## 附录1:主要程序代码

说明：以下程序都基于python3.5.2(32bit,windows)

所需依赖库有numpy, matplotlib, scipy, sklearn，所有程序都在支撑材料内，但由于几个程序微修改输入输出部分就多次使用，因此不重复在附录内出现。

附录1.1 数据转换zb2.py：

|  |
| --- |
| l=list(open('zb2.txt'))  s=[]  for i in l:  k=i.split()  k[0],k[1]=k[1],k[0]  b="\t".join(k)  s.append(b)  f=open("zb2cl2.txt","w")  f.write("\n".join(s))  f.close() |

附录1.2数据转换zb7.py

|  |
| --- |
| l=list(open('da3.txt'))  s=[]  for i in l:  k=i.split()  b="\t".join(k)  s.append(b)  for k in range(0,7):  o=open("聚类%d.txt"%(k))  jlj=o.read()  jl=jlj.split()  o.close()  for j in jl:  s[int(j)]=s[int(j)]+"\t"+"%d"%(k)  f=open("dsf.txt","w")  f.write("\n".join(s))  f.close() |

附录1.3分布图绘制（绘图.py）

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  l=list(open('zb2.txt'))  x=[]  y=[]  for i in l:  k=i.split()  y.append(float(k[0]))  x.append(float(k[1]))  #产生测试数据  fig = plt.figure()  ax1 = fig.add\_subplot(111)  #设置标题  ax1.set\_title('Scatter Plot')  #设置X轴标签  plt.xlabel('X')  #设置Y轴标签  plt.ylabel('Y')  #画散点图  ax1.scatter(x,y,s=3,c = 'r',marker = '.')  #设置图标  plt.legend('x1')  #显示所画的图  plt.show() |

附录1.4计算任务点到中心点的距离并输出到文件（距离计算.py）

|  |
| --- |
| from math import radians, cos, sin, asin, sqrt  from scipy import stats  import numpy as np  import matplotlib.pyplot as plt  def haversine(lat1,lon1,lat2,lon2):  lon1,lat1,lon2,lat2=map(radians, [lon1,lat1,lon2,lat2])  dlon=lon2-lon1  dlat=lat2-lat1  a=sin(dlat/2)\*\*2+cos(lat1)\*cos(lat2)\*sin(dlon/2)\*\*2  c=2\*asin(sqrt(a))  r=6371  return c \*r\*1000  l=list(open('fj1.txt'))  cen=list(open('centerpoints.txt'))  loc=[]  pr=[]  ce=[]  for i in cen:  te=i.split()  te[0],te[1]=float(te[0]),float(te[1])  ce.append(te)  for i in l:  k=i.split()  k[0],k[1]=float(k[0]),float(k[1])  pr.append(float(k[2]))  des=[]  for te in ce:  des.append(haversine(k[0],k[1],te[0],te[1]))  des.sort()  loc.append(des[0])  fig = plt.figure()  ax1 = fig.add\_subplot(111)  ax1.set\_title('price-distance')  plt.xlabel('distance')  plt.ylabel('price')  ax1.scatter(loc,pr,s=3,c = 'r',marker = '.')  plt.legend('x1')  plt.show()  c=[loc,pr]  d=np.corrcoef(c)  print(d)  print(stats.spearmanr(loc, pr, axis=None))  f=open("dis.txt","w")  for i in range(0,len(loc)):  loc[i]=str(loc[i])  f.write("\n".join(loc))  f.close() |

