システム創成 プロジェクトⅠ
画像認識 演習（第2回）

システム創成情報工学科

演習担当：尾下 真樹、齊藤 剛史、斎藤 寿樹

徳永 旭将、宮野 英次、藤本 晶子

プロジェクトⅠ　日程(1)

- 1週目 画像認識(1)
  - 3限目　講義(特徴量を使った識別)(佐藤)
  - 4限目　講義(演習説明)(尾下or齊藤or斎藤or徳永or宮野or藤本)
  - 5限目　演習
- 2週目 画像認識(2)
  - 3限目　講義(演習説明)(尾下or齊藤or斎藤or徳永or宮野or藤本)
  - 4～5限目　演習
- 3週目 画像認識(3)
  - 3～5限目　演習
- 計画書提出(3週目5限目まで)

プロジェクトⅠ　日程(2)

- プログラム提出(4週目の前日まで)
- 課題画像収集作業を4週目3限目までに終える。
  スキャン作業は4週目5限目までに終える。
- 4週目 識別精度
  - 3限目　講義(識別精度)(本田)
  - 4限目　講義(演習説明)(尾下or齊藤or斎藤or徳永or宮野or藤本)
  - 5限目　演習
- 5週目 自由演習
  - 3～5限目　演習
- 6週目 プレゼンテーション
  - 3～4限目　プレゼンテーション

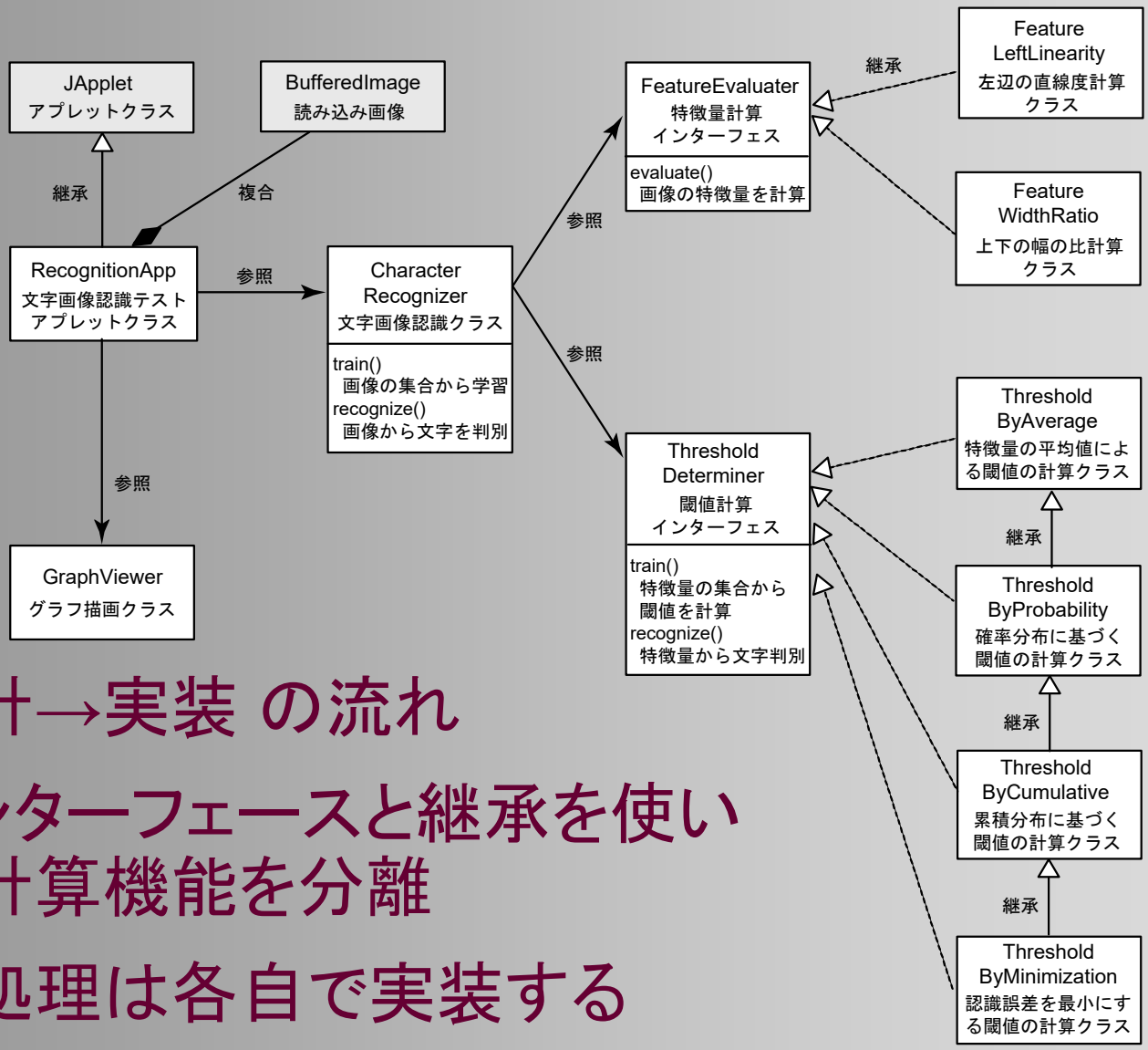
前回の内容

- 3限目 講義（佐藤）
  - 特徴量を使った画像の分類手法
- 4限目 講義（尾下or齊藤or斎藤or徳永or宮野or藤本）
  - 画像認識プログラムの設計の考え方
- 5限目 演習
  - 設計に従ったプログラムの枠組みの準備
  - 1つ目の特徴量の計算（左辺の直線度）
  - 1つ目の1次元の閾値計算（平均値による閾値）

今日以降の内容

- 3限目 講義（尾下or齊藤or斎藤or徳永or宮野or藤本）（60分程度）
  - １次元での閾値計算のプログラミング
  - ２次元での識別のための拡張
  - 各自の実験、プレゼンテーション
- 残りは演習（本日の残りと次回）
  - 本日の説明と資料に従ってプログラムを完成
    - なるべく今日中に資料のプログラムを完成
  - 課題画像の文字2種類（画像でも良い）を決めて、実験結果をプレゼンテーション

