パターン認識・課題5

課題 5.1 決定株(Decision Stump)による識別

課題 5.1.1 決定株1:一つの特徴量に注目したしきい値決定

決定木の枝分かれが1段しかないものを,決定株という.
(決定木を根っこのところで切り倒した切り株のイメージ.)

決定株識別器では,ある特徴量を1つ選び(コマンドライン引数で0, 1, ...を渡す),それに対して1つしきい値を決定する.
手計算の演習では,0/1の特徴量だったのでしきい値の決定をする必要がなかったが,実際は,どの値で2つに分割するかを決めなければならない.
各特徴量でそれぞれしきい値を決定し,最もうまく分類できる特徴量としきい値の組を求めることにより, その特徴量のしきい値に対する大小でクラス分類を行う.

しきい値の決定方法:Gini不純度による決定

Gini不純度は,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.

Gini不純度は$\sum_{i=1}^{C} \frac{n_i}{N}(1-\frac{n_i}{N}) = 1 - \sum_{i=1}^{C}(\frac{n_i}{N})^2$で求められる.
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.

各分割でGini不純度を計算し,各分割に含まれるサンプルの割合を重みとして重み付け和を取る.
$G_{total} = \frac{p_{left}}{N} G_{left} + \frac{p_{right}}{N} G_{right}$

Gini不純度によってしきい値を決定するアルゴリズムは以下の通り.

1. 選んだ特徴量が小さい順になるよう,学習データをソートする
2. 選んだ特徴量の最小値から最大値の範囲で閾値を変えながらループ
    1. すべての特徴量を,しきい値をもとに2クラスに分類する.
    2. しきい値より小さい側のGini不純度と,しきい値以上側のGini不純度を重み付きで計算し,和を求める.
3. Gini不純度が最小になるしきい値を決定する.
4. 識別器は,どの(何個目の)特徴量を使ったかと,求めたしきい値の値とを保存する

レポートには,以下のサンプルデータに対し,各特徴量を選択した時のGini不純度を記載すること.

実行例
```
$ ./gini 0
特徴量:0 しきい値:??? Gini不純度:0.???
$ ./gini 1
特徴量:1 しきい値:??? Gini不純度:0.???
$ ./gini 2
特徴量:2 しきい値:??? Gini不純度:0.???
```

特徴量**0**	特徴量**1**	特徴量**2**	クラス
1	1	1	ω_1
2	2	2	ω_1
3	2	2	ω_1
2	3	1	ω_1

1	4	3	ω_2
2	3	2	ω_2
1	3	3	ω_2
1	1	3	ω_2

課題 5.1.2 決定株2:特徴量の選択

実際に決定株を使う場合，どの特徴量を利用するかが重要．ここでは，どの特徴量を利用して分類するかをGini不純度をもとに判別する．

1. 各特徴量についてGini不純度を最小にするしきい値と，その時のGini不純度を求める．
2. 最もGini不純度が小さい特徴量と，その時のしきい値をファイルに保存する．

課題 5.1.1で求めた各Gini不純度の最小の特徴量が選択されたことを確認する．

```
 実行例
$ ./stump_train train.dat trained.model
特徴量:0 しきい値:??? Gini不純度:0.???
特徴量:1 しきい値:??? Gini不純度:0.???
特徴量:2 しきい値:??? Gini不純度:0.???
選択: 特徴量:? しきい値:??? Gini不純度:0.???
（ファイル(trained.model)に特徴量番号としきい値を保存）
```

課題 5.1.3 決定株による認識プログラムの実行

評価用データを決定株によって認識する．

1. 決定株のパラメータを読み込む．
2. 決定株が保持している特徴量の値を取得する．
3. その値が，決定株が保持している特徴量よりも大きいか小さいかによって（1 or 2）を出力する．

```
 実行例
$ ./stump_test trained.model test1.dat
認識結果: ?
$ ./stump_test trained.model test2.dat
認識結果: ?
```

ただし，以下の例それぞれを，test1.dat, test2.datとする．

特徴量**0**	特徴量**1**	特徴量**2**	クラス
1	2	1	ω_1
3	3	2	ω_2

ヒント

決定株のデータ構造例

```
typedef struct Stump_{
    int feat_index; /* 特徴量の番号 */
    float threshold; /* しきい値 */
} Stump;
```

Gini不純度計算の例

```
#include <stdio.h>
#include <stdlib.h>

#define N 8 /* サンプル数 */
#define P 3 /* 次元数（特徴量の数） */

typedef struct Stump_{
        int feat_index; /* 特徴量の番号 */
        float threshold; /* しきい値 */
} Stump;


float values[N];
int indices[N];

int main(){
        int i;
        float min_gini = 1.0f;
        Stump st;
        float features[N][P] = /* 値を入れて初期化 */
        int labels[N] = /* 値を入れて初期化 */

        /* 使う特徴量を決める */
        st.feat_index = /* 特徴量の次元*/;

        /* 特徴量の値にしたがって並べ替える */
        /* values配列に値をコピーする */
        /* indices配列を準備する */
        for(i=0; i<N; i++){
                values[i] = features[i][st.feat_index];
                indices[i] = i;
        }
        /* values配列の値でindices配列をソートする */

        for(/* 特徴量の小さいものから順にしきい値とする */){
                float g1, g2;
                float sum_gini;

                g1 = /* しきい値未満のサンプルについてGini不純度を計算 */
                g2 = /* しきい値以上のサンプルについてGini不純度を計算 */

                /* Gini不純度の和を計算 */
                sum_gini = g1 + g2;

                if(/* sum_giniが最小なら */){
                        /* sum_giniを更新, その時のしきい値features[indieces[i]][st.feat_index]
                }
        }

        printf("しきい値: %f\n", st.threshold);
}
```

課題 5.2 AdaBoostによる識別

AdaBoostの学習は以下の手順からなる.