附件1.5 利用logestifc对数据点分类（logestifc分类.py）

|  |
| --- |
| from numpy import \*  import matplotlib.pyplot as plt  import time  def sigmoid(inX):      return 1.0 / (1 + exp(-inX))  def trainLogRegres(train\_x, train\_y, opts):      # calculate training time      startTime = time.time()      numSamples, numFeatures = shape(train\_x)      alpha = opts['alpha']; maxIter = opts['maxIter']      weights = ones((numFeatures, 1))      # optimize through gradient descent algorilthm      for k in range(maxIter):          if opts['optimizeType'] == 'gradDescent': # gradient descent algorilthm              output = sigmoid(train\_x \* weights)              error = train\_y - output              weights = weights + alpha \* train\_x.transpose() \* error          elif opts['optimizeType'] == 'stocGradDescent': # stochastic gradient descent              for i in range(numSamples):                  output = sigmoid(train\_x[i, :] \* weights)                  error = train\_y[i, 0] - output                  weights = weights + alpha \* train\_x[i, :].transpose() \* error          elif opts['optimizeType'] == 'smoothStocGradDescent': # smooth stochastic gradient descent              # randomly select samples to optimize for reducing cycle fluctuations              dataIndex = list(range(numSamples))              for i in range(numSamples):                  alpha = 4.0 / (1.0 + k + i) + 0.01                  randIndex = int(random.uniform(0, len(dataIndex)))                  output = sigmoid(train\_x[randIndex, :] \* weights)                  error = train\_y[randIndex, 0] - output                  weights = weights + alpha \* train\_x[randIndex, :].transpose() \* error                  del(dataIndex[randIndex]) # during one interation, delete the optimized sample          else:              raise NameError('Not support optimize method type!')      print('Congratulations, training complete! Took %fs!' % (time.time() - startTime))      return weights  def testLogRegres(weights, test\_x, test\_y):      numSamples, numFeatures = shape(test\_x)      matchCount = 0      for i in range(numSamples):          predict = sigmoid(test\_x[i, :] \* weights)[0, 0] > 0.5          if predict == bool(test\_y[i, 0]):              matchCount += 1      accuracy = float(matchCount) / numSamples      return accuracy  def showLogRegres(weights, train\_x, train\_y):      # notice: train\_x and train\_y is mat datatype      numSamples, numFeatures = shape(train\_x)      if numFeatures != 3:          print("Sorry! I can not draw because the dimension of your data is not 2!")          return 1      # draw all samples      for i in range(numSamples):          if int(train\_y[i, 0]) == 0:              plt.plot(train\_x[i, 1], train\_x[i, 2],12, 'or')          elif int(train\_y[i, 0]) == 1:              plt.plot(train\_x[i, 1], train\_x[i, 2],12, 'ob')      # draw the classify line      min\_x = min(train\_x[:, 1])[0, 0]      max\_x = max(train\_x[:, 1])[0, 0]      weights = weights.getA() # convert mat to array      y\_min\_x = float(-weights[0] - weights[1] \* min\_x) / weights[2]      y\_max\_x = float(-weights[0] - weights[1] \* max\_x) / weights[2]      plt.plot([min\_x, max\_x], [y\_min\_x, y\_max\_x], '-g')      plt.xlabel('X1'); plt.ylabel('X2')      plt.show()  def loadData():      train\_x = []      train\_y = []      fileIn = open('dis3.txt')      for line in fileIn.readlines():          lineArr = line.strip().split()          train\_x.append([1.0, float(lineArr[0]), float(lineArr[1])])          train\_y.append(float(lineArr[2]))      return mat(train\_x), mat(train\_y).transpose()  ## step 1: load data  print("step 1: load data...")  train\_x, train\_y = loadData()  test\_x = train\_x; test\_y = train\_y  ## step 2: training...  print("step 2: training...")  opts = {'alpha': 0.02, 'maxIter': 800, 'optimizeType': 'smoothStocGradDescent'}  optimalWeights = trainLogRegres(train\_x, train\_y, opts)  ## step 3: testing  print("step 3: testing...")  accuracy = testLogRegres(optimalWeights, test\_x, test\_y)  ## step 4: show the result  print("step 4: show the result...")  print('The classify accuracy is: %.3f%%' % (accuracy \* 100))  showLogRegres(optimalWeights, train\_x, train\_y) |

