Damandeep Singh

Professor Liu

CS 549

4 April 2022

HA3 Report

**Problem I.**

Placeholder 1:

In the first placeholder, I have split the dataset using the provided combination of 48/52 for train/test using the np.random.choice() method. I created a randomized array with a size equal to the array 'X' and used the slicing technique to pick the "nTrain" number of samples for training and the rest for testing purposes.

```
# length of X array
maxIndex = len(X)

# Create a random array of size maxIndex (250)
randomTrainingSamples = np.random.choice(maxIndex, maxIndex, replace = False)

# Split data ratio = 48/52 (train/test) using nTrain
pickTraining = randomTrainingSamples[:nTrain] # 48% for training

pickTesting = randomTrainingSamples[nTrain:] # 52% for testing

trainX =  X[pickTraining]   # training samples
trainY =  y[pickTraining] # labels of training samples    nTrain X 1

testX =   X[pickTesting] # testing samples
testY =  y[pickTesting] # labels of testing samples     nTest X 1
```

Figure 1. Splitting Data

Placeholder 2:

I have trained two models in this placeholder: Logistic Regression (Sklearn) and Gradient Descent, using the trainX and trainY variables. For the sklearn logistic regression method, I followed the provided code, replaced the inputs with training samples, and called the fit() method to train the model, and the output depends on the placeholder one for splitting the data and running a scattered plot showing two classes. On the other hand, the self-developed gradient descent model is trained using the trainX and trainY arrays while keeping other factors such as theta and alpha as constants. Figure 2 shows the approach used for both sklearn method and the self-developed gradient descent method. The gradient descent method is commented to let the sklearn method work with other functions and generate the plots.

```
'''First LR Method: Sklearn learn training'''
# training
model = LogisticRegression(fit_intercept=True, C=1e15).fit(trainX, trainY)

# performance
perform = model.score(testX, testY)
print("Performance: ", perform)

''' Second Method: Gradient Descent training '''
from codeLogit.main_logit import *

# m = len(trainX) # number of samples
# cst = [] # store theta
# cost = [] #cost function
# for x in range(250):
#     new_theta = Gradient_Descent(trainX, trainY,theta,m,alpha)
#     theta = new_theta
#     cst.append(theta)
#     print("Cost function: ", Cost_Function(trainX, trainY,theta,m))
```

Figure 2. Training Both Models

Observations:

Sklearn LR method:

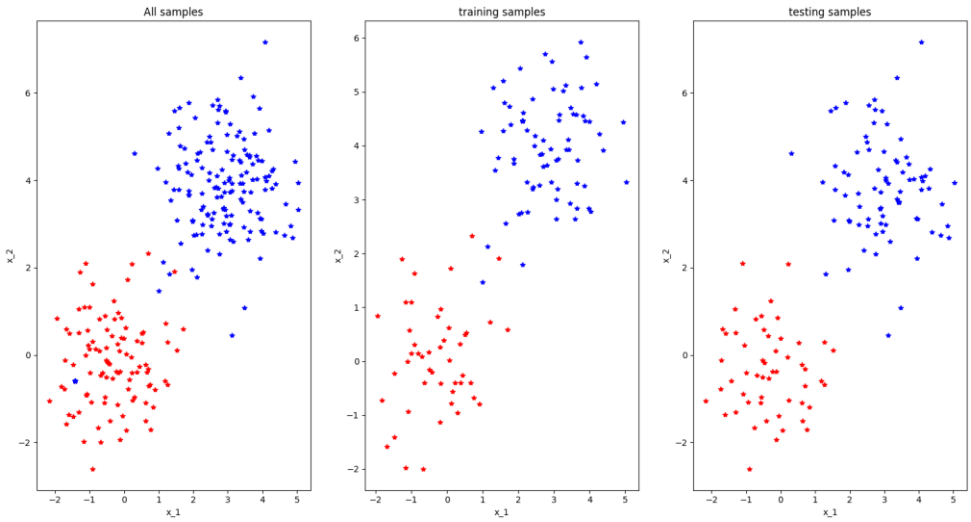


Figure 3. Results of Sklearn LR Method

In figure 3, the first graph with the "all samples" title displays the distribution of dataset samples without training scattered into two parts. A few outliers or data points are mixed with red and blue data points. However, most data points are part of their cluster and exhibit accuracy through their graph position.

The second graph with the "training samples" displays a reduced amount of data used to train the model using the 48/52 combination of train/test split.

The performance of the sklearn logistic regression was better as tested with the score() method, with the value of accuracy being as follows:

```
perform = model.score(testX, testY)
print("Performance: ", perform)
```

```
Performance:  0.9461538461538461
```

The performance figure shows that against the testing sets, the accuracy of the sklearn method for a given training set is about 94.6%. The accuracy percentage is correct because a few data points are mixed, generating inaccuracies in the data.

