```python
from matplotlib import pyplot as plt
import numpy as np
```

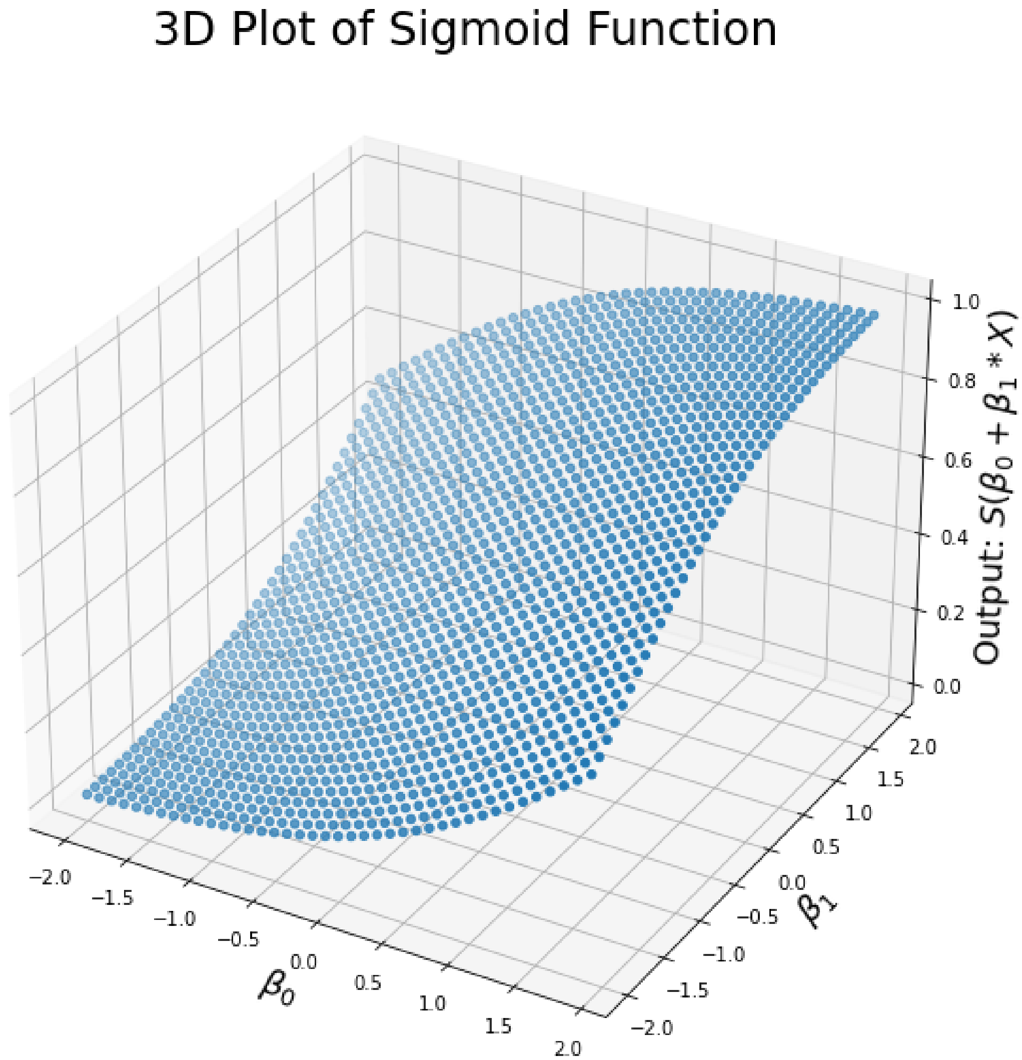
Problem 1 Part A

```python
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))
```

```python
beta_0, beta_1 = np.mgrid[-2.0:2.0:0.1, -2.0:2.0:0.1]
```

```python
X = 1
sigmoid_input_array = beta_0 + X * beta_1
output_array = sigmoid(sigmoid_input_array)
```

```python
plt.rcParams['figure.figsize']=(10, 10)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(beta_0, beta_1, output_array)

ax.set_title("3D Plot of Sigmoid Function", fontsize=24)
ax.set_xlabel(r'$\beta_0$', fontsize=18)
ax.set_ylabel(r'$\beta_1$', fontsize=18)
ax.set_zlabel(r'Output: $S(\beta_0 + \beta_1 * X)$', fontsize=18)
plt.savefig("q1_a.jpg")
plt.savefig("q1_a.png")
plt.show()
```


