ml-01

October 30, 2018

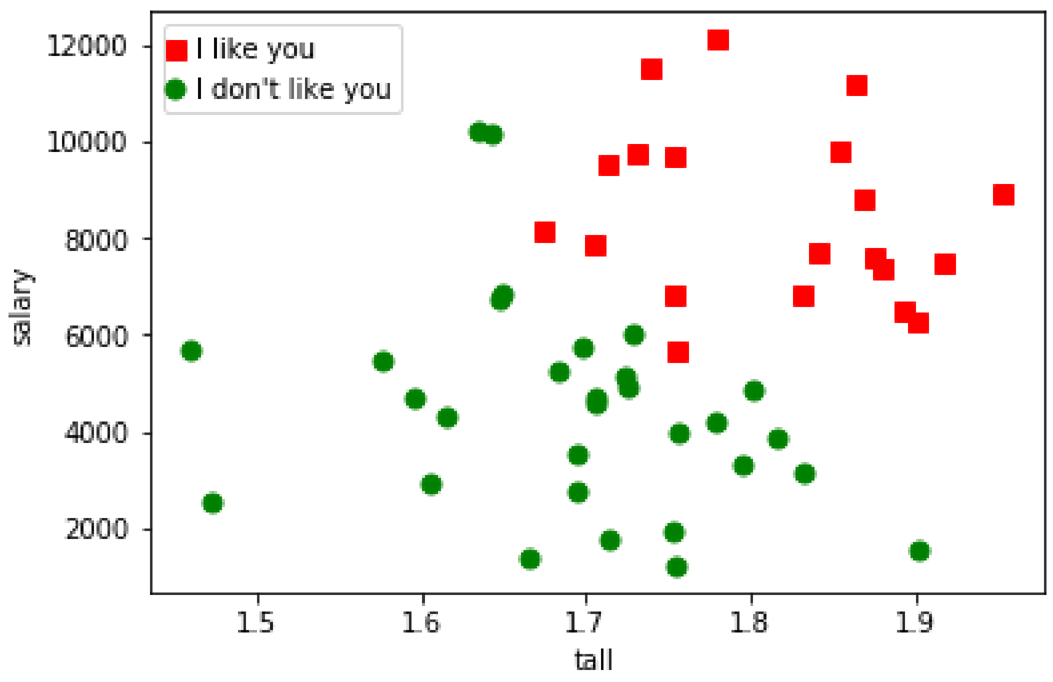
```
In [1]: # -*- coding: UTF-8 -*-
        #numpyhttps://www.jianshu.com/p/a260a8c43e44
        #matplotlibhttps://matplotlib.org/api/index.html

        import xlrd      #  xlrd  xlrd.open_workbook  excel
        import matplotlib.pyplot as plt
        #  matplotlib  matplotlib.pyplot  plt,  plt
        import numpy as np
        #  numpy  np,  np


        # loadData filename data.xlsnumpyarray
        def loadData(filename):
            workbook = xlrd.open_workbook(filename)          #  xlrd.open_workbook  excel  workbo
            boyinfo = workbook.sheet_by_index(0)             #  sheet_by_index   excel   sheet_by
            col_num = boyinfo.ncols                          #  ncols  excel    col_num
            row_num = boyinfo.nrows                          #  nrows  excel    row_num
            col0 = boyinfo.col_values(0)[1:]                 #  col_values(0)[1:]  excel  2 col0
            data = np.array(col0)                            #  np.array   col0  data
            if col_num == 1:                                 #   col_num 1  data
                return data                                      # data
            else:                                            #
                for i in range(col_num-1):                       # for
                    coltemp = boyinfo.col_values(i+1)[1:]            #
                    data = np.c_[data, coltemp]                      #  np.c_     data
            return data                                      # data

        #  plotData   X  flag: y plt, p1, p2
        def plotData(X, y):
            pos = np.where(y==1)
            #  np.where  y == 1  pos
            neg = np.where(y==0)
            #  np.where  y == 0  neg
            #  plt.plot  y == 1  s () square,  7   red
            p1 = plt.plot(X[pos, 0], X[pos, 1], marker='s', markersize=7, color='red')[0]
            #   plt.plot  y == 0  o () circle,  7   green
