*VII-2-3-2.NNMによる判別分析（解析微分）*

VII-2-3-2-i.準備とデータの読み込み

|  |
| --- |
| #必要なライブラリーの読み込みimport pandas as pdimport numpy as npimport matplotlib.pyplot as plt%matplotlib inline#データの読み込みdf =pd.read\_csv("sample2.csv")xn,D=df.shapeK=5 #クラスの数入力D=D-K #説明変数の数M=D #　第１層の細胞の数X=np.zeros((xn,D))T=np.zeros((xn,K))for i in range(D): X[:,i]=df.iloc[:,K+i]for i in range(K): T[:,i]=df.iloc[:,i]X\_range0=[-2,2] #項目1の範囲X\_range1=[-2,2] #項目2の範囲#データをテストデータとトレーニング用のデータに分割して保存TrainingRatio=1X\_n\_training=int(xn\*TrainingRatio)X\_train=X[:X\_n\_training,:]X\_test=X[X\_n\_training:,:]T\_train=T[:X\_n\_training,:]T\_test=T[X\_n\_training:,:] |

VII-2-3-2-ii.データ分布の確認

|  |
| --- |
| import matplotlib.pyplot as plt%matplotlib inline#散布図の関数定義def show\_data(x,t): wk,N=t.shape col=["b","r","g","y","w"] for k in range (K): plt.plot(x[t[:,k]==1,0],x[t[:,k]==1,1],linestyle='none',marker='o',markeredgecolor='black',color=col[k],alpha=0.8) plt.grid(True)#動作plt.figure(1,figsize=(8,3.7))plt.subplot(1,2,1)show\_data(X\_train,T\_train)plt.xlim(X\_range0)plt.ylim(X\_range1)plt.title('Training data')plt.subplot(1,2,2)show\_data(X\_test,T\_test)plt.xlim(X\_range0)plt.ylim(X\_range1)plt.title('Test data')plt.show() |

VII-2-3-2-iii.feed forward neural network model(FNN)の定義

|  |
| --- |
| #シグモイド関数の定義def sigmoid(alfa): y=1/(1+np.exp(-alfa)) return y#net work def FNN(ab,M,K,x): N,D=x.shape #データサイズと項目数 a=ab[:D\*(D+1)] #中間層の係数の選択 a=a.reshape(M,(M+1)) #M行D+1列に配列 b=ab[D\*(D+1):] #出力層の係数選択 b=b.reshape(K,(M+1)) #K行M+１列に配列 alfa=np.zeros((N,M+1)) #中間層の入力総和の配列枠(定数項を含む) y=np.zeros((N,M+1)) #中間層の出力シグナルの枠(定数項を含む) beta=np.zeros((N,K)) #出力層の入力総和の配列枠 z=np.zeros((N,K)) for n in range (N): y[n,0]=1 #出力層のダミー変数 for m in range (M): #中間層 alfa[n,m]=np.dot(a[m,:],np.r\_[1,x[n,:]]) #中間層の入力総和の計算 y[n,m+1]=sigmoid(alfa[n,m]) sum=0 #ソフトマック関数の分母 for k in range(K): beta[n,k]=np.dot(b[k,:],y[n,:]) sum=sum+np.exp(beta[n,k]) for k in range(K): z[n,k]=np.exp(beta[n,k])/sum return z,beta,y,alfa#リスト4-4.平均交差エントロピー誤差の定義def CE\_FNN(ab,M,K,x,t): N,D=x.shape z,beta,y,alfa=FNN(ab,M,K,x) ce=-np.dot(t.reshape(-1),np.log(z.reshape(-1)))/N return ce |

VII-2-3-2-iv.解析的微分の定義

|  |
| --- |
| #交差エントロピー誤差の微分の定義def dCE\_FNN(ab,M,K,x,t): #入力変数の枠 N,D=x.shape a=ab[:M\*(D+1)] a=a.reshape((M,D+1)) b=ab[M\*(D+1):] b=b.reshape((K,D+1)) #FNNにxを入力して、z,beta,y,alfa取得 z,beta,y,alfa=FNN(ab,M,K,x) #出力変数の枠 dab=np.zeros\_like(ab) da=np.zeros((M,D+1)) db=np.zeros((K,M+1)) delta1=np.zeros(M+1) #第一層の誤差 delta2=np.zeros(K) #第二層の誤差 for n in range(N): #第二層の誤差 for k in range(K): delta2[k]=(z[n,k]-t[n,k]) #第一層の誤差 for j in range(M+1): delta1[j]=y[n,j]\*(1-y[n,j])\*np.dot(b[:,j],delta2) #bの勾配 for k in range(K): db[k,:]=db[k,:]+delta2[k]\*y[n,:]/N #aの勾配 for j in range(M): da[j,:]=da[j,:]+delta1[j+1]\*np.r\_[1,x[n,:]]/N #daとｄｂを１列に配列 dab=np.c\_[da.reshape((1,M\*(D+1))),db.reshape((1,K\*(M+1)))] dab=dab.reshape(-1) return dab#微分値の図示def Show\_dAB(ab,m): N=ab.shape[0] plt.bar(range(1,M\*3+1),ab[:M\*3],align="center",color="black") plt.bar(range(M\*3+1,N+1),ab[M\*3:],align="center",color='cornflowerblue') plt.xticks(range(1,N+1)) plt.xlim(0,N+1) |