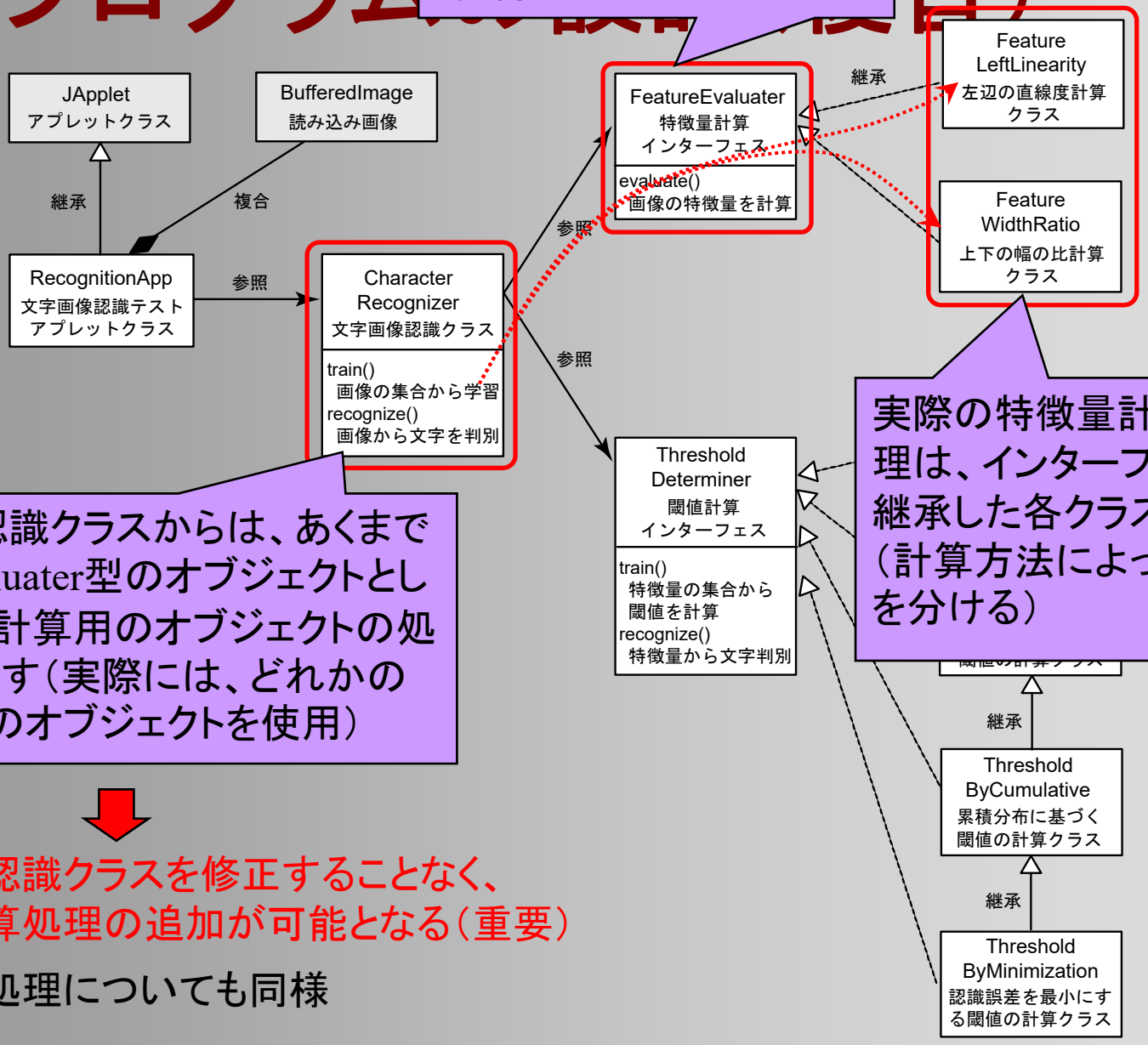
プログラムの設計（復習）



- 設計→実装 の流れ
- インターフェースと継承を使い
  各計算機能を分離
- 各処理は各自で実装する

プログラム設計（復習）

特徴量を計算する処理のインターフェースを定義（実際の処理はまだなし）



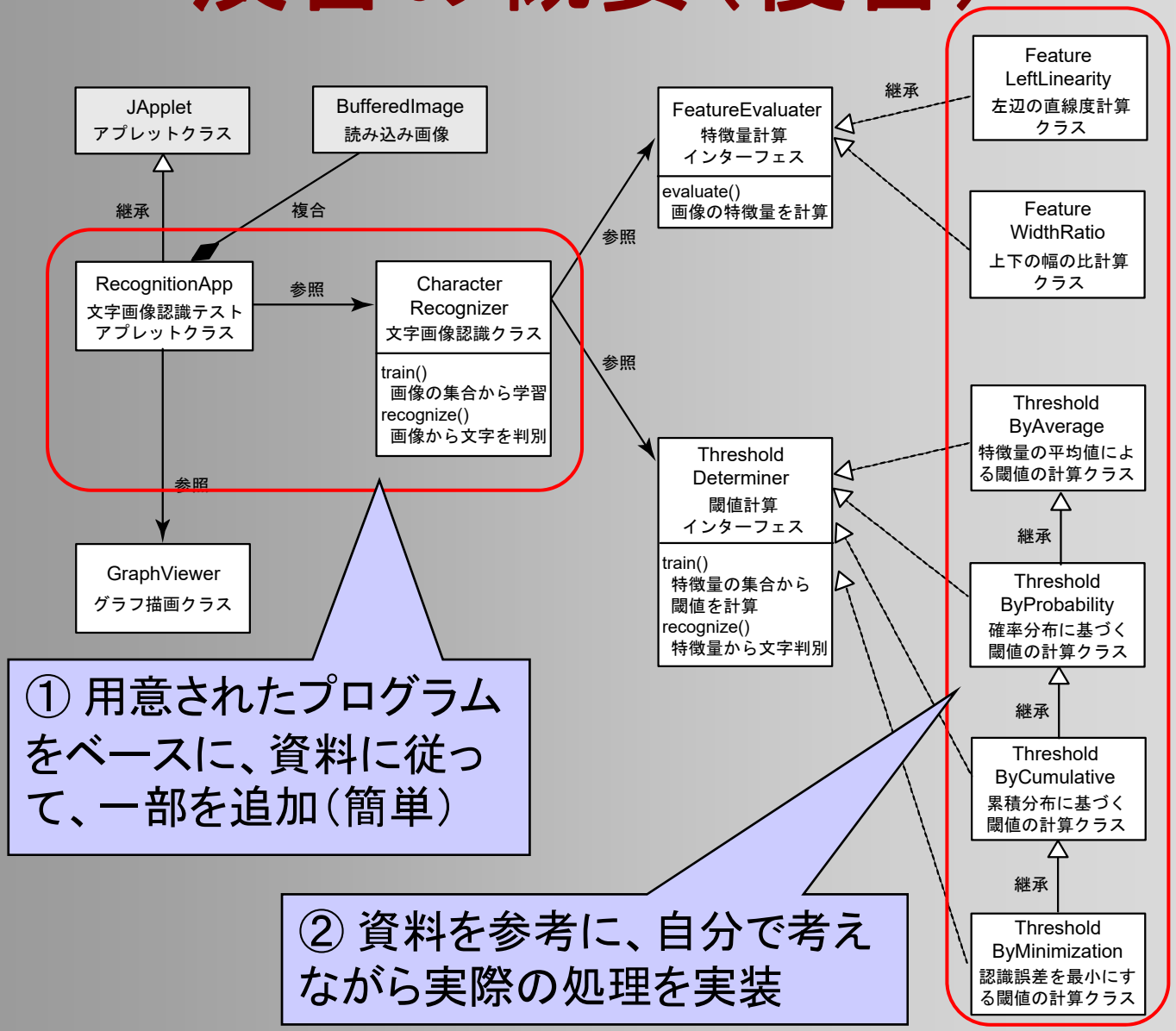
実際の特徴量計算の処理は、インターフェースを継承した各クラスで実装（計算方法によってクラスを分ける）

文字画像認識クラスからは、あくまでFeatureEvaluater型のオブジェクトとして、特徴量計算用のオブジェクトの処理を呼び出す（実際には、どれかのサブクラスのオブジェクトを使用）

文字画像認識クラスを修正することなく、特徴量計算処理の追加が可能となる（重要）

閾値計算処理についても同様

演習の概要(復習)


JApplet
アプレットクラス
BufferedImage
読み込み画像
継承
複合
RecognitionApp
文字画像認識テスト
アプレットクラス
参照
Character
Recognizer
文字画像認識クラス
train()
画像の集合から学習
recognize()
画像から文字を判別
参照
GraphViewer
グラフ描画クラス
FeatureEvaluater
特徴量計算
インターフェス
evaluate()
画像の特徴量を計算
参照
参照
Threshold
Determiner
閾値計算
インターフェス
train()
特徴量の集合から
閾値を計算
recognize()
特徴量から文字判別
継承
Feature
LeftLinearity
左辺の直線度計算
クラス
Feature
WidthRatio
上下の幅の比計算
クラス
Threshold
ByAverage
特徴量の平均値による閾値の計算クラス
継承
Threshold
ByProbability
確率分布に基づく
閾値の計算クラス
継承
Threshold
ByCumulative
累積分布に基づく
閾値の計算クラス
継承
Threshold
ByMinimization
認識誤差を最小にする閾値の計算クラス
① 用意されたプログラムをベースに、資料に従って、一部を追加(簡単)
② 資料を参考に、自分で考えながら実際の処理を実装

資料の内容(復習)

- 1～3章　プログラム全体の設計(1次元での識別)
- 4章　プログラム開発手順(1次元での識別)
- 5章　特徴量1の計算処理の作成方法
- 6章　閾値の計算処理の作成方法
- 7章　プログラム全体の設
- 8章　特徴量2の計算処理
- 9章　閾値の計算処理の作成方法
- 10章　プログラム開発手順(2次元での識別)
- 11章　プログラム開発手順(各自の画像の識別)

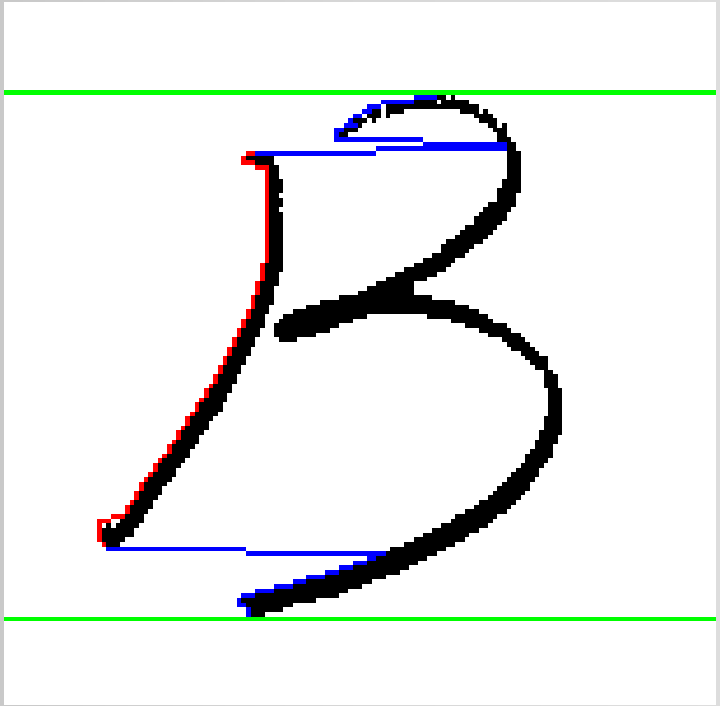
これらの章に書かれた手順に従って、作業を進める。
(適宜、他の章の内容を参照)

1つ目の特徴量の計算

1つ目の特徴量の計算

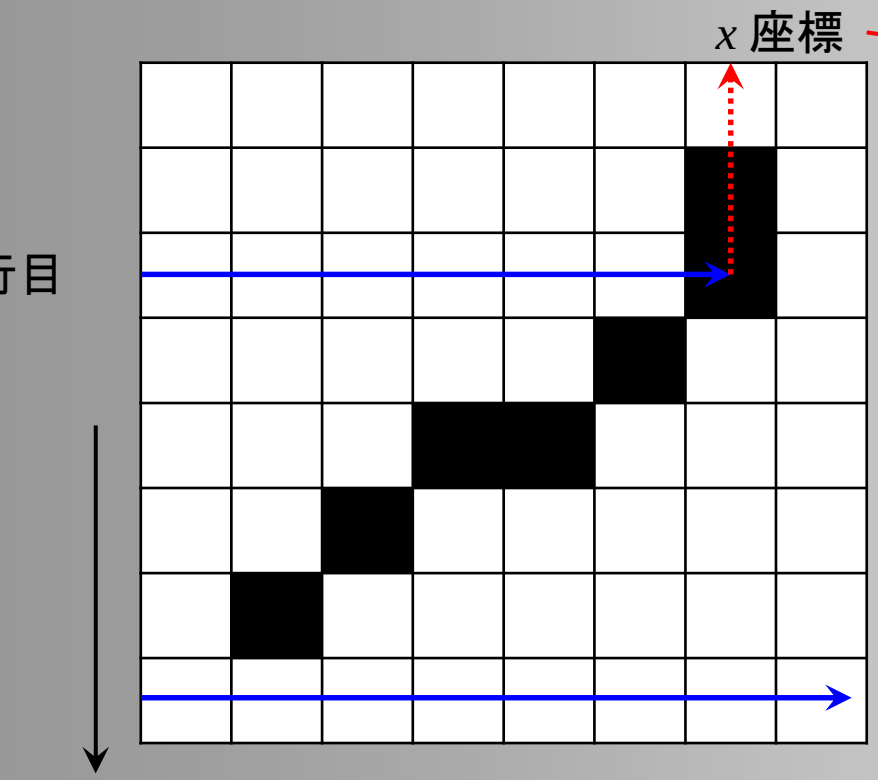
- 左辺の高さ ／ 左辺の長さ
  1. 各行ごとの左辺のX座標を探索して記録
     - 下図の青線＋赤線に相当
  2. 左辺の高さを計算
     - 図の緑線の y 座標の差
  3. 左辺の長さを計算
     - 左辺の各線分の和
  4. 特徴量の計算

