Homework Assignment 3

Due: 11:59pm, October 25, 2024

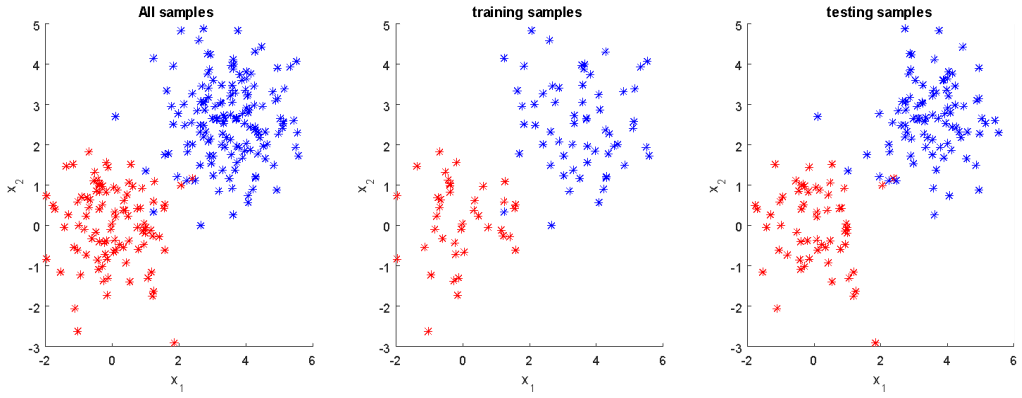
This assignment includes three problems for practicing logistic regression methods and evaluation metrics.

Problem I. Logistic Regression

Overview. The goal of this mini-project is to predict the binary class label for any sample described with two features. You will be instructed to use the Logistic Regression (LR) method for solving a simulated problem.

Sample Codes. The file "main_part1.py" provides sample codes to start with. There are four major steps: generating dataset, training LR model, testing, and evaluating the prediction results.

The **first** step is to generate data (provided) and split it into training and testing subsets. You will need to write your codes to do the split. Then, the code will display the splitting results as the following figures:



The **second** step is to train a logistic regression model using the training data. To do so, you will need to use the functions we provided in the folder 'codeLogit'. Remember there are two different implementations. **Please try both methods in this place folder and report their performance differences.**

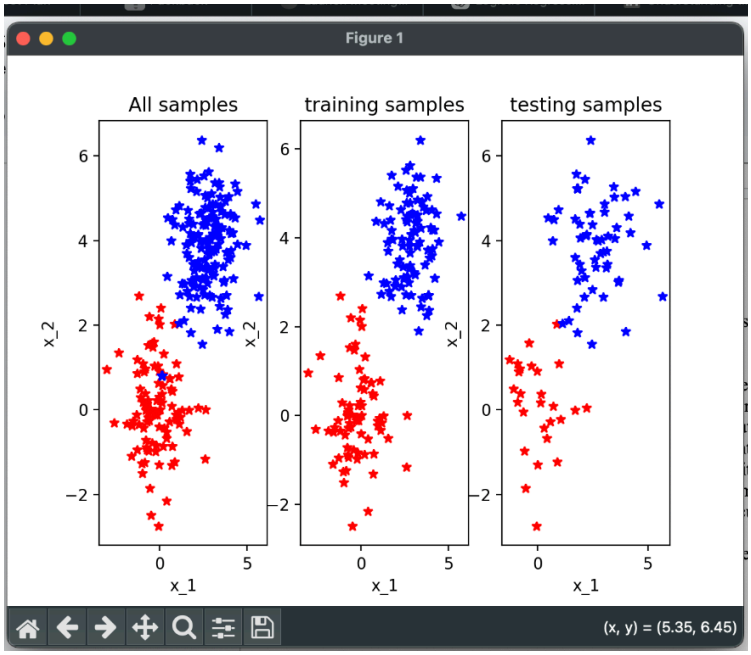
The **third** step is to apply the learned model to get the binary classes of testing samples. This step should be modified according to the implementation of the second step.

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

You will need to replace the PLACEHOLDERs with your codes for splitting datasets (step 1) , training (step 2) , and testing (step 3). While there is no need to change the step 4, you are encouraged to implement your own ways.

In your report, please include both figures of sample scatters and quantitative results of your implementation.

Since I did Quiz02, I just did the assignment the same way I did there. I copy, pasted the functions that I used and the assignment was rather simple after that. I did have a little bit of trouble with step 3 in preparing the data for step 4, but the solution was to concatenate each prediction to hatProb one at a time. I used the gradient descent with an intercept because quiz two indicated that it was more accurate than without it. My custom model manages about the same accuracy and standard deviation as the sklearn model. I also used the sklearn train_test_split function to split my data, but a more custom solution can be found in main_sklearn.py file.
Here is a complete run of the custom model, complete with screenshots and the data:



Average error: 0.3253 (Standard Deviation: 0.4685)

Over 10 iterations.
I found that the custom model was slightly, but measurably less accurate than the sklearn model. However, both of the model's Standard deviations were almost identical.

Cust_AvgErr	Cust_Std	Sklearn_AvgErr	Aklearn_Std
0.3255	0.4685	0.4667	0.4989
0.3855	0.4867	0.3359	0.4723
0.3712	0.4831	0.3256	0.4686
0.4217	0.4938	0.3231	0.4677
0.3614	0.4804	0.3419	0.4741

0.4863	0.4998	0.4077	0.4914
0.3948	0.4888	0.3282	0.4696
0.3922	0.4882	0.3564	0.4789
0.3938	0.4886	0.3462	0.4757
0.3253	0.4997	0.3487	0.4766

Problem II. Confusion matrix

Suppose that there is a trained classifier for predicting the animal classes (e.g., cat, dog) of a photo. The following table lists the prediction class and ground-truth class for each test image.

