4.1.3 二乗和誤差最小化による線形識別モデル

平成 28 年 9 月 11 日

概 要

PRMLの「4.1.3 分類における最小二乗」についての実装と考察

目 次

1　2 クラス

1.1　問題設定

2 クラスの線形識別モデルを考える．データ集合をクラス C_1, C_2 に分離する．
ここでは, もっとも簡単な線形識別関数として, 以下の入力ベクトルの線形関数を用いる．

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \quad (4.4)$$

ここで, w_0 はバイアスパラメータといい．この関数値の正負でクラス分類を行う．$y(\mathbf{x}) \geq 0$ のとき x はクラス C_1, $y(\mathbf{x}) < 0$ のとき x はクラス C_2 というようにする．
目標はパラメータ $\tilde{\mathbf{w}} = (w_0, \mathbf{w})$ を決定することである．

1.2　アルゴリズム

二乗和誤差最小化によってパラメータ $\tilde{\mathbf{w}}$ を求める．二乗和誤差は

$$E_D(\tilde{W}) = \frac{1}{2}Tr\left\{(\tilde{X}\tilde{W} - T)^T(\tilde{X}\tilde{W} - T)\right\} \quad (4.15)$$

ここで, $\tilde{X}$ は第 k 行に $(1, x_k)$ を持つ $N \times (D+1)$ 行列, $\tilde{W}$ は第 k 列に $(w_{k0}, \mathbf{w}_k)$ を持つ $(D+1) \times K$ 行列, T は第 k 行に $t_k(1 - of - k$ 符号化法) を持つ $N \times K$ 行列 となっている．
これを, $\tilde{W}$ で微分すれば, 解 $\tilde{W}$ は,

$$\tilde{W} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T T = \tilde{X}^{pinv}T \quad (4.16)$$

ここで, $\tilde{X}^{pinv}$ は $\tilde{X}$ のムーアペンローズの疑似逆行列である．
これより識別関数 $\mathbf{y}(\mathbf{x})$ は K 次元ベクトルで

$$\mathbf{y}(\mathbf{x}) = \tilde{W}^T\tilde{\mathbf{x}} \quad (4.17)$$

となる．
2 クラス分離の場合, $\mathbf{y}(\mathbf{x}) = (y_1(\mathbf{x}), y_2(\mathbf{x}))$ となるが $y_1(\mathbf{x}) = y_2(\mathbf{x})$ が決定境界．

1.3　コード

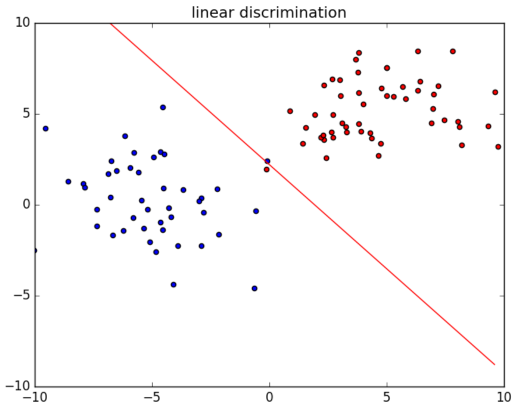
アルゴリズム通り実装してみた．(2class_1.py)

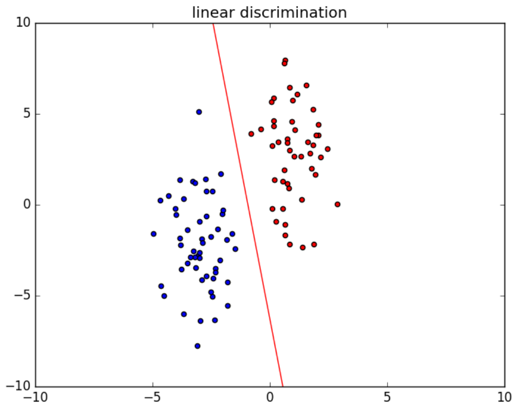
```
#x=(1,x1,x2)
x=np.ones((N,3))
x[:,1]=data[:N,0]
x[:,2]=data[:N,1]
t=np.zeros((N,2))
t[:,0:2]=data[:N,2:4]

#W=(P^tP)^-1P^t T ムーアペンローズの疑似逆行列pinv(P)を用いる
W=np.dot(pinv(x),t).T
print(W)
#プロット
plt.title("linear discrimination")
plt.xlim([-10,10])
plt.ylim([-10,10])
#データ点散布図
for n in range(N):
        if t[n,0]==1:
                plt.scatter(x[n,1],x[n,2],c="red")
```