  – 途切れた辺への対策



1. 左辺のX座標を探索して記録

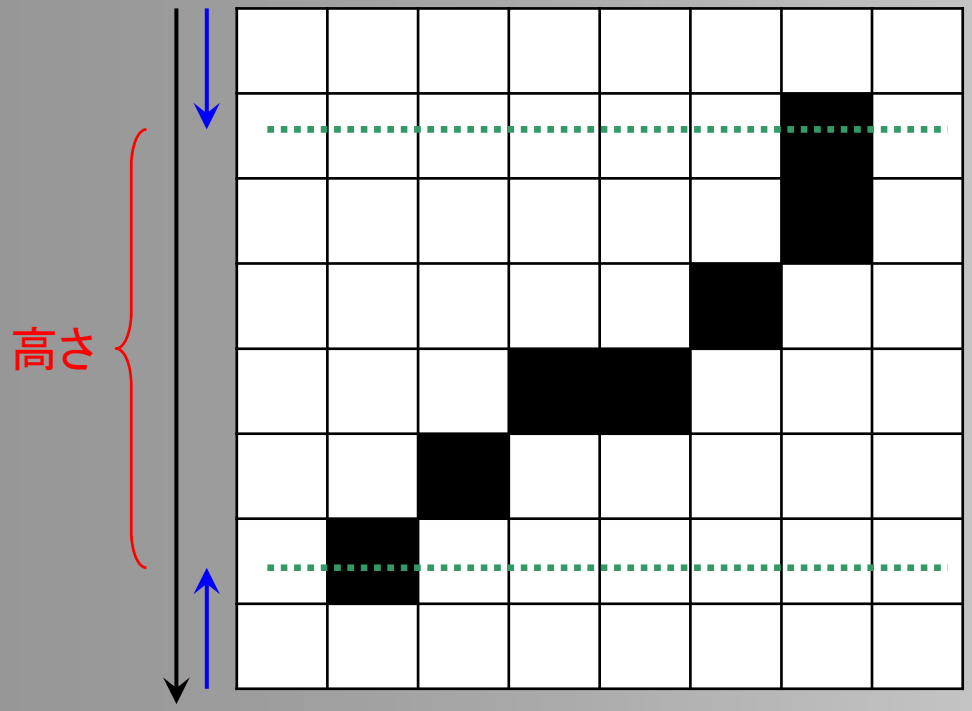
- 最初に黒ピクセルのある x 座標を記録



2. 左辺の高さを計算

- 最初と最後に黒ピクセルがある行の y 座標から高さを計算

height = 2つの y 座標の差から計算



left_x[0] = −1

left_x[1] = ・・・ → 最初に黒ピクセルがある行（y=1）

left_x[2] = ・・・

left_x[3] = ・・・

left_x[4] = ・・・

left_x[5] = ・・・

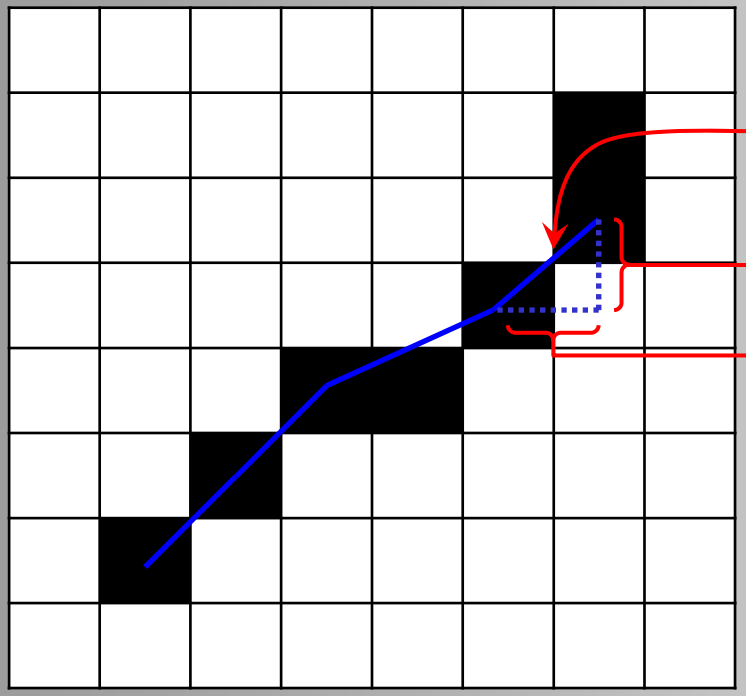
left_x[6] = ・・・ → 最後に黒ピクセルがある行（y=6）

left_x[7] = −1

※ 下から探索したとき、最初に黒ピクセルがある行

3. 辺の長さを計算

- 折れ線の長さの和により辺の長さを計算



i 行目

i+1 行目

三平方の定理により長さが求まる

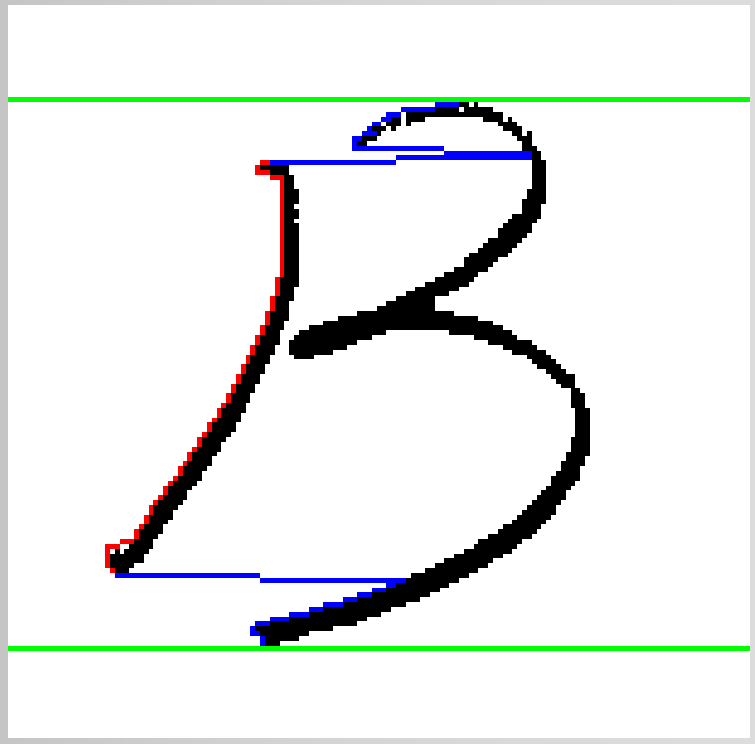
高さ 1

広さ $|\mathrm{left\_x}[i+1] - \mathrm{left\_x}[i]|$

各行ごとに加算していくことで、辺全体の長さが求まる

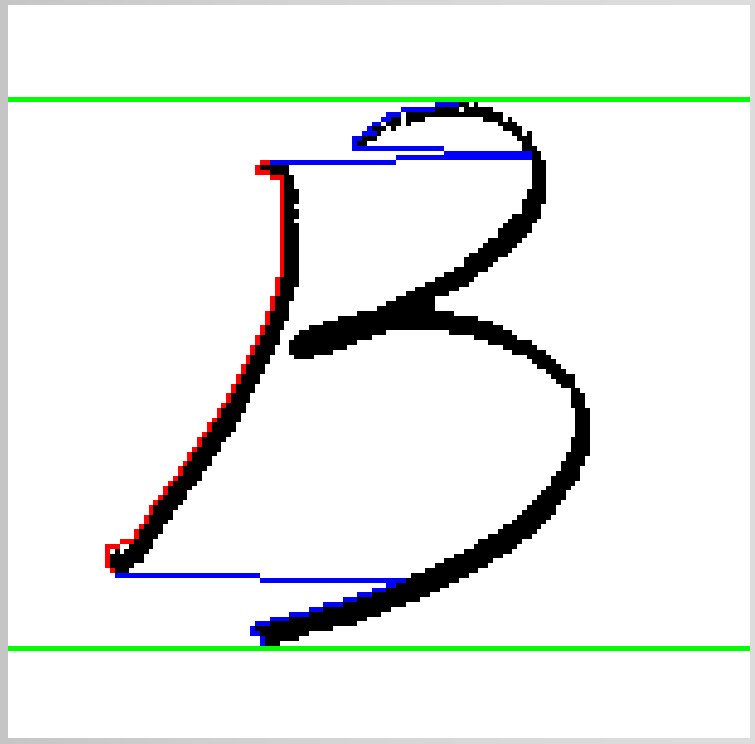
特徴量の計算結果の表示

- 特徴量計算の確認のために、抽出した情報を描画する機能を追加すると、デバッグがやりやすくなる
  - paintImageFeature()
    メソッドに処理を追加



途切れた辺への対策

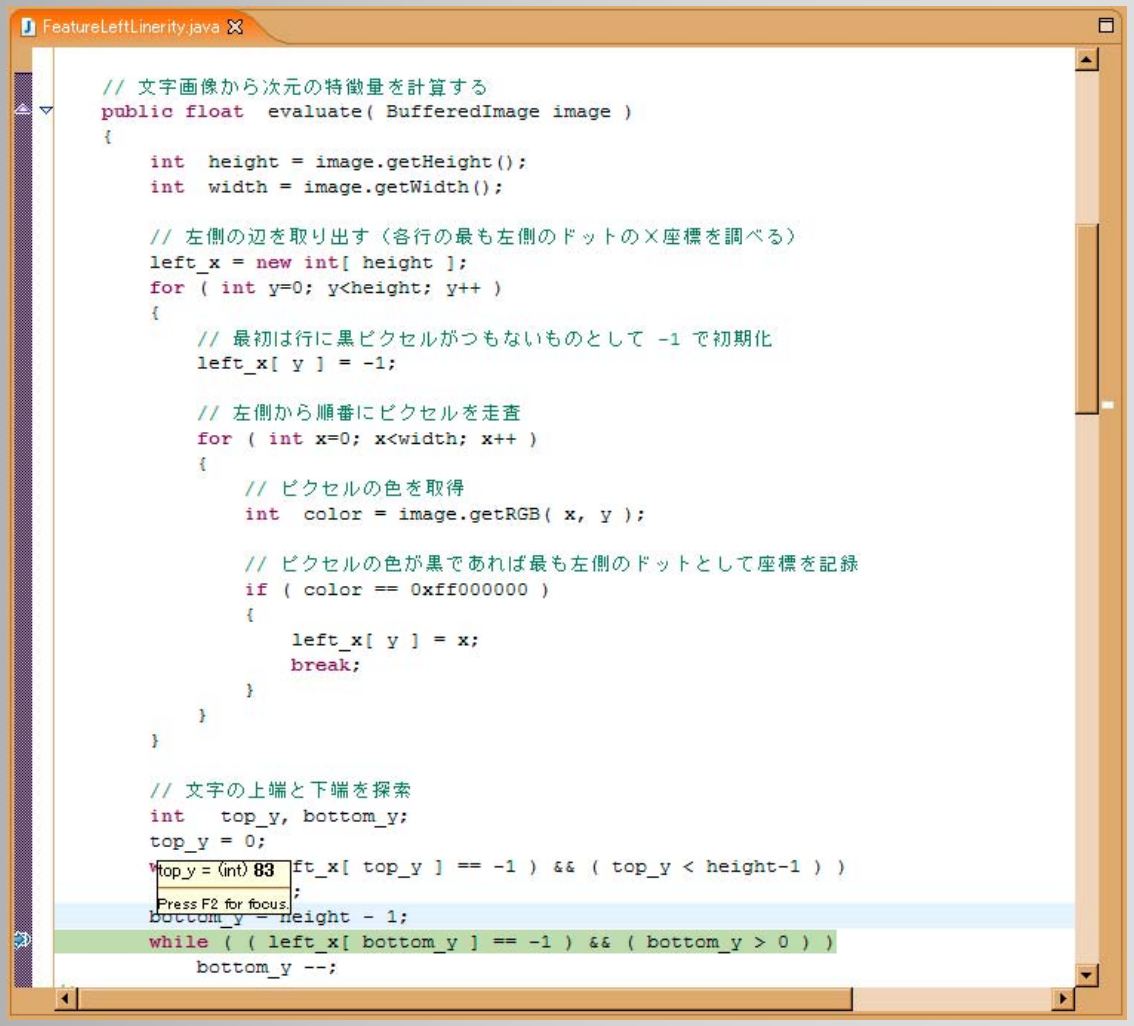
- いくつかの方法が考えられるので、各自工夫する
  - 左辺を複数の範囲に分け、最も長い範囲のみを抽出
  - あらかじめ左辺のX座標をフィルタリングして平滑化する
  - 単純に、一定以上のギャップは無視する



