4.1 フィッシャーの線形判別

平成 28 年 9 月 11 日

概 要

PRMLの「4.1.4 フィッシャーの線形判別」についての実装と考察.

目 次

1 クラス平均の分離度最大化

1.1 問題設定

まず簡単に, クラス平均のの分離度を最大化するような分類規則を導く.

1.2 アルゴリズム

2 つのクラスの平均は

$$\mathbf{m}_1 = \frac{1}{N_1}\sum_{x\in C_1}\mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2}\sum_{x\in C_2}\mathbf{x}_n \quad (4.21)$$

で, $\mathbf{w}$ 上に射影された際のクラス分離度は

$$m_2 - m_1 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1) \quad (4.22)$$

これを最大化するには

$$\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$$

とすればよい.

1.3 コード

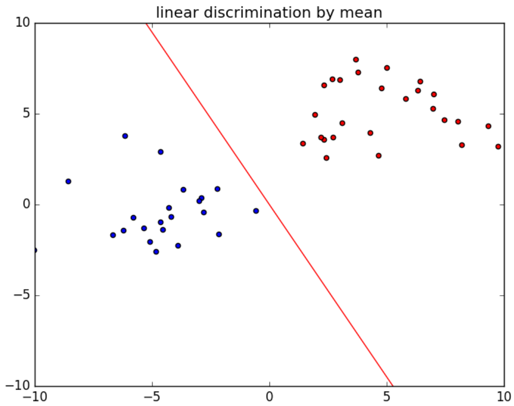
アルゴリズム通り実装してみた. (fisher_1.py)

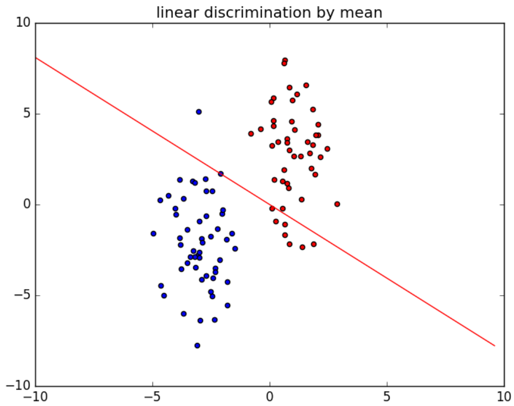
```
x1=data[:N,0]
x2=data[:N,1]
x=np.zeros((N,M))
for n in range(N):
	x[n][0]=1
	x[n][1]=x1[n]
	x[n][2]=x2[n]

t=data[:N,2]

m1=np.zeros(M)
m2=np.zeros(M)
N1=0
N2=0
for n in range(N):
	if t[n]==1:
		m1+=x[n][:]
		N1+=1
	else:
		m2+=x[n][:]
		N2+=1
m1/=N1
m2/=N2
W=m2-m1

#プロット
plt.title("linear discrimination by mean")
plt.xlim([-10,10])
plt.ylim([-10,10])
#データ点散布図
for n in range(N):
	if t[n]==1:
		plt.scatter(x1[n],x2[n],c="red")
```

```
        else:
                plt.scatter(x1[n],x2[n],c="blue")
#線形識別モデル
def model_f(a):
        return -(W[1]*a+W[0])/W[2]

p1 = np.arange(-10, 10, 0.4)
p2 = model_f(p1)
plt.plot(p1, p2, "r")
plt.savefig("f_1.png")
plt.show()
```

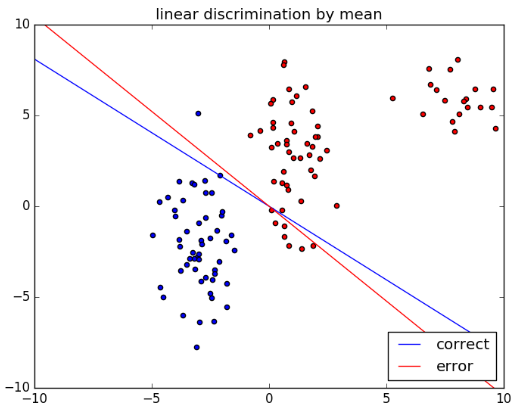
1.4 結果

まず 2 つの例で試してみた.





つぎに, 外れ値があるものに対してこれを実施すると.



外れ値がある場合, これも決定境界の計算に考慮すると, 決定境界は変わる.

2　フィッシャーの線形判別

2.1　問題設定

クラス間分散の最大化, クラス内分散の最小化を目標とするフィッシャーの線形判別について考える.