3D Plot of Sigmoid Function
Output: $S(\beta_0 + \beta_1 * X)$
β_0
β_1
0.0
0.2
0.4
0.6
0.8
1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

PROBLEM 1B

To prove: $P(Y|X) = \frac{1}{1+e^{-y(\beta_0+\beta_1 x)}}$ if $Y \in \{-1, 1\}$

Given: $P(Y=+1|X) = S(\beta_0+\beta_1 X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$

$$\Rightarrow P(Y=-1|X) = 1 - S(\beta_0+\beta_1 X) = \frac{1+e^{-(\beta_0+\beta_1 X)} - 1}{1+e^{-(\beta_0+\beta_1 X)}}$$

$$= \frac{e^{-(\beta_0+\beta_1 x)}}{e^{-(\beta_0+\beta_1 x)}\left[e^{(\beta_0+\beta_1 x)}+1\right]} = \frac{1}{1+e^{(\beta_0+\beta_1 x)}}$$

Thus, to generalize, we use Y as a coefficient in the exponential term

Hence Proved.

To prove: Log likelihood for m data points can be:

$$\ln \mathcal{L}(\beta_0, \beta_1) = -\sum_{i=1}^{m} \ln\left(1+e^{-y_i(\beta_0+\beta_1 X_i)}\right)$$

We know from above that $\mathcal{L}(\beta_0, \beta_1) = \prod_{i=1}^{m} \frac{1}{1+e^{-y(\beta_0+\beta_1 x)}}$

$$\Rightarrow \ln(\mathcal{L}(\beta_0, \beta_1)) = \sum_{i=1}^{m} \frac{1}{1+e^{-y(\beta_0+\beta_1 x)}} = \sum_{i=1}^{m}\left(\ln(1) - \ln\left(1+e^{-y(\beta_0+\beta_1 x)}\right)\right)$$

$$= -\sum_{i=1}^{m} \ln\left(1+e^{-y(\beta_0+\beta_1 x)}\right)$$

Hence Proved.

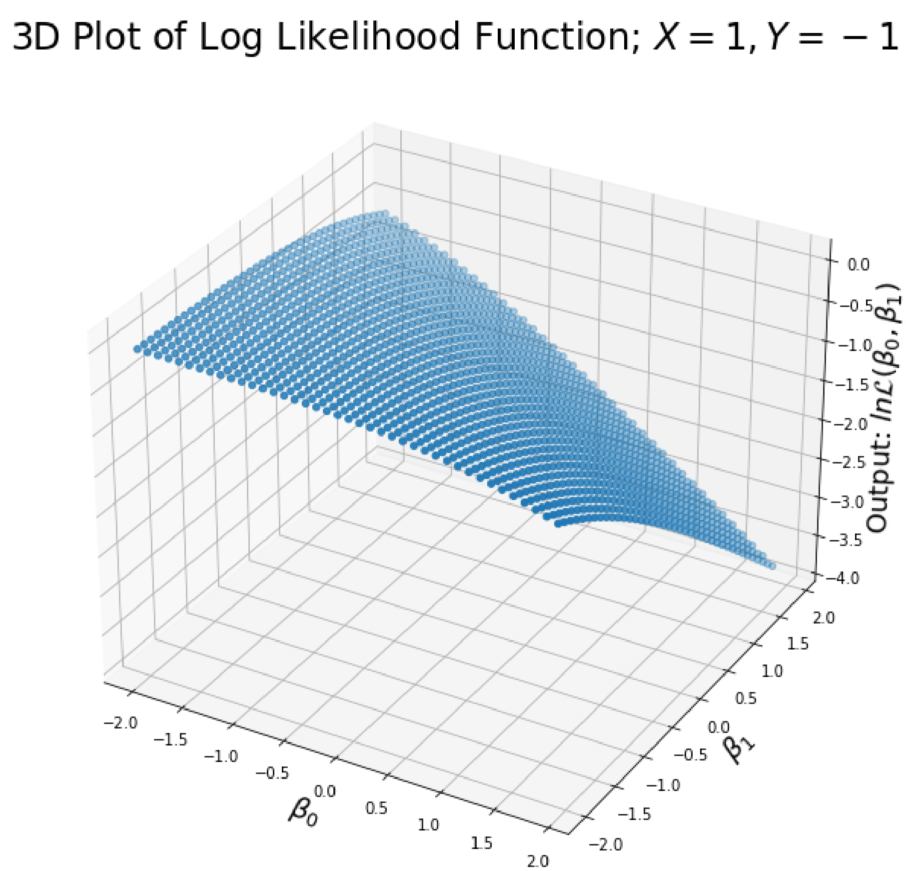
Problem 1 Part B

```python
def log_likelihood(beta_0, beta_1, X, Y):
    return -np.log(1 + np.exp(-Y * (beta_0 + beta_1 * X)))
```

```python
beta_0, beta_1 = np.mgrid[-2.0:2.0:0.1, -2.0:2.0:0.1]
```