Eclipseを使ったデバッグ

Eclipseを使ったデバッグ

- プログラムをステップ実行可能
- 実行中のプログラムの変数の値が見れる

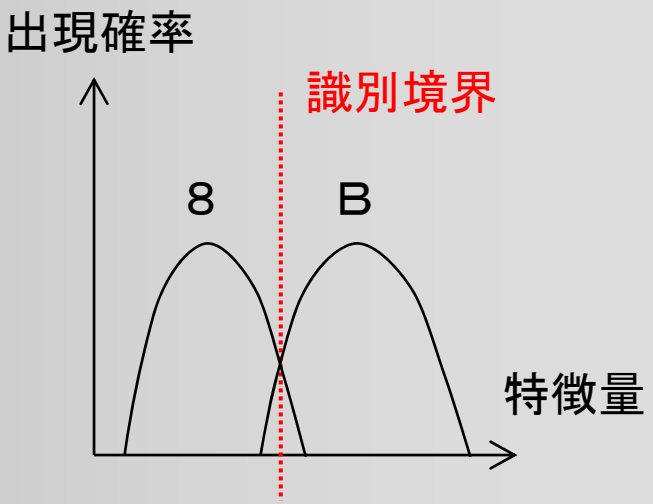


閾値の計算

閾値の計算

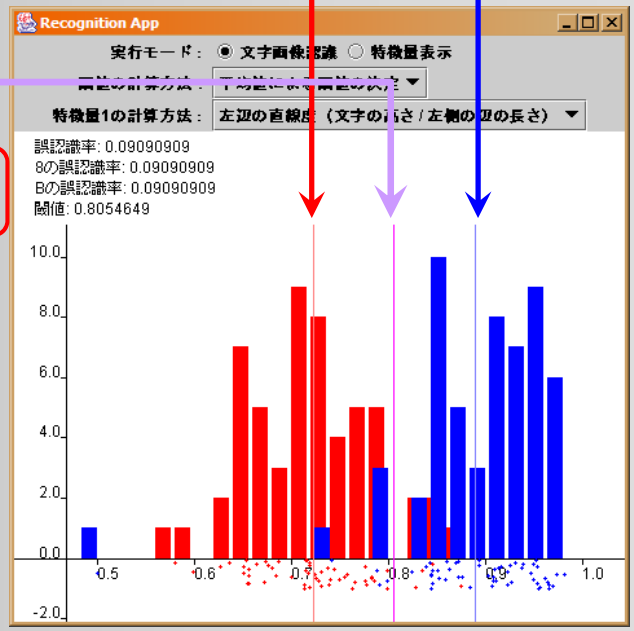
- 平均値に基づく閾値
  - ヒストグラムの計算の実装
- 出現確率が等しくなる閾値
- 誤認識率が等しくなる閾値
- 誤認識率が最小になる閾値

どのようにして、最適な閾値（識別境界）を定めるか？



平均値に基づく閾値の計算

- 2つのグループの特徴量の平均値を計算
  - float features0[] の平均 → average0
  - float features1[] の平均 → average1
- 2つの平均値の中間の値を計算
  - average0,average1→ threshold
- 各グループが閾値のどちら側にあるかを判定
  - average0 の方が小さい → is_first_smaller に真を代入

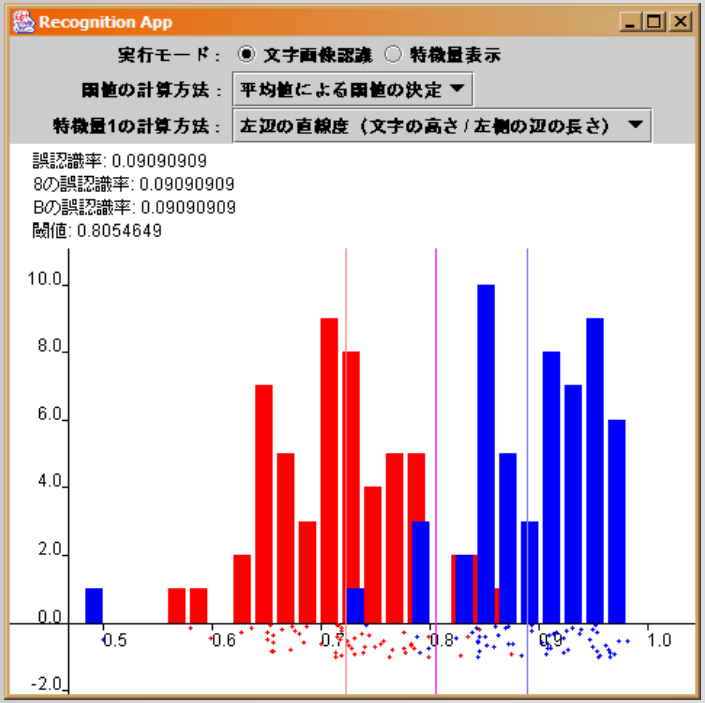


```
// 両グループの特徴量から閾値を決定する
public void   determine( float[] features0, float[] features1 )
{
        // 各グループの平均値を計算
        average0 = 0.0f; // 要実装
        average1 = 0.0f; // 要実装

        // ２つの平均値の中央値を計算
        threshold = 0.0f; // 要実装

        // 符号を計算
        is_first_smaller = true; // 要実装

        // 特徴量データを記録（グラフ描画用）
        this.features0 = features0;
        this.features1 = features1;
}
```

```
// 閾値をもとに特徴量から文字を判定する
public int   recognize( float feature )
{
        // グループ0の特徴量 < 閾値 < グループ1の特徴量
        if ( is_first_smaller )
        {
                if ( feature < threshold )
                        return   0;
                else
                        return   1;
        }
        // グループ1の特徴量 < 閾値 < グループ0の特徴量
        else
        {
                if ( feature < threshold )
                        return   1;
                else
                        return   0;
        }
}
```

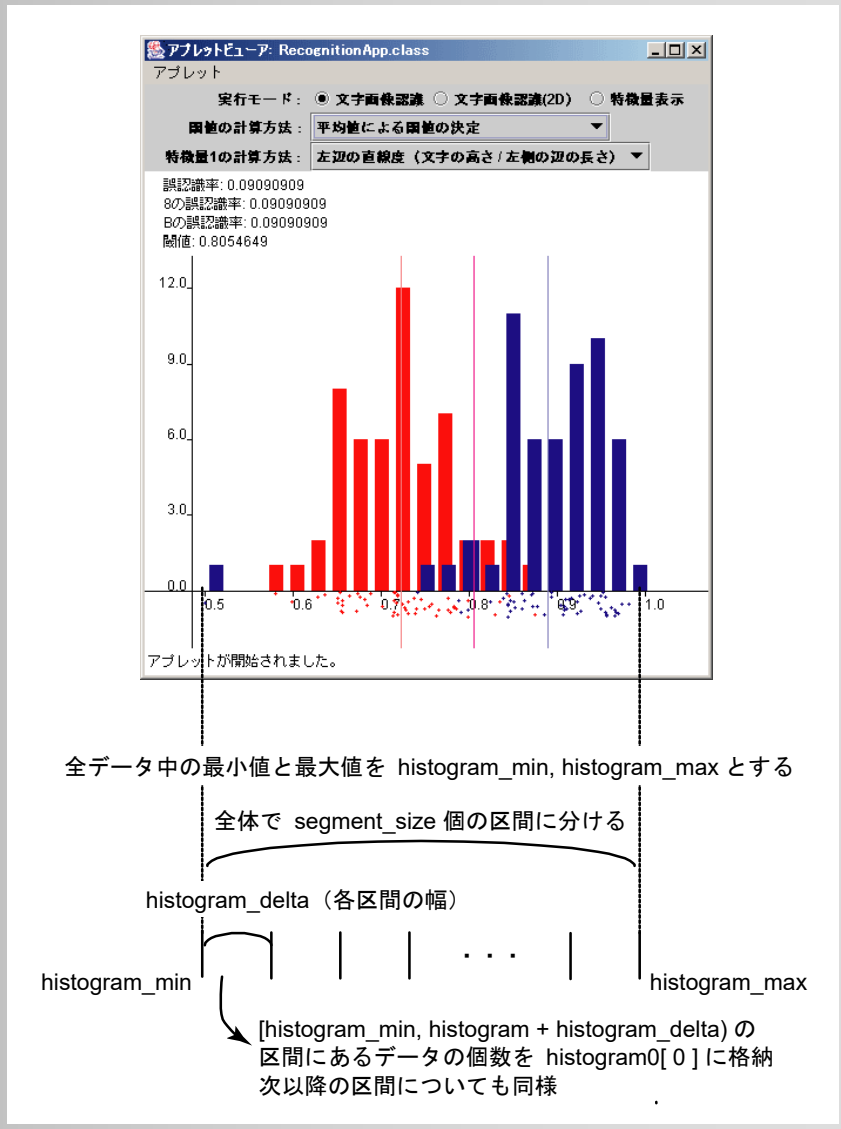
ヒストグラムの計算と描画

- 特徴量の値をいくつかの区分に分ける
- 各区分に含まれるデータの数をカウント
- 区分を自動的に決定
  - 区分の幅を指定する方法
    makeHistogramsBySize()
  - 区分の数を指定する方法
    makeHistogramsByWidth()



ヒストグラムの自動計算

- 区分の幅を指定
  - 省略
- 区分の数を指定
  - 最大･最小値を求める
  - 最大･最小値と区分数から、区分の幅を決定
  - 区分に従ってヒストグラムを計算

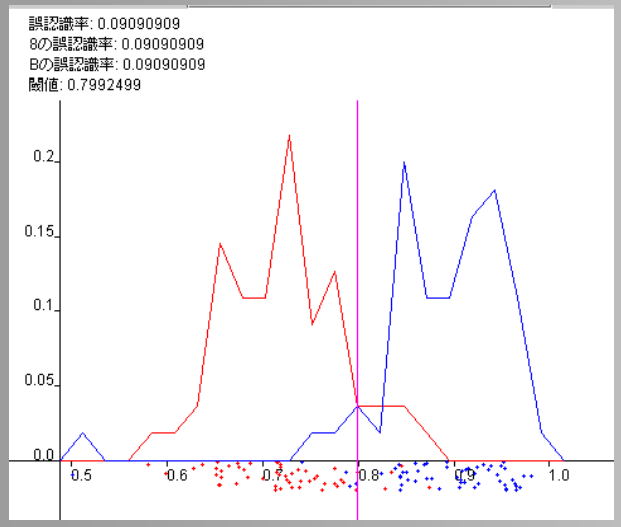


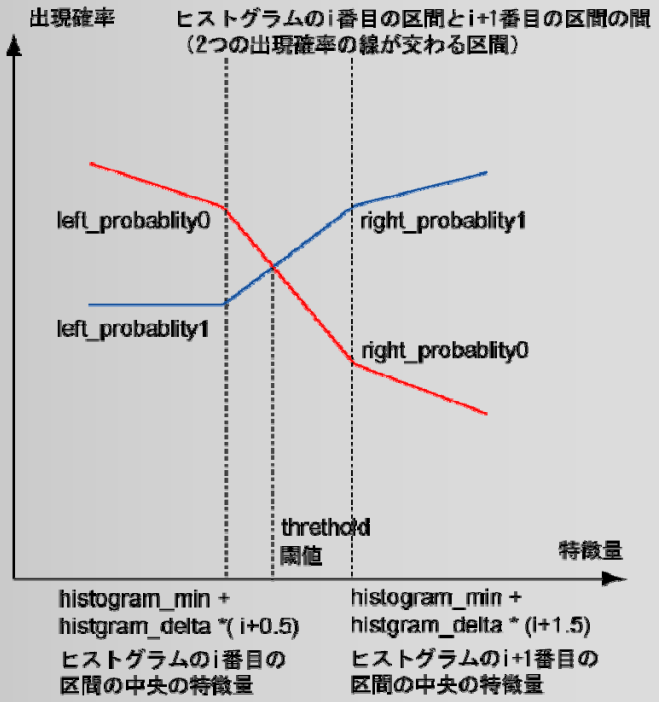
ヒストグラム計算時の注意

- 特徴量の分布によっては区分の調整が必要
  - 初期値では、20区分に分けるようになっている
  - 実験で特徴量計算を自作するときに注意！
  - ひとつでも大きくはずれた値があると、ヒストグラムが正しく作成されないことがある

