# 1.导包

|  |
| --- |
| import math  import numpy as np  import pandas as pd  from sklearn import preprocessing  from sklearn.linear\_model import LogisticRegression  from sklearn.cross\_validation import train\_test\_split  from numpy import loadtxt,where  from pylab import scatter,show,legend,xlabel,ylabel |

# 2.数据准备与可视化

|  |
| --- |
| min\_max\_scaler=preprocessing.MinMaxScaler(feature\_range=(-1,1))  f=open('/data/machine\_learning/分类数据/logistic/data.csv')  df=pd.read\_csv(f,header=0) #header=0表示第一行是列索引，不作为数据  print(df[0:1])  print(df.columns)  df.columns=['grade1','grade2','label']  X=df[["grade1",'grade2']] #特征数据  X=np.array(X)  X=min\_max\_scaler.fit\_transform(X) #归一化  Y=df["label"].map(lambda x:float(x.rstrip(";")))  Y=np.array(Y)  print(X.shape)  X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.33)  print(X\_train.shape,X\_test.shape) |

可视化

|  |
| --- |
| #可视化数据  pos=where(Y==1) #返回Y==1的索引  neg=where(Y==0) #返回Y==0的索引  scatter(X[pos,0],X[pos,1],marker='o',c='b')  scatter(X[neg,0],X[neg,1],marker='x',c='r')  # xlabel('Exam 1 score')  # ylabel('Exam 2 score')  legend(['Not Admitted','Admitted'])  show() |

# 3.sklearn训练logistic模型

|  |
| --- |
| #训练模型  clf=LogisticRegression()  clf.fit(X\_train,Y\_train)  print('sklearn score:',clf.score(X\_test,Y\_test)) |

# 4.自己实现logistic回归

|  |
| --- |
| #sigmoid函数  def sigmoid(z):  result=float(1.0/float((1.0+math.exp(-1.0\*z))))  return result |

|  |
| --- |
| #假设函数  def hypothesis(theta,x):  z=0  for i in range(len(theta)):  z+=x[i]\*theta[i]  return sigmoid(z) |

|  |
| --- |
| #损失函数  def cost\_function(X,Y,theta,m):  sum\_error=0  for i in range(m):  xi=X[i] #第i个样本  hi=hypothesis(theta,xi)  if Y[i]==1:  error=Y[i]\*math.log(hi)  elif Y[i]==0:  error=(1-Y[i])\*math.log(1-hi)  sum\_error+=error  const=-1/m  J=const\*sum\_error  print('cost is:',J)  return J |

|  |
| --- |
| #梯度下降的损失函数  def cost\_function\_derivative(X,Y,theta,j,m,alpha):  sum\_error=0  for i in range(m):  xi=X[i] #第i个样本  xij=xi[j] #第i个样本的第j个特征  hi=hypothesis(theta,X[i])  error=(hi-Y[i])\*xij #  sum\_error+=error  m=len(Y)  constant=float(alpha)/float(m)  J=constant\*sum\_error  return J |
| #梯度下降更新theta  def gradient\_decent(X,Y,theta,m,alpha):  new\_theta=[]  constant=alpha/m  for j in range(len(theta)):  cfderivative=cost\_function\_derivative(X,Y,theta,j,m,alpha)  new\_theta\_value=theta[j]-cfderivative  new\_theta.append(new\_theta\_value)  return new\_theta |

|  |
| --- |
| def logistic\_regression(X,Y,alpha,theta,num\_iters):  m=len(Y)  for x in range(num\_iters):  new\_theta=gradient\_decent(X,Y,theta,m,alpha)  theta=new\_theta  if x%100==0:  cost\_function(X,Y,theta,m)  print("theta:",theta)  print("cost is:",cost\_function(X,Y,theta,m))  declare\_winner(theta) |

# 5.两种方法对比

|  |
| --- |
| def declare\_winner(theta):  score=0  winner=""  scikit\_score=clf.score(X\_test,Y\_test)  length=len(X\_test)  for i in range(length):  prediction=round(hypothesis(X\_test[i],theta))  answer=Y\_test[i]  if prediction==answer:  score+=1    my\_score=float(score)/float(length)  if my\_score >scikit\_score:  print("我赢了!")  elif my\_score==scikit\_score:  print("双赢")  else:  print("scikit赢了")  print("我的得分:",my\_score)  print("scikit得分:",scikit\_score) |
| initial\_theta=[0,0]#theta初始化  alpha=0.1 #学习率  iterations=1000#迭代次数  logistic\_regression(X,Y,alpha,initial\_theta,iterations) |