            p2 = plt.plot(X[neg, 0], X[neg, 1], marker='o', markersize=7, color='green')[0]  #
```

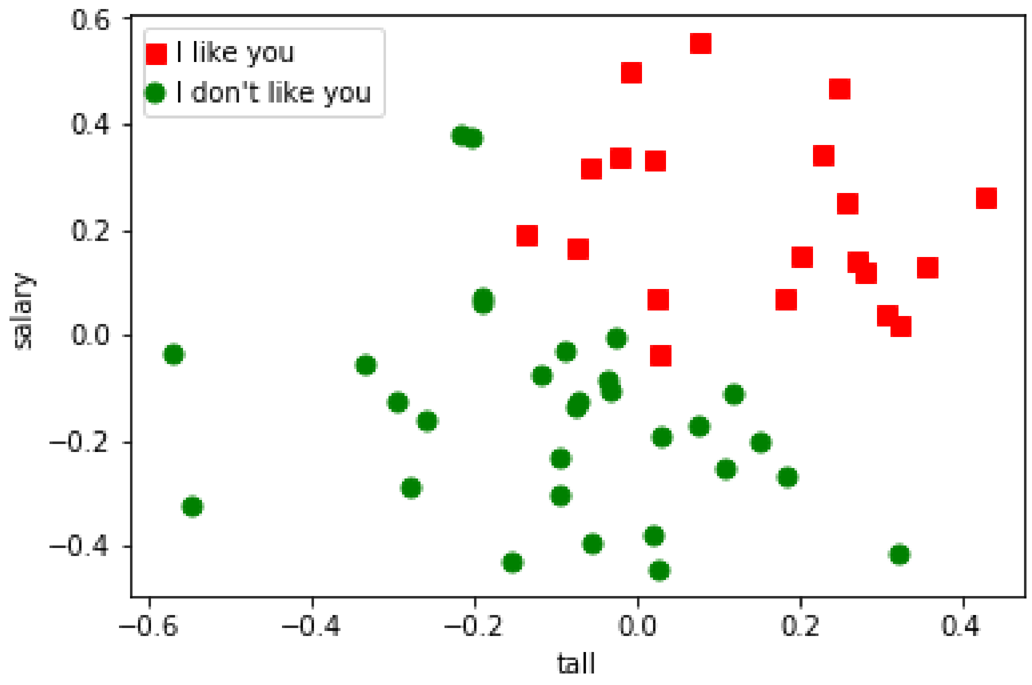
```
    return p1, p2                                   # plt, p1, p2

# normalization  normalization  X  X_norm   X_norm
def normalization(X):
    Xmin = np.min(X,axis=0)#
    #   np.min  axis=0  Xmin
    Xmax =np.max(X,axis=0) #
    #   np.max  axis=0  Xmax
    Xmu  =np.mean(X,axis=0) #
    #   np.mean  Xmu
    X_norm = (X-Xmu)/(Xmax-Xmin) #
    #  (X-Xmu)/(Xmax-Xmin) [-1,1]
    return X_norm  #  X_norm

# plot decision boundary plotDecisionBoundaryn   trainX,  trainY,  w,  iter_num
def plotDecisionBoundary(trainX, trainY, w, iter_num = 0):
    # prepare data
    xcord1 = [];
    ycord1 = [];
    xcord2 = [];
    ycord2 = []
    #  xcord1ycord1xcord2ycord2
    m, n = np.shape(trainX)
    #  np.shape  trainX m  trainX n  trainX
    for i in range(m):
        #  for  trainX i  012...m-1 m
        if trainY[i] == 1:
#  if  trainY 1 trainX trainX[i,1]  trainX[i,2]  xcord1  ycord1
            xcord1.append(trainX[i,1])
#  append  trainX  trainX[i,1]  xcord1  pos ,  positive
            ycord1.append(trainX[i,2])
#  append  trainX  trainX[i,2]  ycord1  pos ,  positive
        else:
        #  trainY 1 trainX trainX[i,1]  trainX[i,2]  xcord2  ycord2
            xcord2.append(trainX[i,1])
            #  append  trainX  trainX[i,1]  xcord2  neg ,  negative
            ycord2.append(trainX[i,2])
            #  append  trainX  trainX[i,2]  ycord2  neg ,  negative
    x_min = min(trainX[:,1])                        #  min  trainX[:,1]  trainX 2 x_min
    y_min = min(trainX[:,2])                        #  min  trainX[:,2]  trainX 3 y_min
    x_max = max(trainX[:,1])                        #  max  trainX[:,1]  trainX 2 x_max
    y_max = max(trainX[:,2])                        #  max  trainX[:,2]  trainX 3 y_max

    # plot scatter  & legend
    fig = plt.figure(1)                              #  plt.figure  fig
    #  plt.scatter  xcord1,  ycord130 s () square,  'I like you'