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

p1 = np.arange(-10, 10, 0.4)
p2 = model_f(p1)
plt.plot(p1, p2, "r")
plt.savefig("p_1.png")
plt.show()
```

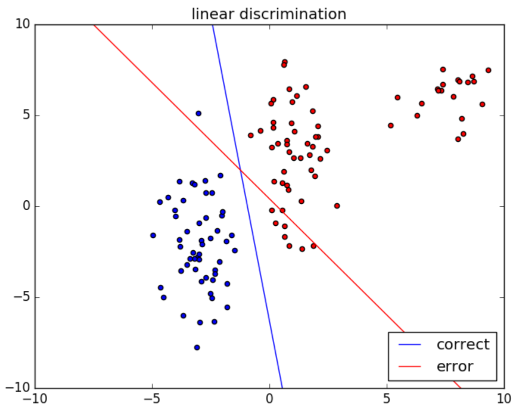
1.4　結果

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.

1.5　気付いたこと

目標変数 t に $1 or -1$ を用いた. (2class.py)

```
#x=(1,x1,x2)
x=np.ones((N,3))
x[:,1]=data[:N,0]
x[:,2]=data[:N,1]
#t=1 or -1
t=data[:N,2]

#W=(P^tP)^-1P^t T ムーアペンローズの疑似逆行列pinv(P)を用いる
W=np.dot(pinv(x),t)

#プロット
plt.title("linear␣discrimination")
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(x[n,1],x[n,2],c="red")
        else:
                plt.scatter(x[n,1],x[n,2],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("2_.png")
plt.show()
```

この場合も 1-of-k 符号加法を用いたときと同じ結果となった.

2 多クラス

2.1 コード

3 クラス分離のコードである.(kclass_3.py)

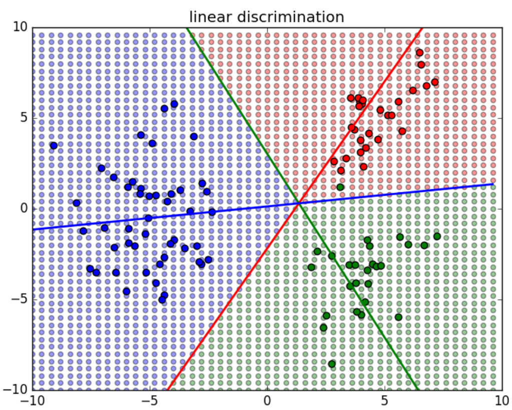
```
#x=(1,x1,x2)
x=np.ones((N,M))
x[:,1:3]=data[:N,0:2]
#t 1-of-k符号化法
t=data[:N,2:5]

#W=(x^tx)^-1x^t T ムーアペンローズの疑似逆行列pinv(x)を用いる
W=np.dot(pinv(x),t).T

#プロット
plt.title("linear␣discrimination")
plt.xlim([-10,10])
plt.ylim([-10,10])

p1=np.arange(-10, 10, 0.4)
p2=np.arange(-10, 10, 0.4)
num=int(20/0.4)

#線形識別モデル
def model_f(a,b,i):
        return W[i,0]+W[i,1]*a+W[i,2]*b

#領域プロット
for i in range(num):
```

```
        for j in range(num):
                if model_f(p1[i],p2[j],0)>model_f(p1[i],p2[j],1) and
                 model_f(p1[i],p2[j],0)>model_f(p1[i],p2[j],2):
                        c="r"
                elif model_f(p1[i],p2[j],1)>model_f(p1[i],p2[j],0) and
                 model_f(p1[i],p2[j],1)>model_f(p1[i],p2[j],2):
                        c="b"
                else:
                        c="g"
                plt.scatter(p1[i], p2[j],c=c,alpha=0.4)

#決定境界モデル
def model_g(a,i,j):
        return -((W[i,1]-W[j,1])/(W[i,2]-W[j,2]))*a
                -(W[i,0]-W[j,0])/(W[i,2]-W[j,2])

#決定境界
z0=model_g(p1,0,1)
z1=model_g(p1,1,2)
z2=model_g(p1,2,0)

#決定境界のプロット
plt.plot(p1,z0,c="g",linewidth=2)
plt.plot(p1,z1,c="r",linewidth=2)
plt.plot(p1,z2,C="b",linewidth=2)

#データ点散布図
for n in range(N):
        if t[n,0]==1:
                plt.scatter(x[n,1],x[n,2],s=40,c="red")
        elif t[n,1]==1:
                plt.scatter(x[n,1],x[n,2],s=40,c="blue")
        else:
                plt.scatter(x[n,1],x[n,2],s=40,c="green")

plt.savefig("k_3.png")
plt.show()
```

プロットに多くを割いていることに注意する.

2.2 結果

3 クラス分離を行った.



3　まとめ

やはり, 外れ点があるとき決定境界はその点に大きく左右されてしまう. 外れ点を無視できるモデルを作る必要があると感じる.