*VII-2-1.ロジスティック回帰*

VII-2-1-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("sample1.csv")xn,D=df.shapeK=3 #クラスの数入力D=D-K #説明変数の数D1=D+1 #定数項を加えた変数ベクトルのディメンジョンX=np.zeros((xn,D))T3=np.zeros((xn,K))for i in range(D): X[:,i]=df.iloc[:,3+i]for i in range(K): T3[:,i]=df.iloc[:,i]#縮尺倍率を決めるscale=100X=X/scale# 作業内容の定義def show\_data(x,t): ni,nj=t.shape #行数・列数を取得 for j in range (nj):plt.plot(x[t[:,j]==1,0],x[t[:,j]==1,1],linestyle="none",markeredgecolor="black",marker="o",color=X\_col[j],) plt.grid(True)#色の指定X\_col=["b","r","g","C","M","Y","K","W"]fig=plt.figure()show\_data(X,T3)plt.show() |

VII-2-1-ii.判別するクラスを指定

|  |
| --- |
| #リスト1-2.判別する一対のクラスを決定する。wk=np.sum(df,axis=0) #列方向に総和を求める#判別するクラスの選択Cluster1=1 #判別するクラスターの一つCluster2=2 #もう一つのクラスターCl0=Cluster1-1Cl1=Cluster2-1n0=wk[Cl0]n0=int(n0)n1=wk[Cl1]n1=int(n1)ns=n0+n1T2=np.zeros((ns,2))X2=np.zeros((ns,D))count=0for i in range(xn): if T3[i,Cl0]==1: T2[count,0]=1 X2[count,:]=X[i,:] count=count+1 if T3[i,Cl1]==1: T2[count,1]=1 X2[count,:]=X[i,:] count=count+1fig=plt.figure()show\_data(X2,T2)plt.show() |

VII-2-1-iii.関数の定義と実行

|  |
| --- |
| #２変数のロジスティック関数の定義def logistic2(x1,x2,w): y=1/(1+np.exp(-(w[0]+w[1]\*x1+w[2]\*x2))) return y#交差エントロピー誤差の定義def cee\_logistic2(w,x,t): X\_r,X\_c=x.shape y=logistic2(x[:,0],x[:,1],w) cee=0 for i in range(X\_r): cee=cee-(t[i,0]\*np.log(y[i])+(1-t[i,0])\*np.log(1-y[i])) dee=cee/X\_r return cee#交差エントロピー誤差の微分def dcee\_logistic2(w,x,t): X\_r,X\_c=x.shape y=logistic2(x[:,0],x[:,1],w) dcee=np.zeros(3) for j in range(X\_r): dcee[0]=dcee[0]+y[j]-t[j,0] dcee[1]=dcee[1]+(y[j]-t[j,0])\*x[j,0] dcee[2]=dcee[2]+(y[j]-t[j,0])\*x[j,1] dcee=dcee/X\_r return dcee #最尤法による判別関数の推定（共役勾配法使う）#scipty.optimize improt minimizefrom scipy.optimize import minimize#最小化のターゲット、微分式、最小化法の指定def fit\_logistic2(w\_init,x,t): res=minimize(cee\_logistic2,w\_init,args=(x,t),jac=dcee\_logistic2,method="CG") return res.x |

VII-2-1-iv.結果の図示

|  |
| --- |
| #リスト1-4.分析結果の図示#最適Wを求める#1出力W\_init = [-1,0,0]W=fit\_logistic2(W\_init,X2,T2)print (W)cee=cee\_logistic2(W,X2,T2)print (cee)#等高線を書く作業の定義def show\_contour\_logistic2(w): xx0,xx1=np.meshgrid(x0,x1) y=logistic2(xx0,xx1,w) cont=plt.contour(xx0,xx1,y,levels=(0.05,0.5,0.95),colors=['k','cornflowerblue','k']) cont.clabel(fmt='%.2f',fontsize=10) plt.grid(True)#実行X\_range0=[1.4,2.2] #x0軸の範囲X\_range1=[0.3,2.4] #x軸の範囲x0g=71 #x0軸のグリッド数x1g=211 #x1軸の議リッド数x0=np.linspace(X\_range0[0],X\_range0[1],x0g)x1=np.linspace(X\_range1[0],X\_range1[1],x1g)fig1=plt.figure()show\_data(X2,T2)show\_contour\_logistic2(W)plt.show() |

VII-2-1-v.等高線図と3Dグラフ

|  |
| --- |
| #リスト1-4.分析結果の図示#最適Wを求める#1出力W\_init = [-1,0,0]W=fit\_logistic2(W\_init,X2,T2)print (W)cee=cee\_logistic2(W,X2,T2)print (cee)#等高線を書く作業の定義def show\_contour\_logistic2(w): xx0,xx1=np.meshgrid(x0,x1) y=logistic2(xx0,xx1,w) cont=plt.contour(xx0,xx1,y,levels=(0.05,0.5,0.95),colors=['k','cornflowerblue','k']) cont.clabel(fmt='%.2f',fontsize=10) plt.grid(True)#実行X\_range0=[1.4,2.2] #x0軸の範囲X\_range1=[0.3,2.4] #x軸の範囲x0g=71 #x0軸のグリッド数x1g=211 #x1軸の議リッド数x0=np.linspace(X\_range0[0],X\_range0[1],x0g)x1=np.linspace(X\_range1[0],X\_range1[1],x1g)fig1=plt.figure()show\_data(X2,T2)show\_contour\_logistic2(W)plt.show() |

VII-2-1-vi.3Dグラフ(color)

|  |
| --- |
| #リスト1-6.確率分布をカラー入り3dで表す。def show3d\_logistic2\_color(ax,w): xx0,xx1=np.meshgrid(x0,x1) y=logistic2(xx0,xx1,w) ax.plot\_surface(xx0,xx1,y,edgecolor="gray",rstride=20,cstride=20,alpha=0.3,shade=True,cmap="plasma")Ax=plt.subplot(1,1,1,projection='3d')fig3=plt.figure()show3d\_logistic2\_color(Ax,W) |

VII-2-1-vii.結果の保存

|  |
| --- |
| #リスト1-7.分析結果の出力#推定されたパラメータ―F=pd.DataFrame(W)import pandas as pdimport csvF.to\_csv('parameter1.csv')#等高線入りの散布図fig1.savefig('ABcontour.png')#3dの確率とデータfig2.savefig('AB3d\_data.png')#3確率fig3.savefig('ABp.png') |