VII-2-3-2-v.勾配降下法の実行

|  |
| --- |
| #リスト4-5.解析的手法による勾配法の実施import timedef Fit\_FNN(ab\_init,M,K,x\_train,t\_train,x\_test,t\_test,n,lr): ab=ab\_init.copy() err\_train=np.zeros(n) err\_test=np.zeros(n) ab\_hist=np.zeros((n,len(ab\_init))) for i in range(n): ab=ab-lr\*dCE\_FNN(ab,M,K,x\_train,t\_train) err\_train[i]=CE\_FNN(ab,M,K,x\_train,t\_train) err\_test[i]=CE\_FNN(ab,M,K,x\_test,t\_test) ab\_hist[i,:]=ab return ab,ab\_hist,err\_train,err\_test#実行startTime=time.time()np.random.seed(1)AB\_init=np.random.normal(0,0.01,M\*3+K\*(M+1))N\_step=1000LR=0.5AB,AB\_hist,Err\_train,Err\_test=Fit\_FNN(AB\_init,M,K,X\_train,T\_train,X\_test,T\_test,N\_step,LR)calculation\_time=time.time()-startTimeprint("Calculation time:{0:.3f} sec".format(calculation\_time)) |

VII-2-3-2-vi.経緯と結果の表示

|  |
| --- |
| #判別境界の表示1def show\_FNN(ab,M,K): xn=60 #解像度 gn=xn\*xn x0=np.linspace(X\_range0[0],X\_range0[1],xn) x1=np.linspace(X\_range1[0],X\_range1[1],xn) xx0,xx1=np.meshgrid(x0,x1) x=np.c\_[xx1.reshape(gn,1), xx0.reshape(gn,1)] z,beta,y,alfa=FNN(AB,M,K,x) plt.figure(1,figsize=(4,4)) for k in range(K): f=z[:,k] f=f.reshape(xn,xn) f=f.T cont=plt.contour(xx0,xx1,f,levels=[0.5,0.95],colors=['cornflowerblue','black']) cont.clabel(fmt='%.2f',fontsize=9) plt.xlim(X\_range0) plt.ylim(X\_range1)#表示plt.figure(1,figsize=(3,3))show\_data(X\_test,T\_test)show\_FNN(AB,M,K)plt.show()#計算過程の表示plt.figure(1,figsize=(12,3))plt.subplots\_adjust(wspace=0.5)#誤差の変化経緯plt.subplot(1,3,1)plt.plot(Err\_train,color='black',label='training')plt.plot(Err\_test,color='cornflowerblue',label='test')plt.legend()#係数の推定値の変化plt.subplot(1,3,2)plt.plot(AB\_hist[:,:M\*3],'black')plt.plot(AB\_hist[:,M\*3:],'cornflowerblue')#境界線の表示plt.subplot(1,3,3)show\_data(X\_train,T\_train)show\_FNN(AB,M,K)plt.show()print(AB) |

VII-2-3-2-vii.拡大図

|  |
| --- |
| #リスト4-7.判別境界の表示(拡大図)def show\_FNN(ab,M,K): xn=60 #解像度 gn=xn\*xn x0=np.linspace(X\_range0[0],X\_range0[1],xn) x1=np.linspace(X\_range1[0],X\_range1[1],xn) xx0,xx1=np.meshgrid(x0,x1) x=np.c\_[xx1.reshape(gn,1), xx0.reshape(gn,1)] z,beta,y,alfa=FNN(AB,M,K,x) plt.figure(1,figsize=(4,4)) for k in range(K): f=z[:,k] f=f.reshape(xn,xn) f=f.T cont=plt.contour(xx0,xx1,f,levels=[0.5,0.95],colors=['cornflowerblue','black']) cont.clabel(fmt='%.2f',fontsize=9) plt.xlim(X\_range0) plt.ylim(X\_range1)#表示import matplotlibmatplotlib.use('Agg')import matplotlib.pyplot as pltplt.figure(1,figsize=(4,4))show\_data(X\_train,T\_train)show\_FNN(AB,M,K)plt.show()fig=plt.show()plt.savefig("analyticfig,png")*#図の保存* |

VII-2-3-2-viii.推定値の出力と保存

|  |
| --- |
| #リスト4-8.推定値の出力と保存print(AB)AA=AB[:D\*(D+1)] #中間層の係数の選択AA=AA.reshape(M,(M+1)) #M行D+1列に配列eras]BB=AB[M\*(D+1):] #出力層の係数選≅BB=BB.reshape(K,(M+1)) #K行M+１列に配列#係数の保存df=pd.DataFrame(AA)df.to\_csv('parameter1Aan.csv')df=pd.DataFrame(BB)df.to\_csv('parameter1Ban.csv') |