附件1.6计算任务点到最近中心点的距离，根据需要可选标准化数据，并根据需要筛选后输出或直接输出到文件（距离画点3 - 标准化.py）

|  |
| --- |
| from math import radians, cos, sin, asin, sqrt  from scipy import stats  import numpy as np  from sklearn import preprocessing  def haversine(lat1,lon1,lat2,lon2):  lon1,lat1,lon2,lat2=map(radians, [lon1,lat1,lon2,lat2])  dlon=lon2-lon1  dlat=lat2-lat1  a=sin(dlat/2)\*\*2+cos(lat1)\*cos(lat2)\*sin(dlon/2)\*\*2  c=2\*asin(sqrt(a))  r=6371  return c \*r\*1000  l=list(open('fj2.txt'))  cen=list(open('centerpoints.txt'))  loc=[]  de=[]  pr=[]  ce=[]  ye=[]  for i in cen:  te=i.split()  te[0],te[1]=float(te[0]),float(te[1])  ce.append(te)  for i in l:  k=i.split()  k[0],k[1]=float(k[0]),float(k[1])  pr.append(float(k[2]))  des=[]  for te in ce:  des.append(haversine(k[0],k[1],te[0],te[1]))  des.sort()  de.append(des[0])  ye.append(k[2])  de1=np.array(de)  de1=preprocessing.scale(de1)  pr1=np.array(pr)  pr1=preprocessing.scale(pr1)  print(stats.spearmanr(de1, pr1, axis=None))  for j in range(0,len(de1)):  loc.append(str(de1[j])+"\t"+str(pr1[j])+"\t"+k[2])  f=open("dis4.txt","w")  f.write("\n".join(loc))  f.close() |

附件1.7logistic regression回归并做图（loge回归.py）

|  |
| --- |
| import numpy as np  from sklearn import linear\_model, datasets  import matplotlib.pyplot as plt  from scipy.stats import norm  l=list(open('dis4.txt'))  x=[]  y=[]  for i in l:  k=i.split()  k[0],k[1]=float(k[0]),float((k[1]))  y.append(int((k[0])\*1000))  x.append(int(k[1]\*10))  X=np.array(x)  y=np.array(y)  #---logistic regression--------  from sklearn.linear\_model import LogisticRegression  logclf = LogisticRegression()  logclf.fit(X.reshape(len(x),1), y)  def lr\_model(clf, X):  k=clf.intercept\_  n=clf.coef\_ \* X  for i in range(0,len(k)):  for j in range(0,len(n)):  n[j]=n[j]+k[i]  for j in range(1,len(n)):  n[0]=n[0]+n[j]  return 1.0 / (1.0 + np.exp(-(n[0])))  #----plot---------------------------  plt.figure(figsize=(10, 5))  plt.scatter(X, y,2,color="b")  plt.plot(X, lr\_model(logclf, X).ravel(),"+",color="r")  plt.xlabel("feature value")  plt.ylabel("class")  plt.title("logistic fit")  plt.grid(True, linestyle='-', color='0.75')  plt.tight\_layout(pad=0.4, w\_pad=0, h\_pad=1.0)  plt.show() |

附件1.8预处理待拟合数据来喂给拟合程序使用（拟合数据准备.py）

|  |
| --- |
| from math import radians, cos, sin, asin, sqrt  def haversine(lat1,lon1,lat2,lon2):  lon1,lat1,lon2,lat2=map(radians, [lon1,lat1,lon2,lat2])  dlon=lon2-lon1  dlat=lat2-lat1  a=sin(dlat/2)\*\*2+cos(lat1)\*cos(lat2)\*sin(dlon/2)\*\*2  c=2\*asin(sqrt(a))  r=6371  return c \*r\*1000  l=list(open('ex3data2.txt'))  cen=list(open('ex2data2.txt'))  loc=[]  ci=[]  ce=[]  for i in cen:  te=i.split()  te[0],te[1],te[2]=float(te[0]),float(te[1]),float(te[2])  ce.append(te)  for i in l:  k=i.split()  k[0],k[1]=float(k[0]),float(k[1])  des=0.0  for te in ce:  if(haversine(k[0],k[1],te[0],te[1]))<=3000:  des=des+te[2]  if des>0:  #loc.append(str(1/des)+"\t"+str(k[3])+"\t"+str(k[2]))  loc.append(str(1/des)+"\t"+str(k[3]))  f=open("nhc3.txt","w")  f.write("\n".join(loc))  f.close() |