Gradient Descent method:

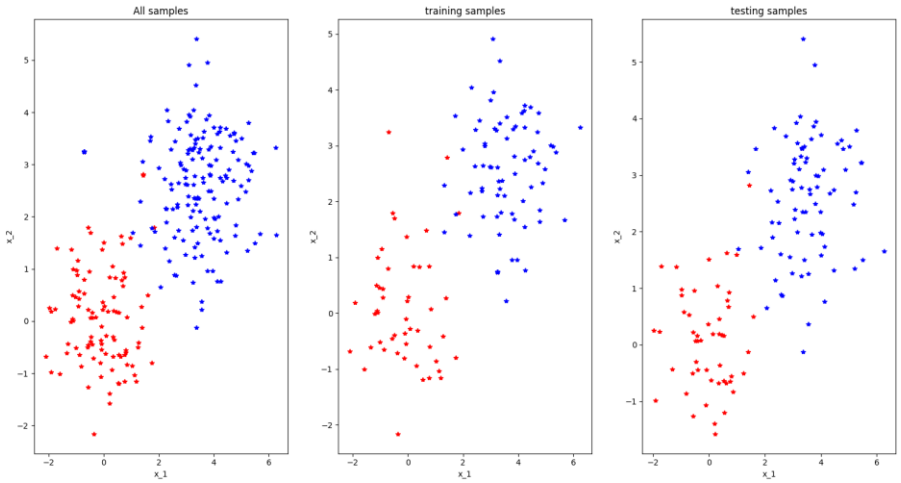


Figure 4. Gradient Descent Results

The scatter plots are no different from the sklearn method for the first two graphs. The drastic difference is in the output of the cost function and performance of the method as follows:

```
Cost function:  [0.82353556]
Cost function:  [0.82141846]
Cost function:  [0.81930315]
Cost function:  [0.81718964]
Cost function:  [0.81507793]
Cost function:  [0.81296803]
Cost function:  [0.81085996]
Cost function:  [0.80875372]
Cost function:  [0.80664932]
Cost function:  [0.80454677]
Cost function:  [0.80244607]
Cost function:  [0.80034724]
Cost function:  [0.79825029]
Cost function:  [0.79615522]
Cost function:  [0.79406204]
Cost function:  [0.79197076]
Cost function:  [0.7898814]
```

Figure 5. Cost Function

The cost function values are decreasing because the approach is iterative, and with up to 250 iterations, the model goes as small as 0.39, which is a significant decrease from the 0.8 initial values. Using the confusion matrix from problem three, the accuracy of the gradient descent method is as follows:

```
Accuracy: 82.308%
```

In contrast, the accuracy of gradient descent is ~12% lower than the sklearn logistic regression model. The amount of data trained, the randomness of data and the alpha value play a crucial role in decreasing the efficiency of the method.

Placeholder 3:

```
'''Sklearn LR Method Predictions:'''

yHat = model.predict(testX).astype(int)
```

Figure 6. Sklearn LR Predictions

In this placeholder, I tested both models using predictions code. For the sklearn, I used the predict method to produce predictions based on the testing data. The output of the model was as follows:

```
[1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 1
 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 1
 0 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1
 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 1 1 1]
```

Figure 6. Binary Data Predictions of Sklearn Method

In figure 3, the third graph displays the testing plot, and the data points are well distributed and clustered with their respective areas.

**Problem II.**

Confusion Matrix:

| Ground-Truth \ Prediction | Cat | Dog | Monkey |
|---|---|---|---|
| **Cat** | 1 | 3 | 1 |
| **Dog** | 3 | 3 | 2 |
| **Monkey** | 2 | 2 | 3 |

1. Accuracy = (1+3+3)/ (1+3+1+3+3+2+2+2+3) = (7 / 20) = 0.35 * 100 = **35 %**
2. Cat:
    a. Precision = 1 / (1+3+2) = (1 / 6) = 0.16666 * 100 = **16.66 %**
    b. Recall = 1 / (1 + 3 + 1) = (1 / 5) = 0.2 * 100 = **20 %**
3. Dog:
    a. Precision = 3 / (3+3+2) = (3 / 8) = 0.375 * 100 = **37.5 %**
    b. Recall = 3 / (3+3+2) = (3 / 8) = 0.375 * 100 = **37.5 %**
4. Monkey:
    a. Precision = 3 / (1+2+3) = (3 / 6) = 0.5 * 100 = **50 %**
    b. Recall = 3 / (2+2+3) = (3 / 7) = 0.42857 * 100 = **42.86 %**

**Problem III.**

In this problem, I coded the confusion matrix. Please refer to the end of file: "main_part1.py".

Sklearn Logistic Regression Observations:

```
Confusion Matrix Results:

Accuracy: 94.615%, Precision: 93.333%, Recall: 91.304%
```

The logistic regression method of sklearn produced accuracy, precision, and recalls above 90% most of the time. The logistic regression library provides the best of class accuracy and output for a logistic regression problem. The precision shows about 93% are the true positives in the LR model. The recall rate of 91% shows that the prediction method works well with the data of testing to predict the correct values.

Gradient Descent Observations:

```
Confusion Matrix Results:

Accuracy: 76.923%, Precision: 100.000%, Recall: 40.000%
```

On the other hand, the gradient descent method produced a much lower accuracy for the trained model with an accuracy of 77%, which is ~17% lower than the logistic regression model. However, the precision comes out to be 100%, which means in the 77% accuracy of the data, there are many true positives. Also, the recall rate is below 50%, which explains that the predictions are not anticipated to be aligned with the desired output through accuracy.