    plt.scatter(xcord1, ycord1, s=30, c='red', marker='s', label='I like you')
    plt.scatter(xcord2,ycord2,s=30,c='green',marker='o',label="I don't like you")
    #   plt.scatter  xcord2,  ycord230 o () circle,  'I don't like you'
    #
    plt.legend(loc='upper right')                          #
      # set axis and ticks
    delta_x = x_max-x_min                                  #  delta_x
    delta_y = y_max-y_min                                  #  delta_y
    #  x_min - delta_x / 10  x_max + delta_x / 10 np.arange  1 my_x_ticks
    my_x_ticks = np.arange(x_min - delta_x / 10, x_max + delta_x / 10, 1)
    #  y_min - delta_y / 10  y_max + delta_y / 10 np.arange  1 my_y_ticks
    my_y_ticks = np.arange(y_min - delta_y / 10, y_max + delta_y / 10, 1)

    plt.xticks(my_x_ticks)                                 #  plt.xticks  my_x_ticks
    plt.yticks(my_y_ticks)                                 #  plt.yticks  my_y_ticks
    #  plt.axis [x_min-delta_x/10, x_max+delta_x/10]  [y_min-delta_y/10, y_max+delta_y/1
    plt.axis([x_min-delta_x/10, x_max+delta_x/10, y_min-delta_y/10, y_max+delta_y/10])

    # drwa a line
    x = np.arange(x_min-delta_x/10, x_max+delta_x/10, 0.01) #  np.arange   x_min - delta
    y = (-w[0]-w[1]*x)/w[2]#                                                #  y = (-w[0]-w
    plt.plot(x, y.T)                                       #  plt.plot  x ,  y.T .T

    # figure name
    #  'Training ' + str(iter_num) + ' times.png'str(iter_num)  iter_num  png
    fig_name = 'Training ' + str(iter_num) + ' times.png'
    # 'Training ' + str(iter_num) + ' times.png'str(iter_num)  iter_num  png
    plt.title(fig_name)
    fig.savefig(fig_name)                                  #  fig.savefig
    plt.show(fig)                                            #  plt.show
    plt.close()
# sigmoid:   sigmoid  activation function wx  sigmoid
def sigmoid(wx):
    sigmoidV =1.0/(1.0+np.exp(-wx))  #                           #    sigmoid  1.0/(1.