附件1.9函数回归程序，通过梯度下降算法给出拟合曲线，并给出预测结果（多元线性回归.py）

|  |
| --- |
| # ## 多变量线性回归(Linear Regression with Multiple Variables)  import random  import numpy as np  import matplotlib.pyplot as plt  plt.rcParams['image.interpolation'] = 'nearest'  plt.rcParams['image.cmap'] = 'gray'  def load\_exdata(filename):  data = []  with open(filename, 'r') as f:  for line in f.readlines():  line = line.split()  current = [float(item) for item in line]  #5.5277,9.1302  data.append(current)  return data  data = load\_exdata('nh1.txt');  data = np.array(data,np.int64)  x = data[:,(0,1)].reshape((-1,2))  y = data[:,2].reshape((-1,1))  m = y.shape[0]  print('First 10 examples from the dataset: \n')  print(' x = ',x[range(10),:],'\ny=',y[range(10),:])  def featureNormalize(X):  X\_norm = X;  mu = np.zeros((1,X.shape[1]))  sigma = np.zeros((1,X.shape[1]))  for i in range(X.shape[1]):  mu[0,i] = np.mean(X[:,i]) # 均值  sigma[0,i] = np.std(X[:,i]) # 标准差  # print(mu)  # print(sigma)  X\_norm = (X - mu) / sigma  return X\_norm,mu,sigma  #计算损失  def computeCost(X, y, theta):  m = y.shape[0]  # J = (np.sum((X.dot(theta) - y)\*\*2)) / (2\*m)  C = X.dot(theta) - y  J2 = (C.T.dot(C))/ (2\*m)  return J2  #梯度下降  def gradientDescent(X, y, theta, alpha, num\_iters):  m = y.shape[0]  #print(m)  # 存储历史误差  J\_history = np.zeros((num\_iters, 1))  for iter in range(num\_iters):  # 对J求导，得到 alpha/m \* (WX - Y)\*x(i)， (3,m)\*(m,1) X (m,3)\*(3,1) = (m,1)  theta = theta - (alpha/m) \* (X.T.dot(X.dot(theta) - y))  J\_history[iter] = computeCost(X, y, theta)  return J\_history,theta    iterations = 20000 #迭代次数  alpha = 0.01 #学习率  x = data[:,(0,1)].reshape((-1,2))  y = data[:,2].reshape((-1,1))  m = y.shape[0]  x,mu,sigma = featureNormalize(x)  X = np.hstack([x,np.ones((x.shape[0], 1))])  # X = X[range(2),:]  # y = y[range(2),:]  theta = np.zeros((3, 1))  j = computeCost(X,y,theta)  J\_history,theta = gradientDescent(X, y, theta, alpha, iterations)  print('Theta found by gradient descent',theta)  # In[17]:  plt.plot(J\_history)  plt.ylabel('lost');  plt.xlabel('iter count')  plt.title('convergence graph')  plt.show()  # ### 预测  def predict(data):  testx = np.array(data)  testx = ((testx - mu) / sigma)  testx = np.hstack([testx,np.ones((testx.shape[0], 1))])  price = testx.dot(theta)  #print('price is %d ' % (price))  return float(price)  #预测输出文件  print(predict([1/1,15000]))  l=list(open('nhc3.txt'))  s=[]  for i in l:  k=i.split()  k.append(str(predict([float(k[0]),float(k[1])])))  s.append(k[2])  f=open("prec1.txt","w")  f.write("\n".join(s))  f.close() |

附件1.10拟合数据修正1，修正了物价指数的缩放

|  |
| --- |
| from math import radians, cos, sin, asin, sqrt  def haversine(lat1,lon1,lat2,lon2):  lon1,lat1,lon2,lat2=map(radians, [lon1,lat1,lon2,lat2])  dlon=lon2-lon1  dlat=lat2-lat1  a=sin(dlat/2)\*\*2+cos(lat1)\*cos(lat2)\*sin(dlon/2)\*\*2  c=2\*asin(sqrt(a))  r=6371  return c \*r\*1000  l=list(open('ex3datapr.txt'))  cen=list(open('prec1.txt'))  ciy=list(open('cities.txt'))  loc=[]  ci=[]  pr=[]  ce=[]  for i in cen:  te=i.split()  te[0]=float(te[0])  pr.extend(te)  for i in ciy:  te=i.split()  te[0],te[1],te[2]=float(te[0]),float(te[1]),float(te[2])  ci.append(te)  for i in range(0,len(l)):  k=l[i].split()  cit=[]  for te in ci:  cit.append(haversine(float(k[0]),float(k[1]),te[0],te[1]))  tem=[]  tem.extend(cit)  cit.sort()  ind=tem.index(cit[0])  pr[i]=pr[i]\*(ci[ind][2]/ci[1][2]) #价格缩放  k.append(str(pr[i]))  loc.append("\t".join(k))  f=open("resulc.txt","w")  f.write("\n".join(loc))  f.close() |

