Homework Assignment 3

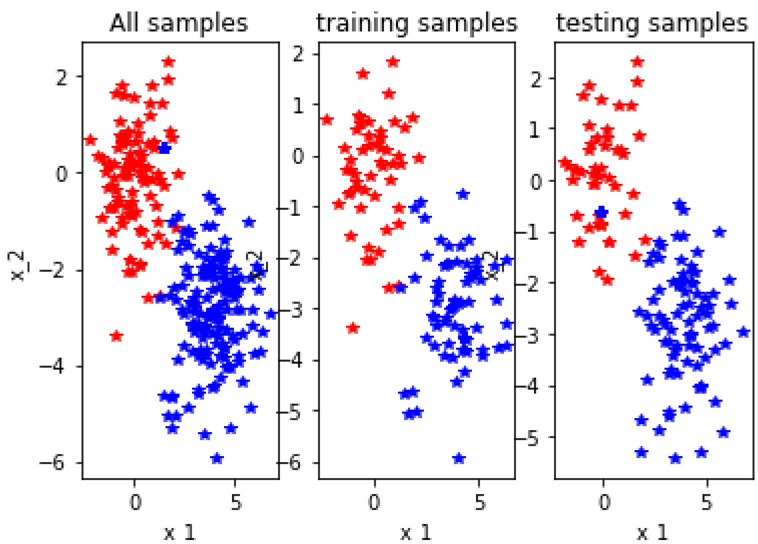
Xinyuan Yang

UID:305255240

Problem I. Logistic Regression

In [45]:
```
import numpy as np
import matplotlib.pyplot as plt
from getDataset import getDataSet
from sklearn.linear_model import LogisticRegression


# Starting codes

# Fill in the codes between "%PLACEHOLDER#start" and "PLACEHOLDER#end"

# step 1: generate dataset that includes both positive and negative samp
les,
# where each sample is described with two features.
# 250 samples in total.

[X, y] = getDataSet()  # note that y contains only 1s and 0s,

# create figure for all charts to be placed on so can be viewed together
fig = plt.figure()


def func_DisplayData(dataSamplesX, dataSamplesY, chartNum, titleMessage
):
    idx1 = (dataSamplesY == 0).nonzero()  # object indices for the 1st c
lass
    idx2 = (dataSamplesY == 1).nonzero()
    ax = fig.add_subplot(1, 3, chartNum)
    # no more variables are needed
    plt.plot(dataSamplesX[idx1, 0], dataSamplesX[idx1, 1], 'r*')
    plt.plot(dataSamplesX[idx2, 0], dataSamplesX[idx2, 1], 'b*')
    # axis tight
    ax.set_xlabel('x_1')
    ax.set_ylabel('x_2')
    ax.set_title(titleMessage)
# plotting all samples
func_DisplayData(X, y, 1, 'All samples')

# number of training samples
nTrain = 120

######################PLACEHOLDER 1#start#########################
# write you own code to randomly pick up nTrain number of samples for tr
aining and use the rest for testing.
# WARNIN:

maxIndex = len(X)
RandomTrainingData = np.random.choice(maxIndex, nTrain, replace=False)
RandomTestingData = [i for i in range(maxIndex)if i not in RandomTrainin
gData]

trainX = X[RandomTrainingData,:]  # training samples
trainY = y[RandomTrainingData,:]  # labels of training samples  nTrain X
1

testX =  X[RandomTestingData,:]  # testing samples
testY =  y[RandomTestingData,:]  # labels of testing samples  nTest X 1
```

```
###################PLACEHOLDER 1#end########################

# plot the samples you have pickup for training, check to confirm that b
oth negative
# and positive samples are included.
func_DisplayData(trainX, trainY, 2, 'training samples')
func_DisplayData(testX, testY, 3, 'testing samples')

# show all charts
plt.show()
```



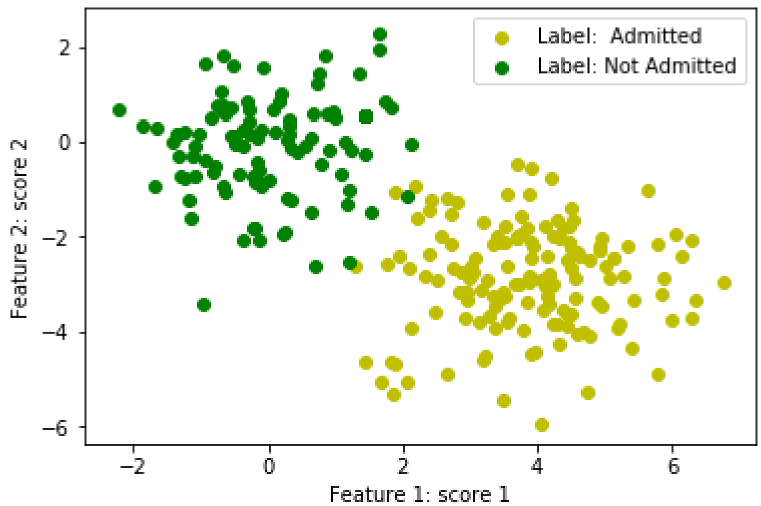
Through step 1, we generate the provided data and split it into training and testing subsets. In this case, we use random 120 samples as training data, and use the left 130 samples as testing data.