    return sigmoidV

# loss fuc Y_   Y
def loss(X, Y, w):
    #   loss  loss function X, Y, w
    m, n = np.shape(X)
    #  np.shape  X m  X n  X
    trainMat = np.mat(X)
    #  np.mat  X  trainMat
    Y_ = []                                                #  Y_,  append
    for i in np.arange(m):                                 #  for  X i 01 2....m-1  X m
        #  append  Y_   trainMat[i]  w  sigmoid
        Y_.append(sigmoid(trainMat[i]*w))
    m = np.shape(Y_)[0]                                    #  np.shape  X np.shape(Y_)[0]  X  m
    sum_err = 0.0                                          #  0.0,  sum_err   sum_err
    for i in range(m):                                     #  for  Y_  i 01 2....m-1  Y_  m
```

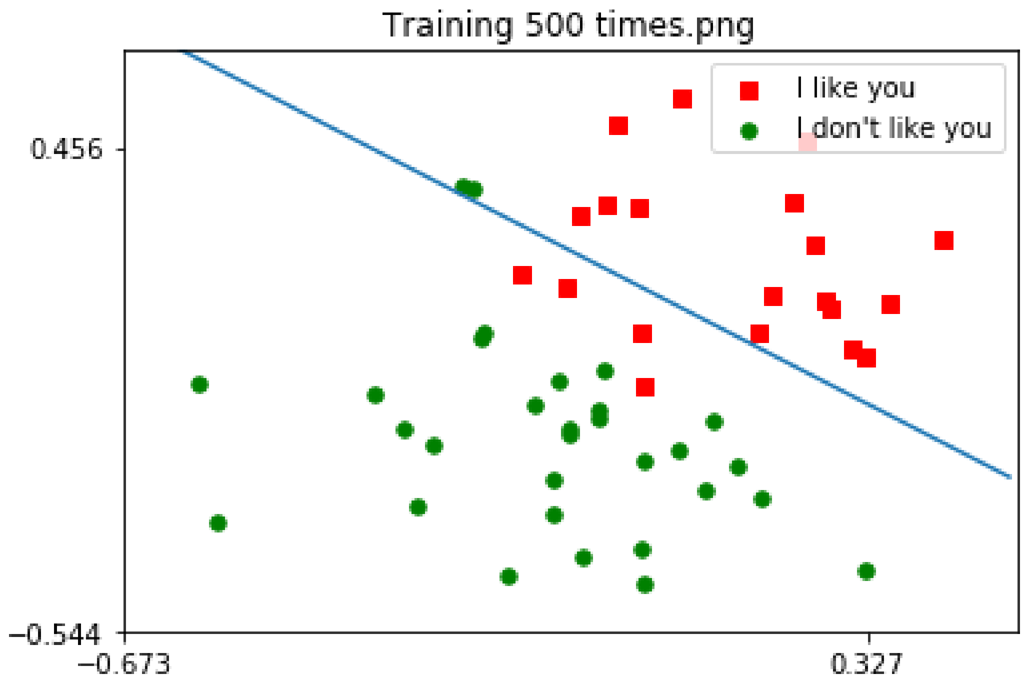
```
        #   sum_err    sum_err  Y[i]*np.log(Y_[i])+(1-Y[i])*np.log(1-Y_[i])  Cross Entro
        sum_err -= Y[i]*np.log(Y_[i])+(1-Y[i])*np.log(1-Y_[i])  #
    return sum_err/m                                    #  sum_err

# BGD
# BGD Batch Gradient DescentBGD  X  y,
#  iter_num,  alpha lr (learning rate), J   w
# Batch Gradient DescentBGD W
def BGD(X, y, iter_num, alpha):
    trainMat = np.mat(X)                                #  np.mat  X  trainMat
    trainY = np.mat(y).T                                #  np.mat  y  trainY
    m, n = np.shape(X)                                  #  np.shape  X m  X n  X
    w = np.ones((n,1))                                  #  np.ones  1  n  1  w,  w  1
    for i in range(iter_num):                           #  for i 01 2....iter_num-1  iter_nu
        error = sigmoid(trainMat*w)-trainY   #                           #  error sigmoi
        w =w - (1.0/m)*alpha*trainMat.T*error       #                            #  w , E
    return w                                            #  w

# classify classify  wx  1  0
def classify(wx):
    prob = sigmoid(wx)                                  #  sigmoid(wx)  prob
    if prob > 0.5:                                      #   prob  0.5  1
        return 1
    else:                                               #   prob  0.5  0
        return 0

# predict predict   testX  w  result
def predict(testX, w):
    m, n = np.shape(testX)                              #  np.shape  testX m  testX n  testX
    testMat = np.mat(testX)                             #  np.mat  testX  testMat
    result = []                                         #  result,  append
    for i in np.arange(m):                              #  for  testX i 01 2....m-1  testX
        #  append  result  classify  1  0   result
        result.append(classify(float(testMat[i]*w)))
    return result                                       # result

# Precision Precision  X, Y  w
def Precision(X, Y, w):
    result = predict(X, w)                              #  predict  X  w  result
    right_sum = 0                                       #  0  right_sum 1
    #  for i 01 2....len(result)-1  result  len(result)
    for i in range(len(result)):
        if result[i]-int(Y[i]) == 0:                    #  if,  result  int(Y[i])  right_sum
            right_sum += 1                              #   right_sum 1
