[n] = 0.5 * log((1.0 - eps) / eps);      // Generate Recognizer's error rate
    for(m = 0; m < LEARNING_NUM; m++){
      /* Weight Correction */
      /* For valuies under threshold,
         if class is right, Correct weight to make it smaller
         if class is wrong , Correct weight to make it bigger*/
      if(array[m].value < forest[n].threshold){
        if(array[m].index != forest[n].class){
          weight[m] = weight[m] * exp( - h_w[n] * (-1.0));
        }
        else{
          weight[m] = weight[m] * exp( -h_w[n] * (1.0));
        }
      }

      else if(array[m].value >= forest[n].threshold){
        if(array[m].index == forest[n].class){
          weight[m] = weight[m] * exp( - h_w[n] * (-1.0));
        }
        else{
          weight[m] = weight[m] * exp( -h_w[n] * (1.0));
        }
      }
    }
    normalization(weight, LEARNING_NUM);// Normalization for new weights
}
```

2.4　プログラム実行例

以下は重み付きサンプリング、Adaboost および識別評価の実行結果である。なお、サンプリング数は 100 として、クラスごとに 50 個のサンプルがあり、重み付きサンプリングの結果は表 3 で示す。

表 3. Sampling Result

パターン番号	クラス	パターン	個数
1	1	(1, 4)	13
2	1	(2, 6)	12
3	1	(3, 2)	12
4	1	(5, 8)	13
5	2	(4, 5)	11
6	2	(6, 3)	9
7	2	(7, 6)	15
8	2	(8, 1)	15

ソースコード 7. Adaboost and Recognition

```
0 1 5.500000 0.100000 1.045371
1 2 7.000000 0.214286 -0.300899

Pattern [0]     3.000000 4.000000
Recon Result 1
Pattern [1]            6.000000 7.000000
Recon Result 2
```

よって、二つの識別器の重みはそれぞれ 1.045371、-0.300899 であるため、決定性作用になっているのは一番目の決定株であることがわかった。そして、試験結果は完全正解 (100%) になった。

なお、課題 5.1 の中で利用した手法で識別した結果は、以下のように示す。

ソースコード 8. Decetion Stump Result

```
Sorted Forest
0 5.500000 0.100000
1 7.000000 0.214286

Judge Dimention [0]
p_arr[0] --> Cluster 1
p_arr[1] --> Cluster 2
```

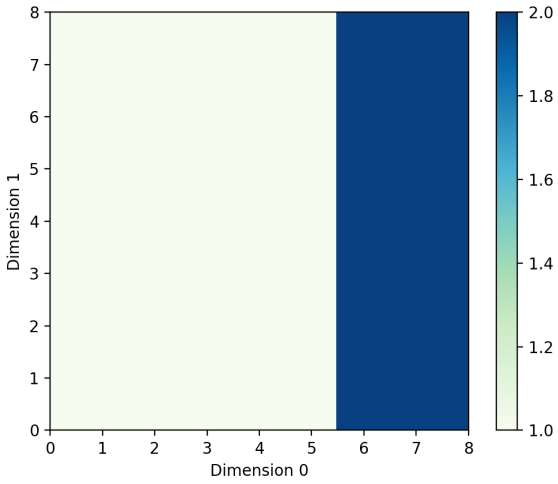
よって、今回の実験において、弱識別器を一個だけ利用する場合と強識別器の場合には、結果が変わらず、同じ識別率であることがわかった。提供された実験データでは、どちらの識別能力が高いことが区別できないため、Adaboost による強識別器の識別性能の評価は考察の方で詳しく説明する。

2.5 考察

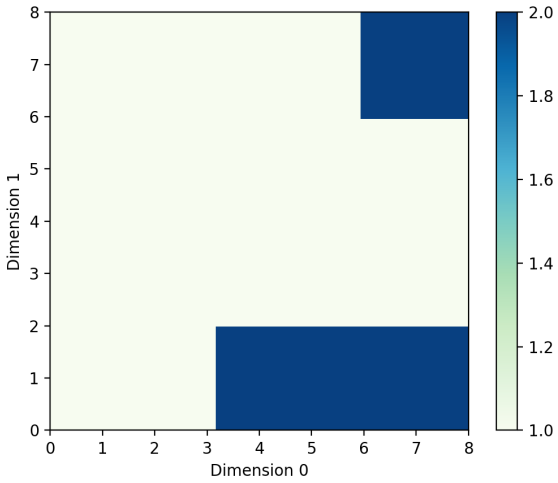
高次元のパターンにおいて、決定木 (弱識別器) の弱点は、課題 5.1 の手法を用いると、一番 Gini 不純度の低い次元しか使わないので、他の次元の含める情報がかなり損失してしまう。なお、閾値左右の不純度が一致する場合 (対応クラスも一致)、両辺が同じクラスになるため、可識別クラスの数がさらに落ちてしまうという弱点もある。

しかし、Adaboost 手法を用いれば、弱識別器の選び方が理想的であれば、各次元の特徴も最後の多数決に貢献可能である。いわば、特徴量より良い効率的に利用する。なお、同じ次元においても、複数個の識別器が共同作用するため、$A|B|A$ のようがパターンも識別可能になる。より、弱識別器と比べ、全体的な識別能力が優れたことが想定できる。

図 1 は、両種類の識別器を二次元上のイメージである。決定株は (i) のような分布にしか可能できないか、Adaboost を用いると、(ii) のような分布も識別可能である。



(i) Decision Stump



(ii) Adaboost

図 1. 弱識別器と強識別器の識別領域