Table of Contents

Breast Cancer Wisconsin (Diagnostic) Data Set

https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

Attribute Information:

1. ID number
2. Diagnosis (M = malignant:-1, B = benign:1) M:悪性，B:良性
3. 3-32

Ten real-valued features are computed for each cell nucleus:

- 半径radius (mean of distances from center to points on the perimeter)
- テクスチャtexture (standard deviation of gray-scale values)
- 境界の長さperimeter
- 面積area
- なめらかさsmoothness (local variation in radius lengths)
- コンパクトさcompactness (perimeter^2 / area - 1.0)
- くぼみ度合いconcavity (severity of concave portions of the contour)
- くぼみの数concave points (number of concave portions of the contour)
- 対称性symmetry
- フラクタル次元fractal dimension ("coastline approximation" - 1)

のそれぞれのmean, stderr, worst数値を保持している．

http://people.idsia.ch/~juergen/deeplearningwinsMICCAIgrandchallenge.html

分類器

用意された訓練(training)データには， $\boldsymbol{A}$に上に記した特徴量が， $\boldsymbol{b}$に悪性(-1)か良性(1)かを 示す数値が入っている．

与えられた特徴ベクトル$\boldsymbol{y}$に対し， 細胞組織が悪性か良性かを分類する関数$C(\boldsymbol{y})$を選び出すプログラムを作成しよう．

仮説クラス

分類器は可能な分類器の集合(**仮説クラス**)から選ばれる．この場合，仮説クラスとは特徴ベクトルの空間$\mathbb{R}^D$から$\mathbb{R}$への線形関数$h(\cdot)$である．すると分類器は次のような関数として定義される．

$$C(\boldsymbol{y}) = \begin{cases} +1 & \text{when} \quad h(\boldsymbol{y}) \geq 0 \\ -1 & \text{when} \quad h(\boldsymbol{y}) < 0 \end{cases}$$

各線形関数$h : \mathbb{R}^D \rightarrow \mathbb{R}$に対して， 次のような$D$ベクトル$\boldsymbol{w}$が存在する．

$$h(\boldsymbol{y}) = \boldsymbol{w} \cdot \boldsymbol{y}$$

したがって，そのような線形関数を選ぶことは，結局Dベクトル$\boldsymbol{w}$を 選ぶことに等しい．特に，$\boldsymbol{w}$を選ぶことは，仮説クラスhを 選ぶことと等価なので，$\boldsymbol{w}$を**仮説ベクトル**と呼ぶ．

問題を単純化すると，分類器を単なるベクトルとみなして，データとの掛け算で予測がつきます．本来は予測を-1,1とかに投影しないといけないんですが，単純化のためにそのままの値を用います．問題はどうやってこの仮説ベクトル$\boldsymbol{w}$の各要素の値を決定するか？ですよね．

最急降下法

損失関数に

$$L(w) = \sum_{i=1}^{n} (A_i \cdot \boldsymbol{w} - b_i)^2$$

を選ぶと，ベクトル$\boldsymbol{w}$の j偏微分は，

$$\begin{aligned} \frac{\partial L}{\partial w_j} &= \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (A_i \cdot w - b_i)^2 \\ &= \sum_{i=1}^{n} 2(A_i \cdot w - b_i) A_{ij} \end{aligned}$$

となる． ここで，A_{ij}はA_iのj番目の要素を意味する． この偏微分$\frac{\partial L}{\partial w_j}$を $\boldsymbol{w}_j$の勾配(slope)として，$L(w)$の極小値(local minimum)を求める．

このような探索方法を最急降下法(steepest descent method)と呼ぶ．

print_w

出てきた$\boldsymbol{w}$のj要素をきれいに表示する関数を用意しておきます．

In [ ]:
```
def print_w(w):
```

```
    params = ["radius", "texture","perimeter","area",
              "smoothness","compactness","concavity","concave points",
              "symmetry","fractal dimension"]
    print("   (params)      :     ",end="")
    print("   (mean)     (stderr)      (worst)")
    for i, param in enumerate(params):
        print("%18s:" %param, end="")
        for j in range(3):
            print("%13.9f" % w[i*3+j], end="")
        print()
```

データの読み込みと初期化

In [ ]:
```
import numpy as np
tmp = np.fromfile('./train_A.data', np.float64, -1, " ")
A = tmp.reshape(300,30)
tmp = np.fromfile('./train_b.data', np.float64, -1, " ")
b = tmp.reshape(300,1)
w = np.zeros(30).reshape(30,1)
for i in range(30):
    w[i] = 0
```

In [ ]:
```
A[0]
```

Out[ ]:
```
array([1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
       3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
       8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
       3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
       1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01])
```

In [ ]:
```
b[0]
```

Out[ ]:
```
array([-1.])
```

最急降下法によるw探索(steepest descent)