Then, the code will display the splitting results of all samples, traning samples and testing samples. From the plot, we can see that the distributions of features are similiar in the three pictures.

In [50]:
```
#  step 2: train logistic regression models

#####################PLACEHOLDER2 #start########################
# in this placefolder you will need to train a logistic model using the
 training data: trainX, and trainY.
# please delete these coding lines and use the sample codes provided in
 the folder "codeLogit"
# logReg = LogisticRegression(fit_intercept=True, C=1e15) # create a mod
el
# logReg.fit(trainX, trainY)# training
# coeffs = logReg.coef_ # coefficients
# intercept = logReg.intercept_ # bias
# bHat = np.hstack((np.array([intercept]), coeffs))# model parameters

clf = LogisticRegression()

# utilize the function fit() to train the class samples
clf.fit(trainX,trainY)

# scores over testing samples

# print the fearure distribution plot of data using functions in the lib
rary pylab
from pylab import scatter, show, legend, xlabel, ylabel
positive = np.where(y == 1)
negative = np.where(y == 0)
scatter(X[positive, 0], X[positive, 1], c='y')
scatter(X[negative, 0], X[negative, 1], c='g')
xlabel('Feature 1: score 1')
ylabel('Feature 2: score 2')
legend(['Label:  Admitted', 'Label: Not Admitted'])
show()


theta = [0,0] #initial model parameters
alpha = 0.1 # learning rates
max_iteration = 1000 # maximal iterations


from util import Cost_Function, Gradient_Descent, Cost_Function_Derivati
ve, Cost_Function, Prediction, Sigmoid

theta = [0,0] #initial model parameters
alpha = 0.1 # learning rates
max_iteration = 1000 # maximal iterations

m = len(y) # number of samples

for x in range(max_iteration):
    # call the functions for gradient descent method
    new_theta = Gradient_Descent(X,y,theta,m,alpha)
    theta = new_theta
    if x % 200 == 0:
        # calculate the cost function with the present theta
        Cost_Function(X,y,theta,m)
        print('theta is ', theta)
```

```
        print('cost is ', Cost_Function(X,y,theta,m))

score = 0
for i in range(len(testX)):
    prediction = round(Prediction(testX[i],theta))
    answer = testY[i]
    if prediction == answer:
        score += 1

gdScore = float(score) / float(len(testX))


print('Coefficients from sklearn:', clf.coef_)
print('Coefficients from gradient descent:',theta)
print('Score from sklearn: ', clf.score(testX,testY))
print('Score from gradient descent: ', gdScore)
######################PLACEHOLDER2 #end #########################
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:761:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```




```
theta  [array([0.11671255]), array([-0.08379834])]
cost is  [0.52449911]
theta  [array([0.77303262]), array([-0.29445986])]
cost is  [0.32582141]
theta  [array([0.78820085]), array([-0.27878733])]
cost is  [0.32578743]
theta  [array([0.78959445]), array([-0.27735696])]
cost is  [0.32578714]
theta  [array([0.78972252]), array([-0.27722573])]
cost is  [0.32578714]
Coefficients from sklearn: [[ 1.45032831 -0.6900035 ]]
Coefficients from gradient descent: [array([0.78973427]), array([-0.277
21369])]
Score from sklearn:  0.9846153846153847
Score from gradient descent:  0.823076923076923
```

The second step is to train a logistic regression model using the 120 training samples. To do so, we use the functions in the folder ‘codeLogit’ and there are two different implementations, that is, sklearn and gradient descent.

The accurancy in this case is 0.9, so we can conclude that with the used model, the accuancy of the test dataset is considerably high.

Additionally, we can see that there is a decreasing trend of theta, it even goes from positive to negative, and the output of the cost function decreases as well.

Lastly, coparing the sklearn model and gradient descent model, the score for sklearn model is 0.9846153846153847 and the score for gradient descent model is 0.823076923076923. The difference here is due to the shorages in Gradient Descent Model, it has a strict rule for learning rate, it is relatively sensitive.

In [34]:

```
# step 3: Use the model to get class labels of testing samples.