出現率が等しくなる閾値

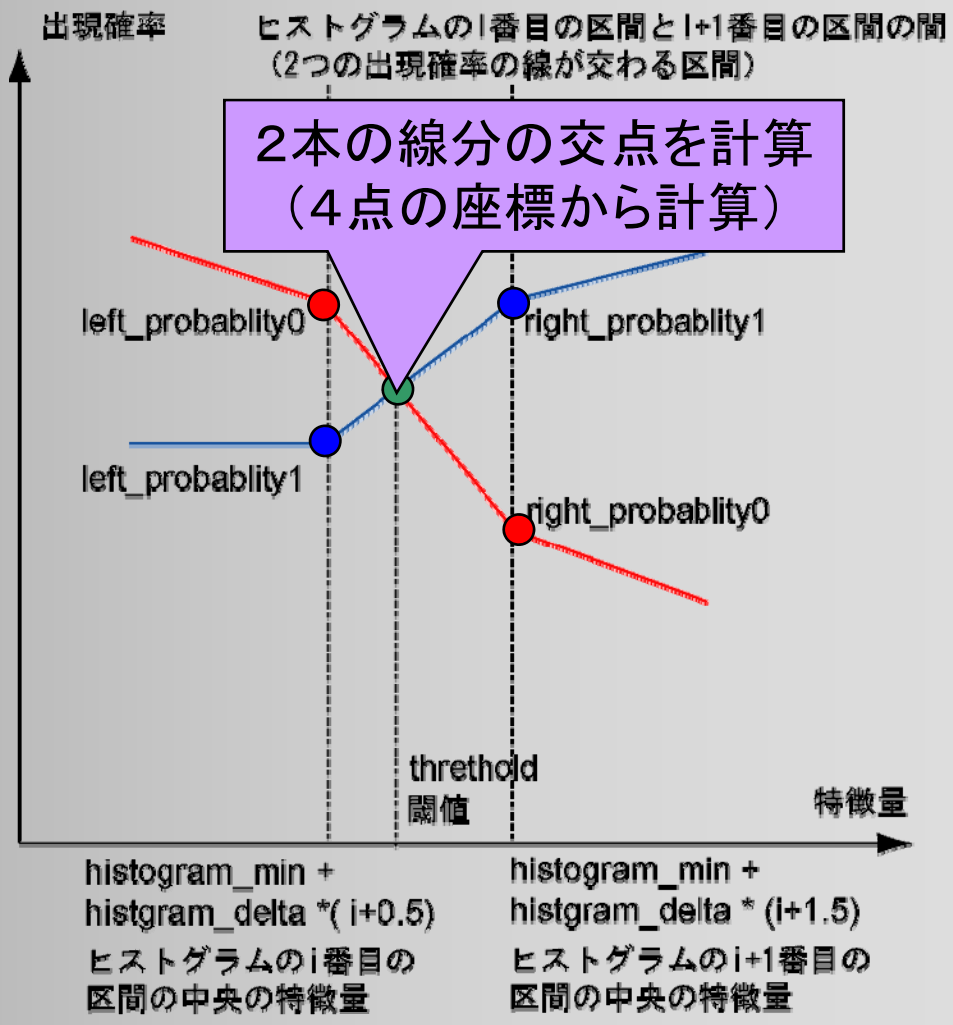
- 出現率
  - 各区間でデータが出現する確率
    - 区間でのデータ数／全データ数
  - 各隣接区間での出現率から、線分の交点を計算





出現率が等しくなる閾値

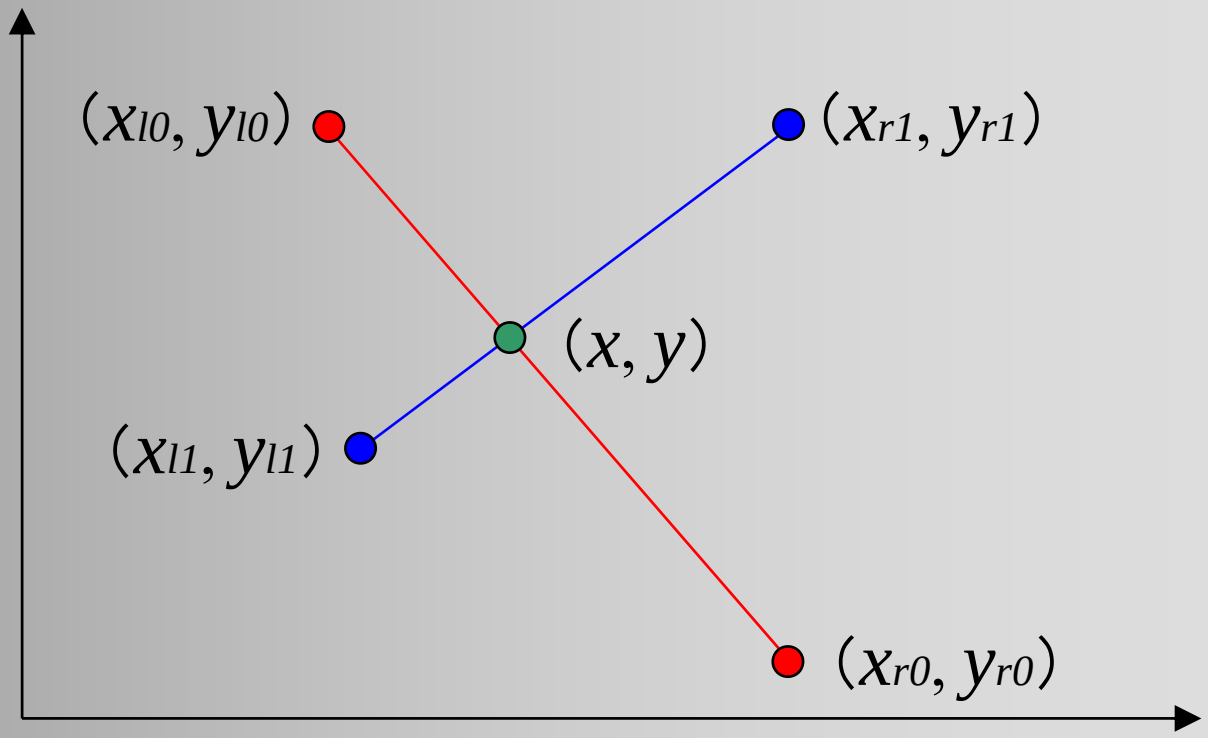
- 処理
  - 各隣接区間を調べ、出現確率の高低関係が逆転する区間を探索
  - 隣接区間内での線分の交点から、特徴量の閾値を計算
  - 交点が複数ある場合への対応



出現率が等しくなる閾値

- 2つの線分の端点から、交点を計算
  - まずは紙の上で、交点の x 座標を求める式を書いて、その式を計算するプログラムを作成すると良い

$$x = f(x_{l0}, y_{l0}, x_{r0}, y_{r0}, x_{l1}, y_{l1}, x_{r1}, y_{r1})$$



```
//
//  確率分布に基づく閾値の計算クラス
//
class   ThresholdByProbability extends ThresholdByAverage
{
        // 確率分布
        float[]   probability0;
        float[]   probability1;

        // 両グループの特徴量から閾値を決定する
        public void   determine( float[] features0, float[] features1 )
        {
                // 基底クラス（ThresholdByAverage）の計算処理を実行（初期値として使用）
                super.determine( features0, features1 );

                // ２つの特徴量の度数分布（ヒストグラム）を計算
                makeHistogramsBySize( default_histogram_size );

                // 累積度数分布から確率分布を計算
                makeProblility();

                // 各隣接区間（i番目の区間とi+1番目の区間の間の区間）ごとに、
                // 出現確率が等しくなる点があるかどうかを調べる
                for ( int seg_no=0; seg_no<histogram0.length-1; seg_no++ )
                {
                        // 区間の右端・左端の特徴量の値を計算する
                        float   feature_left, feature_right;
                        feature_left = histogram_min_f + histogram_delta_f * ( seg_no + 0.5f );
                        feature_right = histogram_min_f + histogram_delta_f * ( seg_no + 1.5f );

                        // 区間の右端・左端での各グループの出現確率を取得する
                        float   prob0_left, prob1_left, prob0_right, prob1_right;
                        prob0_left = probability0[ seg_no ];
                        prob1_left = probability1[ seg_no ];
                        prob0_right = probability0[ seg_no + 1 ];
                        prob1_right = probability1[ seg_no + 1 ];

                        // 右端・左端で出現確率の高いグループが異なっている、
                        // もしくはどちらかで出現確率が等しければ、
                        // その区間で必ず出現確率が等しい点が存在する
                        if ( /* 要実装 */ )
                        {
                                // 区間内の出現確率が等しい点を計算する
                                // 要実装
                                threshold = ...;
                        }
                }
        }

        // 累積度数分布から確率分布を計算
        protected void   makeProblility()
        {
                probability0 = new float[ histogram0.length ];
                for ( int i=0; i<probability0.length; i++ )
                        probability0[ i ] = (float) histogram0[ i ] / features0.length;

                probability1 = new float[ histogram1.length ];
                for ( int i=0; i<probability0.length; i++ )
                        probability1[ i ] = (float) histogram1[ i ] / features1.length;
        }
}
```

```
// 各隣接区間（i番目の区間とi+1番目の区間の間の区間）ごとに、
// 出現確率が等しくなる点があるかどうかを調べる
for ( int seg_no=0; seg_no<histogram0.length-1; seg_no++ )
{
        // 区間の右端・左端の特徴量の値を計算する
        float   feature_left, feature_right;
        feature_left = histogram_min_f + histogram_delta_f * ( seg_no + 0.5f );
        feature_right = histogram_min_f + histogram_delta_f * ( seg_no + 1.5f );

        // 区間の右端・左端での各グループの出現確率を取得する
        float   prob0_left, prob1_left, prob0_right, prob1_right;
        prob0_left = probability0[ seg_no ];
        prob1_left = probability1[ seg_no ];
        prob0_right = probability0[ seg_no + 1 ];
        prob1_right = probability1[ seg_no + 1 ];

        // 右端・左端で出現確率の高いグループが異なっている、
        // もしくはどちらかで出現確率が等しければ、
        // その区間で必ず出現確率が等しい点が存在する
        if ( /* 要実装 */ )
        {
                // 区間内の出現確率が等しい点を計算する
                // 要実装
                threshold = ...;
        }
}
```

```
// 各隣接区間（i番目の区間とi+1番目の区間の間の区間）ごとに、
// 出現確率が等しくなる点があるかどうかを調べる
for ( int seg_no=0; seg_no<histogram0.length-1; seg_no++ )
{
        // 区間の右端・左端の特徴量の値を計算する
        float   feature_left, feature_right;
        feature_left = histogram_min_f + histogram_delta_f * ( seg_no + 0.5f );
        feature_right = histogram_min_f + histogram_delta_f * ( seg_no + 1.5f );

        // 区間の右端・左端での各グループの出
        float   prob0_left, prob1_left, prob0_righ
        prob0_left = probability0[ seg_no ];
        prob1_left = probability1[ seg_no ];
        prob0_right = probability0[ seg_no + 1 ];
        prob1_right = probability1[ seg_no +

        // 右端・左端で出現確率の
        // もしくはどちらか
        // その区間で必ず出現確率が等しい点が
        if ( /* 要実装 */ )
        {
                // 区間内の出現確率が等しい点
                // 要実装
                threshold = ...;
        }
}
```