附件1.11整理输出的价格数据（拟合数据修正2.py）

|  |
| --- |
| from math import radians, cos, sin, asin, sqrt  def haversine(lat1,lon1,lat2,lon2):  lon1,lat1,lon2,lat2=map(radians, [lon1,lat1,lon2,lat2])  dlon=lon2-lon1  dlat=lat2-lat1  a=sin(dlat/2)\*\*2+cos(lat1)\*cos(lat2)\*sin(dlon/2)\*\*2  c=2\*asin(sqrt(a))  r=6371  return c \*r\*1000  l=list(open('ex3data2.txt'))  cen=list(open('ex2data2.txt'))  loc=[]  ci=[]  ce=[]  for i in cen:  te=i.split()  te[0],te[1],te[2]=float(te[0]),float(te[1]),float(te[2])  ce.append(te)  for i in l:  k=i.split()  des=0.0  for te in ce:  if(haversine(float(k[0]),float(k[1]),te[0],te[1]))<=3000:  des=des+te[2]  if des>0:  loc.append("\t".join(k))  f=open("ex3datapr.txt","w")  f.write("\n".join(loc))  f.close() |

附件1.12决策树，判定模型可靠程度（决策树.py）

|  |
| --- |
| from sklearn import metrics  from sklearn.tree import DecisionTreeClassifier  from sklearn import preprocessing  import numpy as np  # download the file  raw\_data = "dg.csv"  raw\_data1 = "test1.csv"  # load the CSV file as a numpy matrix  dataset = np.loadtxt(raw\_data, delimiter=",")  dataset1 = np.loadtxt(raw\_data1, delimiter=",")  # separate the data from the target attributes  X = dataset[:,0:3]  y = dataset[:,4]  z = dataset1[:,0:3]  ob=dataset1[:,4]  # normalize the data attributes  normalized\_X = preprocessing.normalize(X)  normalized\_z = preprocessing.normalize(z)  # standardize the data attributes  standardized\_X = preprocessing.scale(X)  standardized\_z = preprocessing.scale(z)  model = DecisionTreeClassifier()  model.fit(X, y)  print(model)  # make predictions  expected = y  predicted = model.predict(X)  predicted2 = model.predict(z)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted))  print(metrics.classification\_report(ob, predicted2))  print(metrics.confusion\_matrix(ob, predicted2))  c=0  for k in predicted2:  if k==1:  c+=1  print(c) |

附件1.13 KMeans算法分类会员信息（会员聚类.py）

|  |
| --- |
| from sklearn.cluster import KMeans  from sklearn.externals import joblib  import numpy  import time  import matplotlib.pyplot as plt  if \_\_name\_\_ == '\_\_main\_\_':  ## step 1: 加载数据  print("step 1: load data...")  dataSet = []  fileIn = open('zb2cl3.txt')  for line in fileIn.readlines():  lineArr = line.strip().split()  dataSet.append([float(lineArr[0]), float(lineArr[1])])  #设定不同k值以运算  for k in range(7,8):  clf = KMeans(n\_clusters=k) #就是调用KMeans算法  s = clf.fit(dataSet) #加载数据集合  numSamples=len(dataSet)  centroids = clf.labels\_  print(centroids,type(centroids)) #显示中心点  print(clf.inertia\_) #显示聚类效果  mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']  #画出所有样例点 属于同一分类的绘制同样的颜色  for i in range(numSamples):  #markIndex = int(clusterAssment[i, 0])  plt.plot(dataSet[i][0], dataSet[i][1], mark[clf.labels\_[i]]) #mark[markIndex])  f=open("聚类%d.txt"%(clf.labels\_[i]),"a")  f.write("%d"%(i)+"\n")  f.close()  mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']  # 画出质点，用特殊图型  centroids = clf.cluster\_centers\_  for i in range(k):  plt.plot(centroids[i][0], centroids[i][1], mark[i], markersize = 12)  #print centroids[i, 0], centroids[i, 1]  plt.show() |