    # 1.0*right_sum/len(Y) 1.0  float
    return 1.0*right_sum/len(Y)

# python
```

```
if __name__ == "__main__":
    # load data and visualization
    data = loadData('data.xls')                           #  loadData   'data.xls' data
    X = data[:,:2]                                        #  data     ,  m
    y = data[:,2]                                         #  data    y Y=1/N=0 0  1

    # plot data
    plt_data = plt.figure(1)
    p1, p2 = plotData(X, y)                          #  plotData   X  y

    #Labels and Legend
    plt.xlabel('tall')                                    #  plt.xlabel 'tall' m
    plt.ylabel('salary')                                  #  plt.ylabel 'salary'
    #  plt.legend  'I like you'  "I don't like you"
    #   numpoints  1 handlelength 0
    plt.legend((p1, p2), ('I like you', "I don't like you"), numpoints=1, handlelength=0

    # show and save visualized image
    plt_data.savefig('visualization_org.jpg')      #  plt.savefig 'visualization_org.jpg'
    plt.show(plt_data)                              #  plt.show
    plt.close(plt_data)                             #  plt.close

    # normalization and visualization normalization  X
    X_norm = normalization(X)
    # plot data
    plt_norm = plt.figure(1)
    #  plotData   X_norm  y plt_norm, p1_norm  p2_norm
    p1_norm, p2_norm = plotData(X_norm, y)

    # Labels and Legend
    plt.xlabel('tall')                               #  plt.xlabel 'tall' m
    plt.ylabel('salary')                             #  plt.ylabel 'salary'
    #  plt.legend  'I like you'  "I don't like you"
    #   numpoints  1 handlelength 0
    plt.legend((p1_norm, p2_norm), ('I like you', "I don't like you"), numpoints=1, hand

    # show and save visualized image
    #  plt.show
       #
    plt.show(plt_norm)
    #  plt.savefig 'visualization_norm.jpg' 'jpg'
       #
    figname='visualization_norm.jpg'
    plt.savefig(figname)
    #  plt.close
       #
    plt.close()
```

```
# optimizing by BSD
iter_num=500                                        #  iter_num 200  iter_num
lr=0.05                                              #  lr 0.001  lr
m,n = np.shape(data)                                  #  np.shape  data m  data n  data
offset = np.ones((m, 1))                              #  np.ones  1  m  1  offset,  offset
trainMat = np.c_[offset, X_norm]                      #  np.c_  offset   X_norm  trainMat
theta=BGD(trainMat,y,iter_num,lr)                     #  BGD Batch Gradient DescentBGD  th

## Plot Boundary
#  plotDecisionBoundary  trainMat,  y,  theta   iter_num
plotDecisionBoundary(trainMat, y, theta, iter_num)
cost = loss(trainMat, y, theta)                       #  loss ,   trainMat,  y   theta  co
print('Cost theta: {0}'.format(cost))                 #  .format(cost)  %s

# Compute accuracy on our training set
p = Precision(trainMat, y, theta)                     #  Precision   trainMat,  y   theta
print('Train Accuracy: {0}'.format(p))                #  .format(p)  %s
print('finished!')                                    # 'finished!'
```







```
Cost theta: [[0.42092422]]
Train Accuracy: 0.88
```

```
finished!
```

```
In [ ]:
```