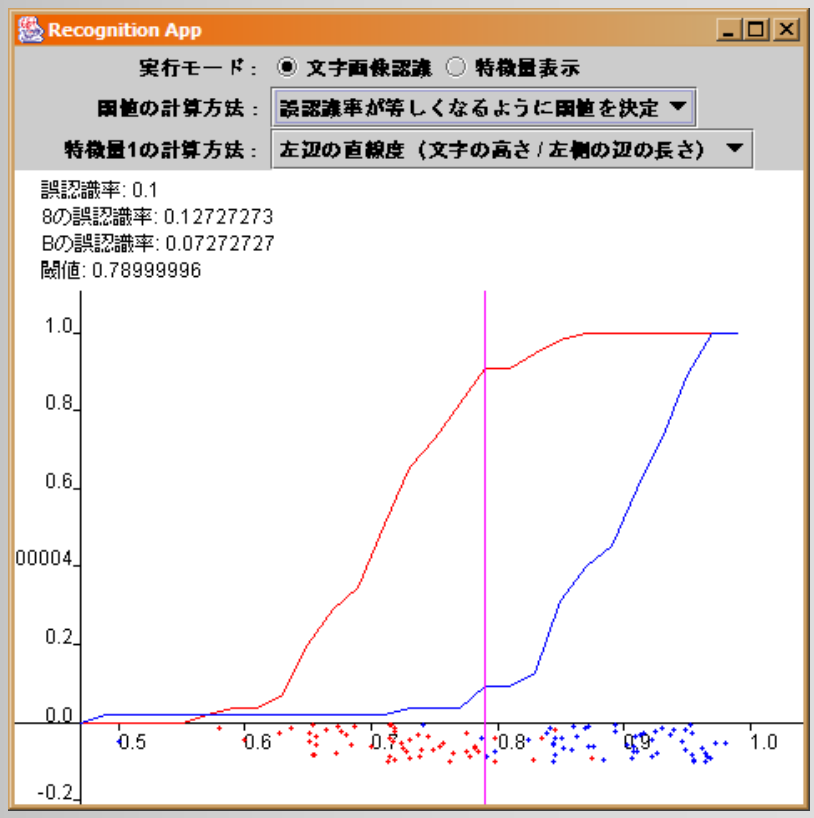
① 適切な条件を記述
区間の右端と左端で、2つのグループの出現確率の大きさの関係が入れ替わるかどうか（＝区間内で折れ線が交差）

② 適切な計算を記述
2つの出現確率の線分が交差する点での、特徴量の値（閾値）を計算

誤認識率が等しくなる閾値

- 誤認識率
  - その値を閾値にしたとき、どの程度の割合のデータを誤認識するか
- 誤認識率が等しくなる
  - 左右にある誤りデータの出現率が等しい
  - 出現確率の累計の和が 1 になる値

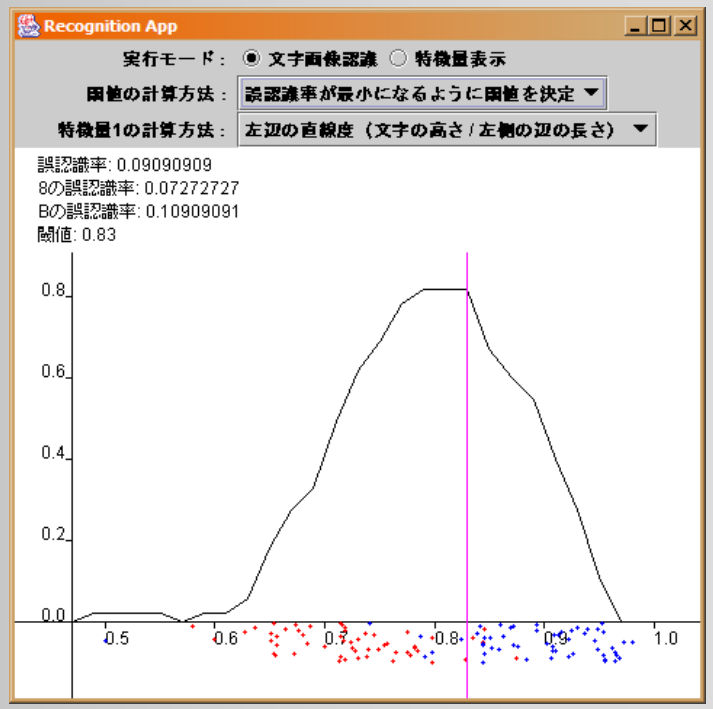
出現確率の累計のグラフ



誤認識率が最小になる閾値

- 2つの特徴量の出現確率の累積の差（正しく認識されるデータの割合）が最大になる値

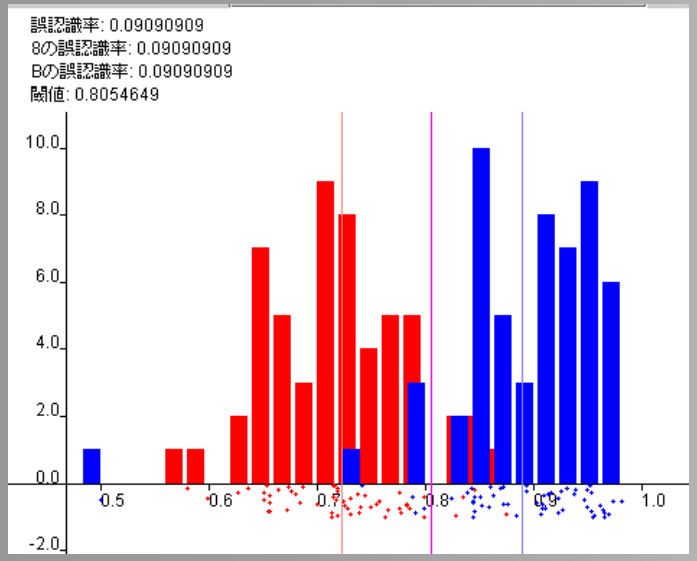
出現確率の累計の差のグラフ

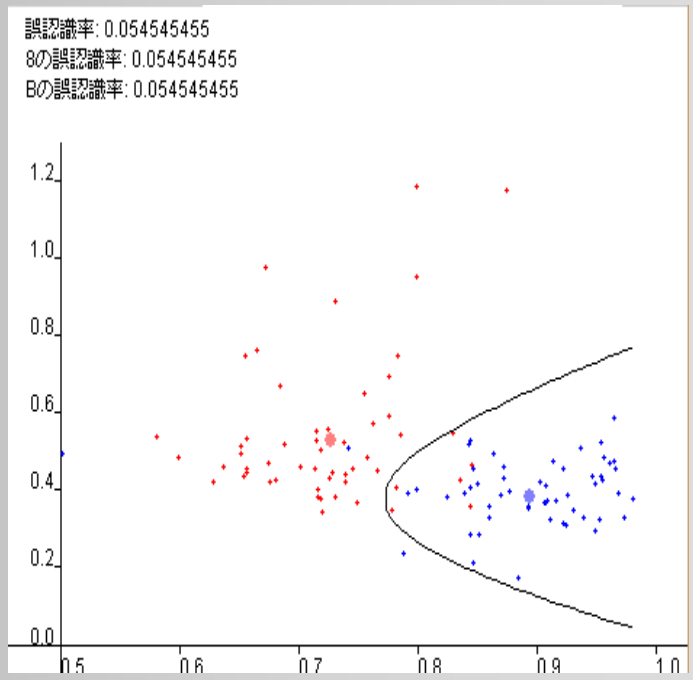


閾値の計算（まとめ）

- 平均値に基づく閾値
  - ヒストグラムの計算の実装
- 出現確率が等しくなる閾値
- 誤認識率が等しくなる閾値
- 誤認識率が最小になる閾値

2次元の特徴量を使った認識





2次元への拡張

- 2つの特徴量を使った画像認識に拡張
  - 特徴量の計算はそのままで問題ない
    - 2つ目の特徴量の計算を追加する
  - 閾値の計算に、2つの特徴量を入力できるように変更する必要がある
    - 資料28ページのインターフェース定義を参照
- 具体的な内容は資料を参照

```
//
//    １次元の特徴量の閾値計算クラスのインターフェース
//
interface   ThresholdDeterminer
{
        // 閾値の決定方法の名前を返す
        public String   getThresholdName();

        // 両グループの特徴量から閾値を決定
        public void   determine( float[] features0, float[] features1 );

        // 与えられた特徴量からどちらの文字かを判定
        public int   recognize( float feature );

        // 閾値を返す
        public float   getThreshold();

        // 特徴空間のデータをグラフに描画（グラフオブジェクトに図形データを設定）
        public void   drawGraph( GraphViewer gv );
}


//
//        ２次元の特徴量の閾値計算クラスのインタ
//
interface   ThresholdDeterminer2D
{
        // 閾値の決定方法の名前を返す
        public String   getThresholdName();

        // 両グループの特徴量から閾値を決定
        public void   determine( float[][] features0, float[][] features1 );

        // 与えられた特徴量からどちらの文字かを判定
        public int   recognize( float[] feature );

        // 閾値を返す
        public float   getThreshold( float f0 );

        // 特徴空間のデータをグラフに描画（グラフオブジェクトに図形データを設定）
        public void   drawGraph( GraphViewer gv );
}
```