Image ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
True class	C	C	C	C	C	D	D	D	D	D	D	D	D	M	M	M	M	M	M	M
Predicted class	D	C	D	D	M	D	D	C	C	M	M	D	C	C	C	M	M	D	D	M

Notes: C, cat; D, dog; M, monkey

Please manually compute and report the confusion matrix and accuracy. For each of the three categories, calculate its precision and recall rates.

TP = True Positive: Correctly Identified
FP = False Positive: Incorrectly Identified
FN = False Negative: Misidentified

	Predicted C	Predicted D	Predicted M
True C	TP	FP	FN
True D	FP	TP	FN
True M	FN	FP	TP

Confusion Matrix:

	Predicted C	Predicted D	Predicted M
True C	1	3	1
True D	3	3	2
True M	2	2	3

Accuracy = numCorrect / numAttempts = 7 / 20 = .35 = 35%
Precision = TP/(TP + FP)

Recall = TP/(TP + FN)

Precision Dog = 3/5= 60%, Recall Dog = 3/6 = 50%
Precision Cat =1/4 = 25%, Recall Cat=1/2=50%
Precision Monkey=3/5=60%, Recall Monkey=3/5=60%

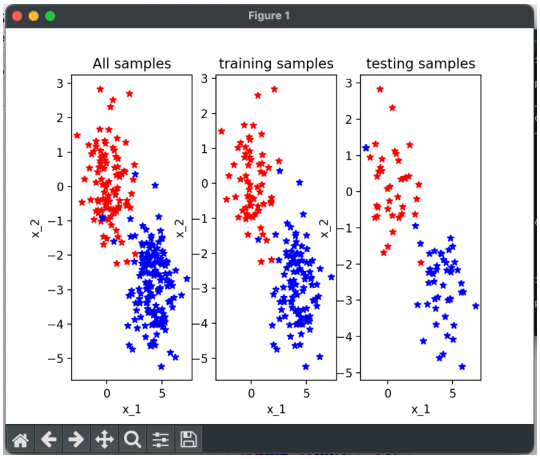
Problem III. Comparative Studies

Please write a function to calculate the confusion matrix for the prediction results of a classifier. This function should take the form:

def func_calConfusionMatrix(predY, trueY)

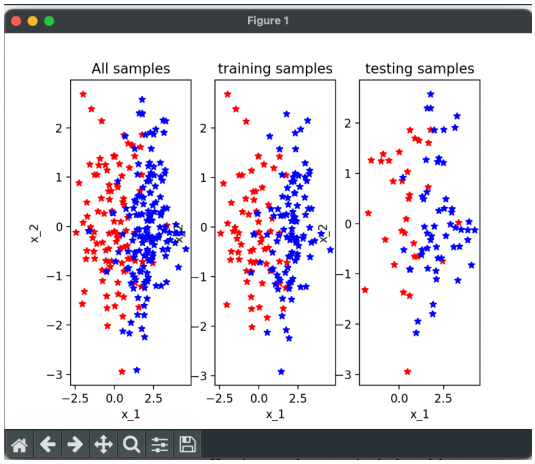
where predY is the vector of the predicted labels and trueY is the vector of true labels. This function should return accuracy, per-class precision, and per-class recall rate.

Please use above function in the script "main_part1.py", and report the confusion matrix of both logistic regression implementations.
Here is a complete run using both models:
Custom:



Confmatrix:
[[TP,FP]
[FN,TN]]
[[17,6]
[28,32]]
Average error: 0.5188 (Standard Deviation: 0.4996)
Accuracy: 0.5903614457831325
Precision: 0.7391304347826086
Recall: 0.37777777777777777

Sklearn:



Confmatrix:
[[TP,FP]
[FN,TN]]
[[26,13]
[25,19]]
average error: 0.43373493975903615 (0.4955894891856176)
Accuracy: 0.5421686746987951
Precision: 0.6666666666666666
Recall: 0.5098039215686274

I did this two ways, one using Sklearn and one using a completely custom implementation. Here are both:

```
def func_calConfusionMatrix(predY,trueY):
    TP = TN = FP = FN = 0

    for pred, true in zip(predY, trueY):
        if pred == 1 and true == 1:
            TP += 1
        elif pred == 0 and true == 0:
            TN+=1
        elif pred == 1 and true == 0:
            FP += 1
        elif pred == 0 and true == 1:
            FN+=1

print("Confmatrix:\n[[TP,FP]\n[FN,TN]]\n[[{:},{:}]\n[{:},{:}]]".format(TP,FP,FN,TN))

    accuracy = (TP + TN) / (TP + TN + FP + FN)
```

```
    precision = TP / (TP + FP) if (TP + FP) != 0 else 0
    recall = TP / (TP + FN) if (TP + FN) != 0 else 0

    return  accuracy, precision, recall
```

```
def calConfustionMatrix(predY,trueY):
    labels = [0,1]

    cMatrix = confusion_matrix(trueY,predY,labels=labels)

    accuracy = accuracy_score(trueY,predY)
    precision = precision_score(trueY,predY,labels=labels,average=None)
    recall = recall_score(trueY,predY,labels=labels,average=None)

    return cMatrix,accuracy,precision,recall
```