2.2　アルゴリズム

フィッシャーの線形判別では

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \quad (4.26)$$

ここでは, S_B はクラス間共分散行列

$$S_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \quad (4.27)$$

で, S_W は総クラス内共分散行列

$$S_W = \sum_{x_1 \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{x_n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T \quad (4.28)$$

で与えられる.
$J(\mathbf{w})$ を $\mathbf{w}$ で微分し 0 とすると

$$(\mathbf{w}^T S_B \mathbf{w}) S_W \mathbf{w} = (\mathbf{w}^T S_W \mathbf{w}) S_B \mathbf{w} \quad (4.29)$$

となり

$$\mathbf{w} \propto S_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1) \quad (4.30)$$

となる.

2.3　コード

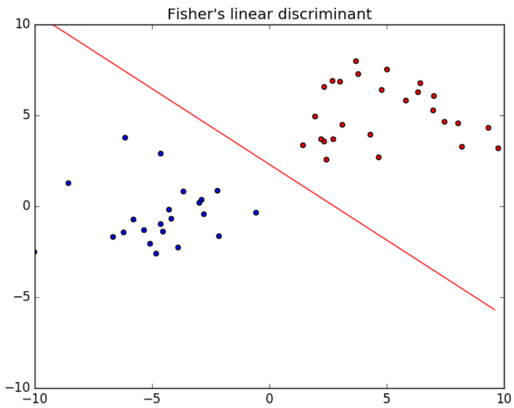
アルゴリズム通り実装してみた. (fisher_4.py)

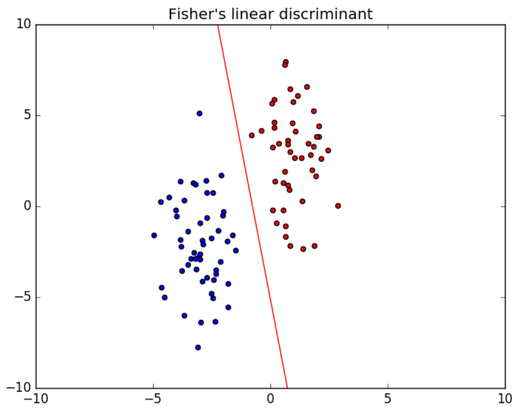
```
x1=data[:N,0]
x2=data[:N,1]
x=np.zeros((N,M))
for n in range(N):
        x[n][0]=1
        x[n][1]=x1[n]
        x[n][2]=x2[n]

t=data[:N,2]

m1=np.zeros(2)
m2=np.zeros(2)
N1=0
N2=0
for n in range(N):
        if t[n]==1:
                m1+=x[n][1:]
                N1+=1
        else:
                m2+=x[n][1:]
                N2+=1
```

```
m1/=N1
m2/=N2

Sw=np.zeros((2,2))
for n in range(N):
        if t[n]==1:
                Sw+=np.outer(x[n][1:]-m1,x[n][1:]-m1)
        else:
                Sw+=np.outer(x[n][1:]-m2,x[n][1:]-m2)

W=np.dot(inv(Sw),m2-m1)

#プロット
plt.title("Fisher's␣linear␣discriminant")
plt.xlim([-10,10])
plt.ylim([-10,10])
#データ点散布図
for n in range(N):
        if t[n]==1:
                plt.scatter(x1[n],x2[n],c="red")
        else:
                plt.scatter(x1[n],x2[n],c="blue")
#線形識別モデル 0.05は勝手に決めた1.5.1節見るべしcdcd
def model_f(a):
        return -(W[0]*a+0.1)/W[1]

p1 = np.arange(-10, 10, 0.4)
p2 = model_f(p1)
plt.plot(p1, p2, "r")
plt.savefig("f_4.png")
plt.show()
```

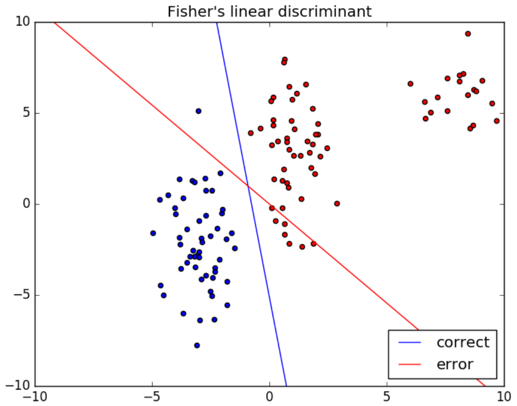
2.4 結果

まず 2 つの例で試してみた.





つぎに, 外れ値があるものに対してこれを実施すると.



外れ値がある場合, これも決定境界の計算に考慮すると, 決定境界は変わる.

3　まとめ

クラス平均の分離度最大化では, 簡単な例でもちゃんと分類できない. フィッシャーの線形判別では簡単な例は正しく分類できたように感じる. ただどちらも, 外れ点があるときには決定境界がずれてしまう.