2次元の特徴量の配列を受け取るよう引数を変更

feature0

x_1	x_2	x_3	x_4	x_5	…
y_1	y_2	y_3	y_4	y_5	…

feature1

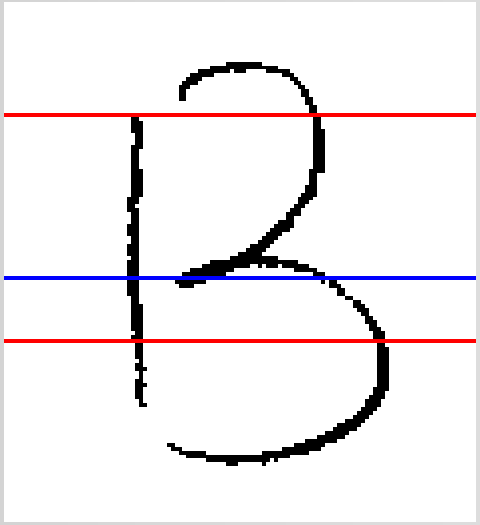
x_1	x_2	x_3	x_4	x_5	…
y_1	y_2	y_3	y_4	y_5	…

```
//
//   文字画像認識クラス
//
class   CharacterRecognizer
{
        // 特徴量の評価用オブジェクト
        protected FeatureEvaluater      feature_evaluater;

        // 閾値の決定用オブジェクト
        protected ThresholdDeterminer   threshold_determiner;

        // 学習に使用した画像の特徴量
        protected float   features0[];
        protected float   features1[];

        // 与えられた２つのグループの画像データを判別するような特徴量の閾値を計算
        public void   train( BufferedImage[] images0, BufferedImage[] images1 );

        // 学習結果に基づいて与えられた画像を判別（判別した画像の種類 0 or 1 を返す）
        public int   recognizeCharacter( BufferedImage image );

        // 特徴空間のデータを描画（グラフオブジェクトにデータを設定）
        public void   drawGraph( GraphViewer gv );
}
```

```
//
//   文字画像認識クラス（２次元の特徴量に対応した拡張版）
//
class   CharacterRecognizer2D extends CharacterRecognizer
{
        //  １次元・２次元のどちらの特徴量を認識に使用するかの設定
        protected int     dimension = 1;

        // 特徴量の評価用オブジェクト
        protected FeatureEvaluater      feature_evaluater2;

        // 閾値の決定用オブジェクト（２次元の特徴量への対応版）
        protected ThresholdDeterminer2D   threshold_determiner_2d;

        // 学習に使用した画像の特徴量（２次元の特徴量）
        protected float   features2d0[][];
        protected float   features2d1[][];


        // 与えられた２つのグループの画像データを判別するような特徴量の閾値を計算
        public void   train( BufferedImage[] images0, BufferedImage[] images1 );

        // 学習結果に基づいて与えられた画像を判別（判別した画像の種類 0 or 1 を返す）
        public int   recognizeCharacter( BufferedImage image );

        // 特徴空間のデータを描画（グラフオブジェクトにデータを設定）
        public void   drawGraph( GraphViewer gv );
}
```

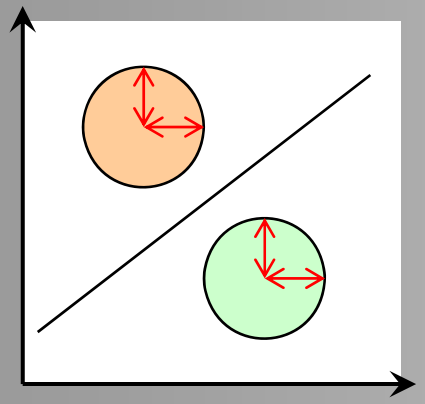
2つ目の特徴量の計算

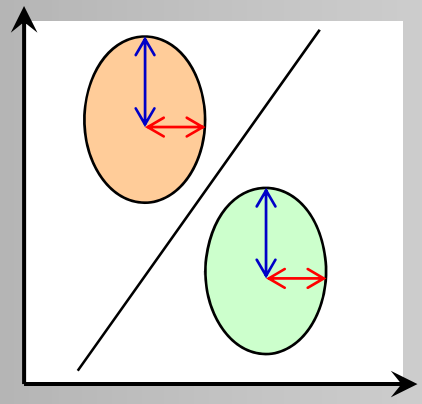
- 上部の幅 ／ 下部の幅
  - 各ラインごとの文字の幅を調べる。
  - 画像の中央部で文字の幅が極小になるラインを見つけ、文字を上部と下部に分る。（青線）
  - 上部で文字の幅が極大になるライン、下部で文字の幅が極大になるラインを見つける。（赤線）
  - 上部の幅 / 下部の幅 を計算する。
- サンプルプログラムを参考

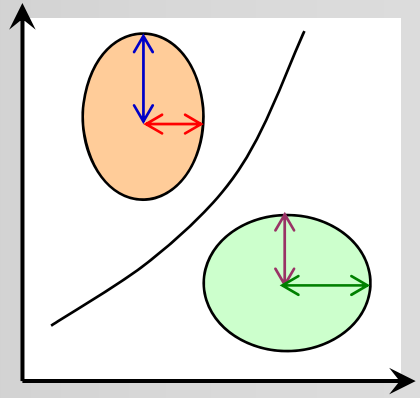


2次元の特徴量を使った認識

- 同一の正規分布に従うと仮定
（Threshold2DByGaussian1）
- 特徴量ごとに分散の異なる正規分布に従うと仮定
（Threshold2DByGaussian2）
- 一般の正規分布に従うと仮定
（Threshold2DByGaussian3）



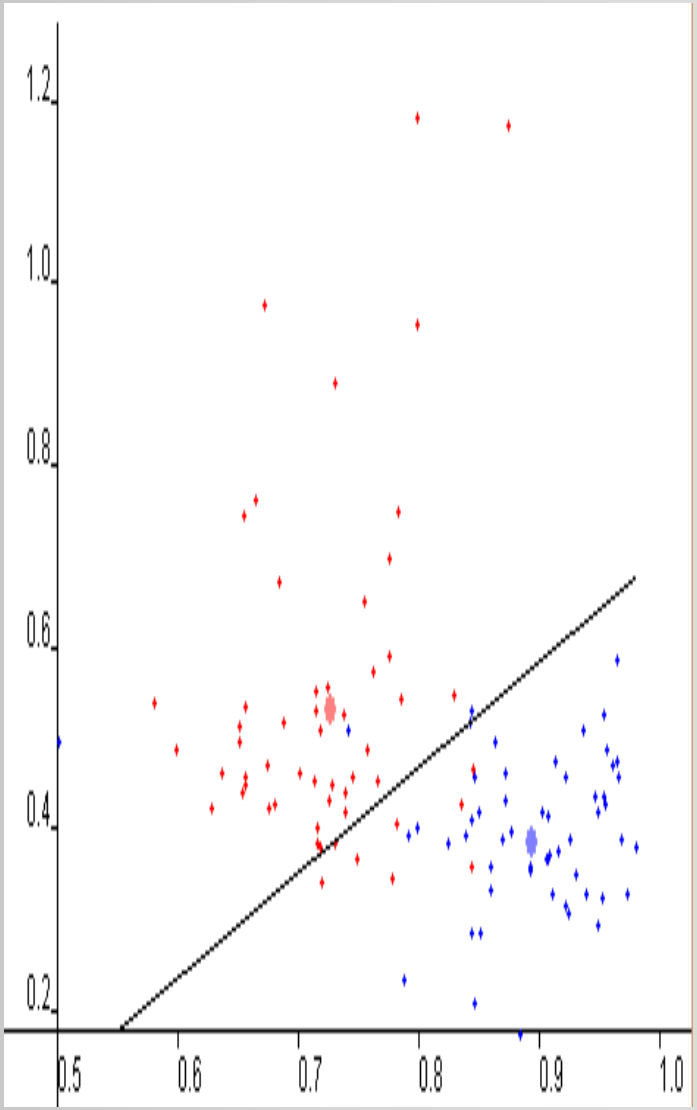




同一の正規分布に従うと仮定

- X軸・Y軸は、各特徴量
- 各グループの各特徴量の分散が同じと仮定

- 境界線の方程式
  - 各グループの特徴量の平均座標の中央を通り、２点を結ぶ線分に直交
  - 各グループが境界線のどちら側かの情報も必要

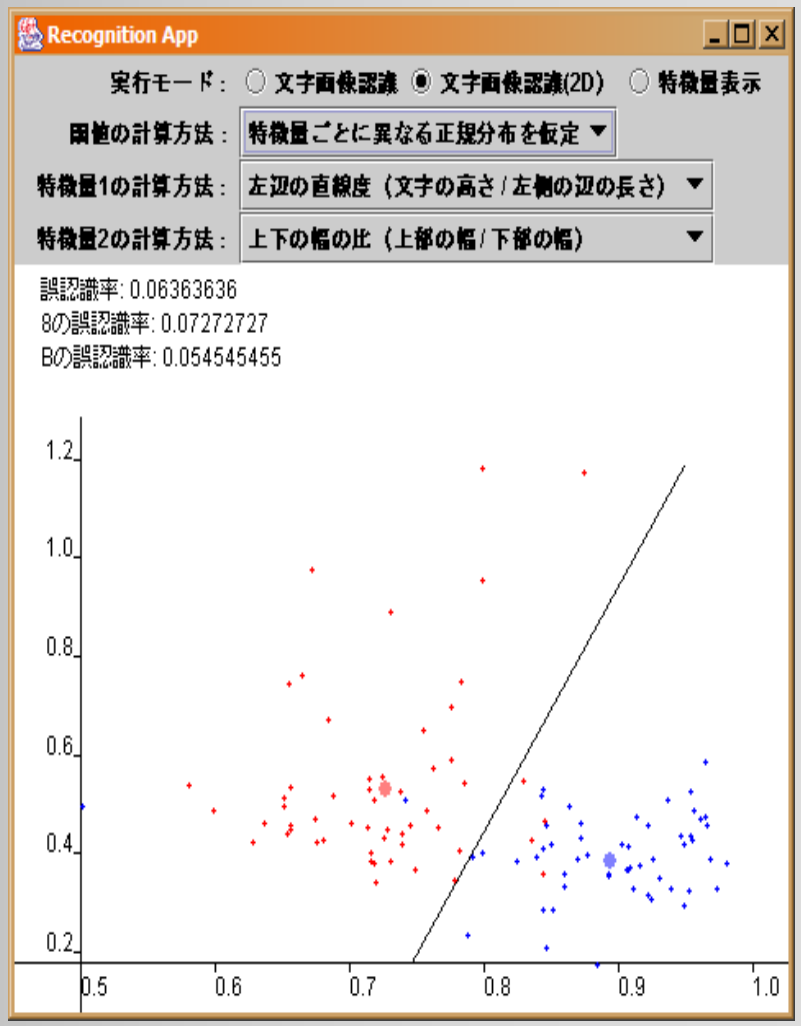


```
//
//   同一の正規分布に基づく２次元の特徴量の閾値の計算クラス
//
class   Threshold2DByGaussian1 implements ThresholdDeterminer2D
{
        // 特徴量の平均値
        protected float   mean0x, mean0y;
        protected float   mean1x, mean1y;

        // 閾値（境界）の方程式
        protected float   border_ox, border_oy; // 境界線の中心点
        protected float   border_dy; // 境界線の傾き

        // 閾値の符号（グループ０の方が特徴量が閾値よりも小さければ真）
        protected boolean   is_first_smaller;

        // 両グループの特徴量から閾値を決定
        public void   determine( float[][] features0, float[][] features1 )
        {
                // 特徴量の平均値（mean0x, mean0y, mean1x, mean1y）を計算
                // 要実装

                // 境界の方程式を計算（２つの平均値からの距離が等しくなる直線を境界とする）
                // border_ox, border_oy, border_dy を求める
                // 要実装

                // 境界の符号を判定
                // （グループ０が特徴量が境界よりも下にあれば is_first_smaller を真にする）
                // 要実装

        }
```