In [ ]:
```
loop, sigma = 300, 3.0*10**(-9)
for i in range(loop):
  dLw = A.dot(w)-b
  w = w - (dLw.transpose().dot(A)).transpose()*sigma

print_w(w)
```

```
   (params)      :     (mean)     (stderr)     (worst)
           radius:  0.000426997  0.000741817  0.002548876
          texture:  0.001687946  0.000004707  0.000000127
        perimeter: -0.000003968 -0.000002078  0.000008954
             area:  0.000003595  0.000002569  0.000070324
       smoothness:  0.000001139 -0.000881778  0.000000430
      compactness:  0.000000441  0.000000723  0.000000267
        concavity:  0.000001200  0.000000191  0.000411499
   concave points:  0.000921972  0.002395138 -0.001932789
         symmetry:  0.000005930 -0.000003750 -0.000008147
fractal dimension: -0.000002341  0.000011565  0.000003523
```


```
(params)      :        (mean)      (stderr)       (worst)
           radius:  0.000426997  0.000741817  0.002548876
          texture:  0.001687946  0.000004707  0.000000127
        perimeter: -0.000003968 -0.000002078  0.000008954
             area:  0.000003595  0.000002569  0.000070324
       smoothness:  0.000001139 -0.000881778  0.000000430
      compactness:  0.000000441  0.000000723  0.000000267
        concavity:  0.000001200  0.000000191  0.000411499
   concave points:  0.000921972  0.002395138 -0.001932789
         symmetry:  0.000005930 -0.000003750 -0.000008147
fractal dimension: -0.000002341  0.000011565  0.000003523
```


結果

In [ ]:
```
def show_accuracy(mA, vb, vw):
  # M:悪性(-1),B:良性(1)

  correct,safe_error,critical_error=0,0,0
  predict = mA.dot(vw)
  n = vb.size
  for i in range(n):
    if predict[i]*vb[i]>0:
      correct += 1
    elif (predict[i]<0 and vb[i]>0): # 良性なのに悪性と予測:再検査
      safe_error += 1
    elif (predict[i]>0 and vb[i]<0): # 悪性なのに良性と予測:見落とし
      critical_error += 1
  print("      correct: %4d/%4d" % (correct,n))
  print("   safe error: %4d" % safe_error)
  print("critical error: %4d" % critical_error)
```

In [ ]:
```
show_accuracy(A, b, w)
```

```
      correct:  274/ 300
   safe error:    5
critical error:   21
```

In [ ]:
```
tmp = np.fromfile('./validate_A.data', np.float64, -1, " ")
A = tmp.reshape(260,30)
tmp = np.fromfile('./validate_b.data', np.float64, -1, " ")
b = tmp.reshape(260,1)

show_accuracy(A, b, w)
```

```
      correct:  240/ 260
   safe error:   10
critical error:   10
```

QR decomposition

QR分解を使うとより簡単に最小値を求めることができる．行列Aは正方行列でないので，逆行列をもとめることができない．しかし，その場合でも$||A.w-b||^2$を最小にするwを求めることができる．

QR分解によって，$n \times m$行列は

$$A = QR$$

と分解される．ここで，Qは$n \times m$行列，Rは$m \times m$の正方行列で，逆行列を求めることができる．

$||Aw - b||$が最小となるのはQRを使って,

$$
\begin{gathered}
Q.R.w = b \\
R.w = Q^t.b \\
R^{-1}.R.w = R^{-1}.Q^t.b
\end{gathered}
$$

となりそう.

In [ ]:
```
import numpy as np

tmp = np.fromfile('./train_A.data', np.float64, -1, " ")
A = tmp.reshape(300,30)
tmp = np.fromfile('./train_b.data', np.float64, -1, " ")
b = tmp.reshape(300,1)

q, r = np.linalg.qr(A)
```

In [ ]:
```
ww = np.linalg.inv(r).dot(np.transpose(q).dot(b))
```

In [ ]:
```
q.shape
```

Out[ ]: (300, 30)

In [ ]:
```
print(r[0,0:5])
```

```
[-2.57579883e+02 -3.32324268e+02 -1.68607899e+03 -1.29450676e+04
 -1.65446346e+00]
```

In [ ]:
```
show_accuracy(A, b, ww)
```

```
     correct:  286/ 300
  safe error:    1
critical error:   13
```

In [ ]:
```
print_w(ww)
```

```
   (params)     :      (mean)     (stderr)      (worst)
          radius:  0.869921844 -0.024313948 -0.062679561
         texture: -0.003274619 -8.790300861  1.747147500
       perimeter: -0.202849407 -6.506451098  5.061760446
            area: 49.167541566 -0.956591421 -0.082052658
      smoothness: -0.007943157  0.004976908-27.841944367
     compactness:  3.301527110  4.985959134-16.318886295
       concavity: 10.316289081-21.332232171 -0.408605816
  concave points: -0.003345722 -0.000677873  0.002510735
        symmetry:  4.531369718  0.590110016 -0.719368704
fractal dimension: -2.158965299 -3.803467225-12.298417038
```


```
   (params)     :      (mean)     (stderr)      (worst)
          radius:  0.869921844 -0.024313948 -0.062679561
         texture: -0.003274619 -8.790300861  1.747147500
       perimeter: -0.202849407 -6.506451098  5.061760446
            area: 49.167541566 -0.956591421 -0.082052658
      smoothness: -0.007943157  0.004976908-27.841944367
     compactness:  3.301527110  4.985959134-16.318886295
       concavity: 10.316289081-21.332232171 -0.408605816
  concave points: -0.003345722 -0.000677873  0.002510735
        symmetry:  4.531369718  0.590110016 -0.719368704
fractal dimension: -2.158965299 -3.803467225-12.298417038
```


In [ ]:
```
tmp = np.fromfile('./validate_A.data', np.float64, -1, " ")
A = tmp.reshape(260,30)
tmp = np.fromfile('./validate_b.data', np.float64, -1, " ")
b = tmp.reshape(260,1)

show_accuracy(A, b, ww)
```

```
     correct:  252/ 260
  safe error:    6
critical error:   2
```