1. Bootstrapサンプルの生成

2. 生成したBootstrapサンプルを用いた弱識別器の学習
3. 重み付き学習誤差の計算
4. 弱識別器の信頼度の計算
5. 重み付き学習誤差がしきい値よりも小さければ終了
6. サンプル重みの計算

上記手順を繰り返すことで，多数の弱識別器を得る．
この演習では，弱識別器には5.1で実装した決定株を用いる．

動作確認には，講義スライドP.25とほぼ同様の配置の以下のサンプルを用いよ．

特徴量**0**	特徴量**1**	クラス
1	1	ω_1
2	2	ω_1
3	2	ω_1
2	3	ω_2
1	4	ω_1
2	3	ω_2
1	3	ω_2
1	1	ω_2

課題 5.2.1 Bootstrapサンプルの生成

AdaBoostでは，学習サンプルに重みが付けられる．
ここでは，各学習サンプルの重みの割合に応じた「重み付きサンプリング」を行う．

例えば以下のように重みがつけられているとする．

サンプル	1	2	3	4
重み	0.1	0.2	0.6	0.1

この学習サンプルに対し，10回のサンプリングを行うと，
1,2,2,3,3,3,3,3,3,4の10個の学習サンプルが得られる．

このような，重み付きサンプリングを実装せよ．

ヒント：重み付きサンプリング

M回の重み付きサンプリングは以下のアルゴリズムによって実現できる．

1. サンプルの重みの総和が1になるように正規化する．
2. 累積重みを計算する．
3. [0,1]の範囲で一様サンプリングを行い，乱数pを1つ得る（一様分布から乱数を生成．）
4. 生成した乱数pと累積重みを比較し，乱数pの値を超えない最大の累積重みをもつ番号iを得る．
5. 生成したサンプルに，i番目の特徴量を追加する．
6. サンプリングした個数がM個になるまで3-5を繰り返す．

例：
前述の例で言うと，重みは既に1に正規化されているので，Step2から．
累積重みを計算すると以下のようになる．

サンプル	1	2	3	4
重み	0.1	0.2	0.6	0.1
累積重み	0.1	0.3	0.9	1.0

ここから，M回のループm=0,1,...M-1を行う．
ただし，サンプリング結果を格納する配列 samples[M][P] は予め確保しておく．（ここではPは特徴量次元数）

まず，乱数を1個生成する．

この時,

```
p = (rand()%100)/100.0
```

でも良いが，厳密には擬似乱数ではダメなので（ダメな理由は何？），性能の良いメルセンヌ・ツイスタ(http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/mt19937ar.html)を利用するのがおすすめ.

次に，例えば1つ生成した乱数が$p = 0.5$だったとすると，
累積重みと1つずつ比較していき，p>0.3でp<0.9なので，サンプル3を選択する.
選択したサンプル3を，samples[m]に格納する.

課題 5.2.2 AdaBoost学習プログラムの作成

クラスごとのBootstrapサンプルのサンプリング

5.2.1 のサンプリングを，全学習データに適用すると，重みのバランスによっては，
片方のクラスが全く選択されない，という状況が発生しうる.
そうすると，決定木の学習ができない.
これを避けるために，クラスごとにサンプリングを行う.

具体的には，クラスごとにサンプリングを行う回数を求め（講義スライドP.20），それぞれのクラスのサンプルだけを集めたデータ中から，5.2.1のサンプリングを行う.

重み付き学習誤差の計算

作成した決定株を用いて，（Bootstrapサンプルではなく）元の学習データを全て分類する.
分類を間違えたサンプルに対し，そのサンプルの重みの和を取ることで，学習誤差とする.
$\epsilon_t = \sum_{i=1}^{N} w[i] I(y_i \neq h_t(x_i))$
ただし，$I(a \neq b)$は，$a \neq b$ならば1，それ以外は0を返す.

弱識別器の信頼度の計算

学習誤差が小さければ，その弱識別器の信頼度は高く，そうでなければ低くする.
計算式は以下の通り
$\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

サンプル重みの計算

現在の弱識別器において，分類を間違えていれば，そのサンプルの重みを大きく，正解していれば重みを小さくする.
i番目の学習サンプルに対する重みの更新式は以下の通り.

$w[i] = w[i] \exp(-\alpha_t y_i h_t(x_i))$
ただし，α_tはt番目の（今作成した）識別器h_tの信頼度.

ヒント：決定株の構造

```
typedef struct Stump_{
    int feat_index; /* 特徴量の番号 */
    float threshold; /* しきい値 */
    float importance; /* 決定株の信頼度 */
} Stump;
```

課題 5.2.3 AdaBoostによる認識プログラムの実行

AdaBoostによる認識処理は以下の通り.

$H(\mathbf{x}) = \mathrm{sign}\left[\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right]$
以下の未知サンプルに対して認識処理を行え.

特徴量**0**	特徴量**1**	クラス

3	4	ω_1
6	7	ω_2

また, 他の例についても認識を試してみるのも良い.

提出期限

2018/7/30

レポート提出の注意は, こちら (../report.html)

戻る