附件1.14将价格，距离等用来判定的数据用DBSCAN打包处理（数据一聚类.py）

|  |
| --- |
| import numpy as np  from math import cos,pi  import matplotlib.pyplot as plt  from sklearn.cluster import DBSCAN  from sklearn import metrics  from sklearn.datasets.samples\_generator import make\_blobs  from sklearn.preprocessing import StandardScaler  l=list(open('ex1data4.txt'))  s=[]  x=[]  uop=[]  xx=[]  for i in l:  k=i.split()  k[0],k[1]=float(k[1]),float(k[0])  k[0]=k[0]/cos(pi\*(k[1]/180))  x.append(k[0:2])  xx.append(k[1])  uop.append(k[4])  #生产数据  X=np.array(x)  #计算  db = DBSCAN(eps=0.01,min\_samples=5).fit(X)  core\_samples\_mask = np.zeros\_like(db.labels\_,dtype=bool)  core\_samples\_mask[db.core\_sample\_indices\_] = True  labels = db.labels\_  n\_clusters\_ = len(set(labels))-(1 if -1 in labels else 0)  unique\_labels = set(labels)  colors = plt.cm.Spectral(np.linspace(0,1,len(unique\_labels)))  for k,col in zip(unique\_labels,colors):  if k == -1:  col = 'k'  class\_member\_mask = (labels == k)  xy = X[class\_member\_mask & core\_samples\_mask]  plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,  markeredgecolor='k', markersize=7)  xy = X[class\_member\_mask & ~core\_samples\_mask]  plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,  markeredgecolor='k', markersize=2)  s.append(list(xy))  #print(s)  for i in range(0,len(s)):  f=open(".\分类2\分类%d.txt"%(i),"w")  sty=[]  for k in s[i]:  k=list(k)  for b in range(0,len(k)):  ind=xx.index(k[1])  k[b]=str(k[b])  k.append(uop[ind])  sty.append("\t".join(k))  f.write("\n".join(sty))  f.close()  plt.title('Estimated number of clusters: %d' % n\_clusters\_)  plt.show() |

附件1.15将任务打包后的模型做判定（决策树-打包.py）

|  |
| --- |
| from sklearn import metrics  from sklearn.tree import DecisionTreeClassifier  from sklearn import preprocessing  import numpy as np  # download the file  raw\_data = "dg.csv"  raw\_data1 = "test1.csv"  # load the CSV file as a numpy matrix  dataset = np.loadtxt(raw\_data, delimiter=",")  dataset1 = np.loadtxt(raw\_data1, delimiter=",")  # separate the data from the target attributes  X = dataset[:,0:3]  y = dataset[:,4]  z = list(dataset1[:,0:3])  ob=dataset1[:,4]  numb=[]  for i in range(0,21):  l=list(open(".\分类2\分类%d.txt"%(i)))  if len(l)>=2 and len(l)<15:  po=[]  for j in range(0,len(l)):  k=l[j].split()  po.append(int(k[2]))  for cLo in range(1,len(po)):  z[po[0]][2]+=z[cLo][2]  z[po[0]][1]+=z[cLo][1]  del po[0]  numb.append(po[0])  else: continue  cC=0  for j in range(0,len(numb)):  del z[j]  cC+=1  if j<len(numb)-1:  numb[j+1]-=cC  z=np.array(z)  # normalize the data attributes  normalized\_X = preprocessing.normalize(X)  normalized\_z = preprocessing.normalize(z)  # standardize the data attributes  standardized\_X = preprocessing.scale(X)  standardized\_z = preprocessing.scale(z)  model = DecisionTreeClassifier()  model.fit(X, y)  print(model)  # make predictions  expected = y  predicted = model.predict(X)  predicted2 = model.predict(z)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted))  #print(metrics.classification\_report(ob, predicted2))  #print(metrics.confusion\_matrix(ob, predicted2))  c=0  for k in predicted2:  if k==1:  c+=1  print(c/len(predicted2)) |