######################PLACEHOLDER3 #start#########################
# codes for making prediction,
# with the learned model, apply the logistic model over testing samples
# hatProb is the probability of belonging to the class 1.
# y = 1/(1+exp(-Xb))
# yHat = 1./(1+exp( -[ones( size(X,1),1 ), X] * bHat )); ));
# WARNING: please DELETE THE FOLLOWING CODEING LINES and write your own
 codes for making predictions
# xHat = np.concatenate((np.ones((testX.shape[0], 1)), testX), axis=1)
 # add column of 1s to left most  ->  130 X 3
# negXHat = np.negative(xHat)  # -1 multiplied by matrix -> still 130 X
 3
# hatProb = 1.0 / (1.0 + np.exp(negXHat * bHat))  # variant of classific
ation   -> 130 X 3
# predict the class labels with a threshold
# yHat = (hatProb >= 0.5).astype(int)  # convert bool (True/False) to in
t (1/0)
# PLACEHOLDER#end

#print('score Scikit learn: ', clf.score(testX,testY))
#this is the prediction using gradient decent
yHat = [Prediction(row,theta) for row in testX]
yHat = np.array([float(int(val >= .6)) for val in yHat])
#yHat

yHatSk = clf.predict(testX)
#yHatSk
#testY
#len(yHatSk)
#len(testY)
######################PLACEHOLDER 3 #end #########################
```

The third step is to apply the learned model to get the binary classes of testing samples. This step is modified according to the implementation of the second step.

In [54]:
```
# step 4: evaluation
# compare predictions yHat and and true labels testy to calculate averag
e error and standard deviation
testYDiff = np.abs(yHat - testY)
avgErr = np.mean(testYDiff)
stdErr = np.std(testYDiff)

print('average error of the Gradient decent model: {} ({})'.format(avgEr
r, stdErr))
score = 0
winner = ""
# accuracy for sklearn
scikit_score = clf.score(testX,testY)
length = len(testX)
for i in range(length):
    prediction = round(Prediction(testX[i],theta))
    answer = testY[i]
    if prediction == answer:
        score += 1

my_score = float(score) / float(length)
if my_score > scikit_score:
    print('You won!')
elif my_score == scikit_score:
    print('Its a tie!')
else:
    print('Scikit won.')
print('Your score: ', my_score)
print('Scikits score: ', scikit_score)
```

```
average error of the Gradient decent model: 0.44769230769230767 (0.4972
5637786301324)
Scikit won.
Your score:  0.823076923076923
Scikits score:  0.9846153846153847
```

The fourth step is to compare the predictions with the ground-truth labels and calculate average errors and standard deviation.

Problem II. Confusion matrix

Confusion Matrix:

```
              Cat          Dog            Monkey
```

Cat 1 3 1 Dog 3 3 2 Monkey 2 2 3

Accuracy: (1+3+3)/20=0.35

For Cat: Precision: 1/(1+3+2)=0.167 Recall:1/(1+3+1)=0.2

For Dog: Precision:3/(3+3+2)=0.375 Recall:3/(3+3+2)=0.375

For Monkey: Precision:3/(1+2+3)=0.5 Recall:3/(3+2+2)0.429

Problem III. Comparative Studies!

In [55]:
```
#this function returns the accuracy and a precision/recall array.
#included in each row of the precision recall array is:
# 0 - the value
# 1 - the precision
# 2 - the recall
def func_calConfusionMatrix(predY,trueY):
    print("Confusion Matrix:")
    for pred1 in np.unique(predY):
        print(int(pred1), end="    ")
        for pred2 in np.unique(predY):

            correctCount = 0
            for i in range(len(predY)):
                if(predY[i] == pred1 and trueY[i] == pred2):
                    correctCount += 1
            print(correctCount, end=" ")
        print()
    #accuacy
    correctCount = 0
    for index in range(len(trueY)):
        if(trueY[index] == predY[index]):
            correctCount += 1
    #print(correctCount)
    accuracy = correctCount / len(trueY)

    precRec = []
    for pred in np.unique(trueY):
        pred = int(pred)
        #print(pred)
        #precision
        correctCount = 0
        for i in range(len(trueY)):
            if(int(trueY[i]) == int(predY[i]) and int(trueY[i]) == pred
):
                correctCount += 1
        #print(correctCount)
        #print(len(predY[predY == pred]))
        prec = correctCount / len(predY[predY == pred])

        #recall
        correctCount = 0
        for i in range(len(trueY)):
            if(trueY[i] == predY[i] and int(trueY[i]) == pred):
                correctCount += 1
        rec = correctCount / len(trueY[trueY == pred])
        #print(len(trueY[trueY == pred]))
        #print(rec)
        precRec.append([pred,prec,rec])

    return accuracy, precRec


values = func_calConfusionMatrix(yHatSk,testY)
print('Accurracy, Precision, and Recall for our SkLearn model:')
print(values)
print()
```

```
values = func_calConfusionMatrix(yHat,testY)
print('Accurracy, Precision, and Recall for our gradient decent model:')
print(values)
```

```
Confusion Matrix:
0   44 2
1   4 80
Accurracy, Precision, and Recall for our SkLearn model:
(0.9538461538461539, [[0, 0.9565217391304348, 0.9166666666666666], [1,
0.9523809523809523, 0.975609756097561]])

Confusion Matrix:
0   37 2
1   11 80
Accurracy, Precision, and Recall for our gradient decent model:
(0.9, [[0, 0.9487179487179487, 0.7708333333333334], [1, 0.8791208791208
791, 0.975609756097561]])
```