特徴量ごとに分散の異なる正規分布に従うと仮定

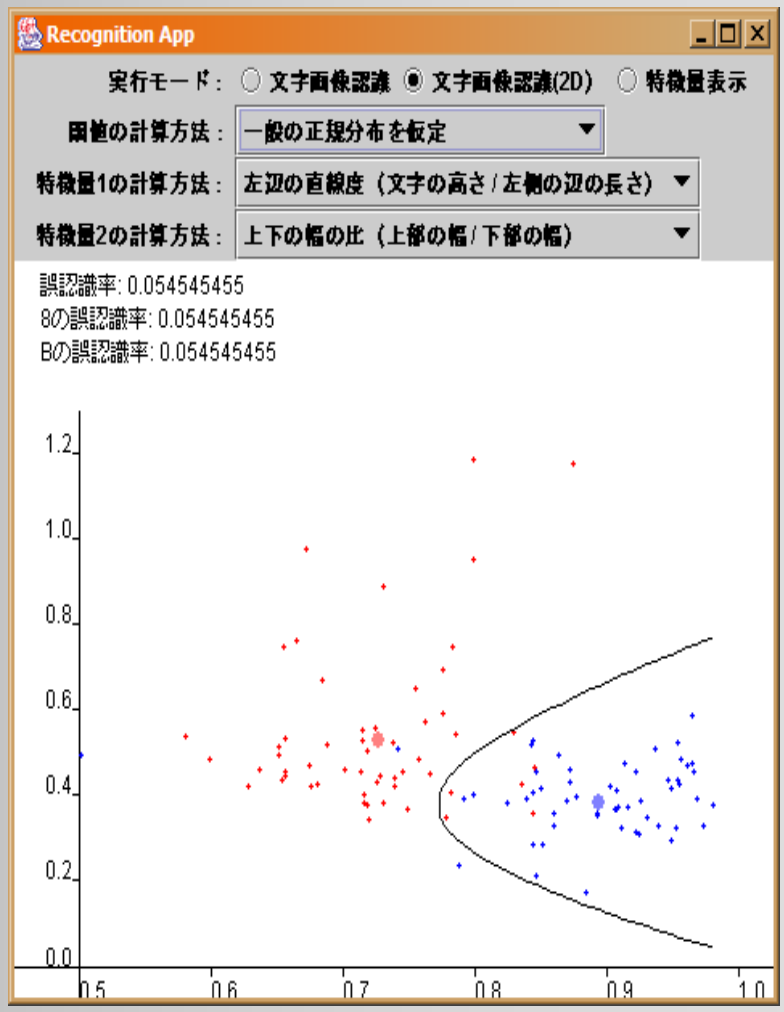
- 2つの特徴量で分散が異なると仮定
- 分散の大きさに応じて境界線の傾きを修正

$$\text{傾き} \times \frac{x\text{軸（特徴量0）方向の分散}}{y\text{軸（特徴量1）方向の分散}}$$



一般の正規分布に従うと仮定

- 2つのグループ・2つの特徴量のそれぞれで分散が異なると仮定
- 境界線が曲線になる
- マハラノビス距離による判定
- サンプルプログラムを参照



計画書提出

計画書提出

- 3週目5限目までに、演習担当教員に提出する。
  - 課題画像の文字2種類
  - 作成予定の特徴量3個

  を計画書に記入する。
    - 計画書は1週目で配布済み。
    - 紛失した場合は、様式はMoodleからダウンロードして利用する。

中間プログラム提出

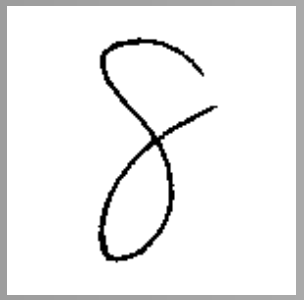
中間プログラム提出

- 4週目の前日までに、少なくともここまでで説明したプログラムの内容までは完成させ、提出する
  - 4週目に行う講義・演習は、ここまでの内容が終っていることを前提としているため
- Moodleから提出
  - 提出締め切り 木曜日 18:00（厳守）
  - 進捗度に応じて評価、未提出は大幅減点

各自の実験

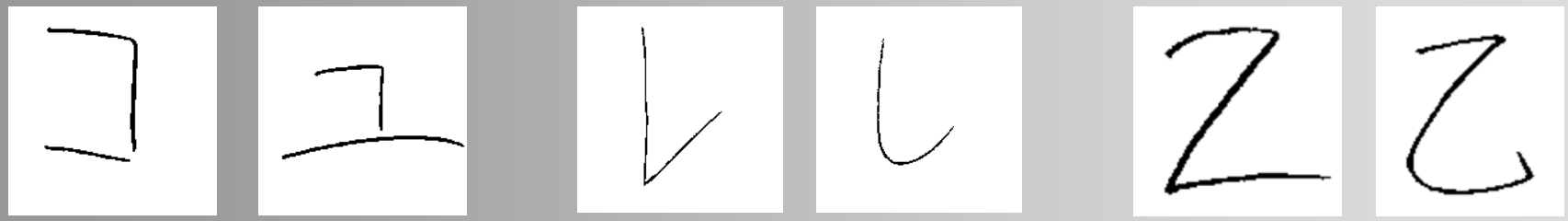
プロジェクト課題(1)(確認)

- 準備演習(全員共通)
  - 2種類の画像を識別するプログラムを開発
  - 最初は、こちらで用意した画像データ(8とBの画像データ)を使用する



プロジェクト課題(2)(確認)

- 識別する画像を自分達で決める



※ 過去の受講者の課題の例

  - グループの決めた2種類の課題画像を識別できるような特徴量を計算するようプログラムを改良
  - 実際にサンプルを採取(友人等に書いてもらう)して、実験を行い、結果を考察
  - プレゼンテーション

グループでの演習の流れ（1）

- 演習資料で説明されているプログラムを、２人で共同で作成する。
  - 作業を分担して、それぞれの端末で作成して、後でプログラムを統合する。
  - ２人で相談しながら、どちらかの端末で作成してもよい。

グループでの演習の流れ(2)

- 複数グループで取り組む課題画像を相談して決める。
  - 課題画像の収集も複数グループで共同で取り組む。
  - {A1，A2，A3}で相談して課題画像を決める。
  - {B1，B2，B3}で相談して課題画像を決める。
  - {C1，C2}で相談して課題画像を決める。

グループでの演習の流れ(3)

- グループ(二人一組のグループ)で取り組む特徴量(3種類)を相談して決める。
- グループごとに課題画像と作成予定の特徴量(3種類)を計画書に記入し、3週目16:10までに演習担当教員に提出する。
  - 計画書を提出し、画像や特徴量が妥当かどうかを相談する。
    - 簡単すぎ or 難しすぎないか？
    - うまく識別ができそうか？　など

グループでの演習の流れ(4)

- 複数グループ(A、B、C)で課題画像のサンプル画像を収集する。
  - 収集作業は4週目3限目までに終える。
  - スキャン作業は4週目5限目までに終える。

サンプル画像の収集手順（1）

1. 講義HPにある画像スキャン用フレームを利用して、友人などに書いてもらう。
   - 十分な数のサンプルを使用する（各文字100サンプル以上）。
   - 一人につき最大10サンプルとする。
   - なるべく多くの人に協力してもらう。
2. スキャナを利用してスキャン画像を作成する。

サンプル画像の収集手順(2)

3. 講義HPにあるツール(Image Clip)を用いて、各サンプル画像が1枚の画像ファイルになるようにする。
4. 収集した画像をMoodleにアップロードする。
   - 各グループで収集したファイルをzip形式で圧縮する。
   - その際、バージョン情報をファイル名に含める。
     - 例:「A1_v1.zip」
   - 「実験用サンプル画像置き場」ページにアップロードする。

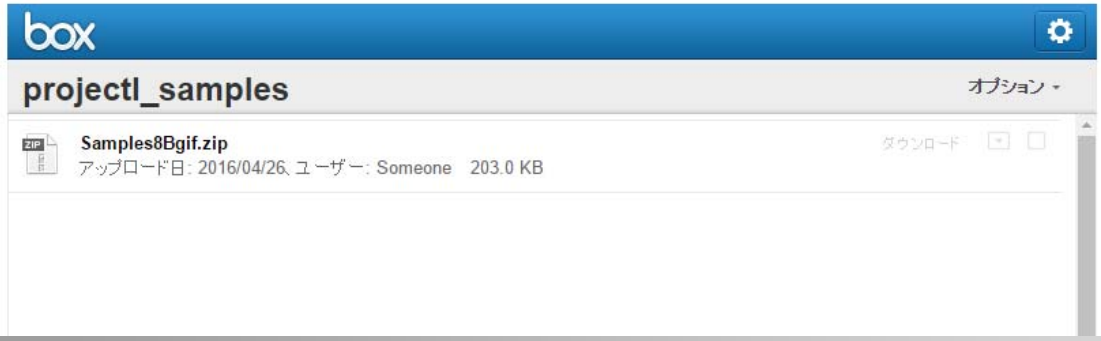
サンプル画像の収集手順(3)

実験用サンプル画像置き場

ファイルのアップロード

収集した全てのサンプル画像を「グループ名_vバージョン番号.zip」(例: A3_v1.zip)のファイル名で圧縮してアップロードする。
(一度提出したファイルは削除・上書きできないので、サンプル画像に変更が生じた場合は場合は、バージョン番号を加算した新しいファイルをアップロードすること。)

ファイルのダウンロード



サンプル画像の収集手順(4)

5. A、B、Cのそれぞれのグループで協力し、アップロードされたファイルをダウンロードして、データ(ファイル名など)を整理する。
   - 各グループがアップロードされたファイルのファイル名を連番にする。
   - 講義HPにあるツール(Namery)などを用いると便利。

サンプル画像の収集手順(5)

6. 整理されたファイルをzip形式で圧縮し、再びMoodleにアップロードする。
   - 例:「A.zip」
7. 整理されたファイルをダウンロードして、各グループ(二人一組のグループ)の実験に利用する。

グループでの演習の流れ(5)

- 特徴量(3種類)のプログラムを作成して実験する。
  - 3種類の特徴量から3通りの組み合わせ全てを実験し、結果を考察する。
  - 特徴量3個(A、B、C)を用いて2次元における3通り({A, B}、{A, C}、{B, C})の識別実験を行い、結果を考察する。
- プレゼンテーション用のスライドを作成する。