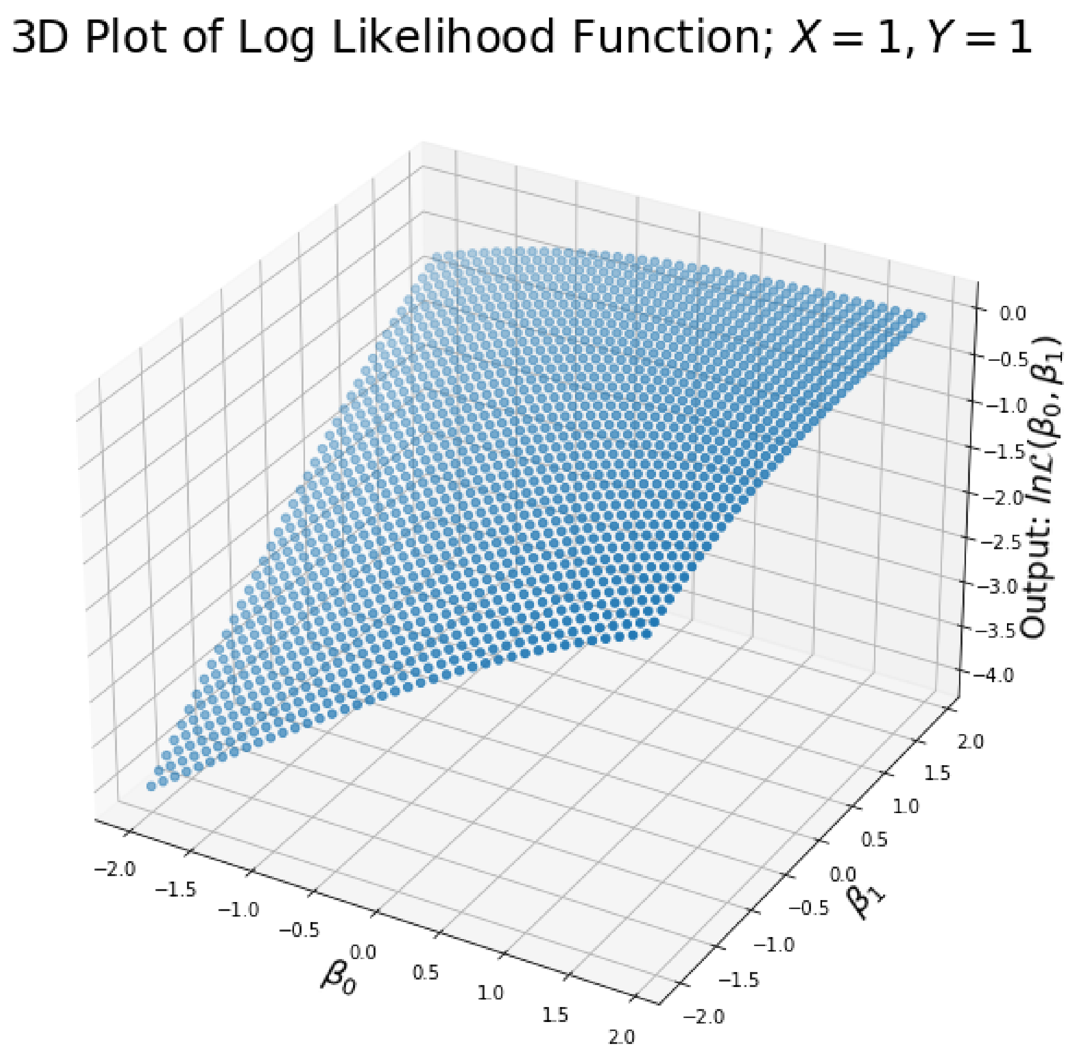
Case 1: $X = 1, Y = -1$

```python
X = 1.0
Y = -1.0
output_array = log_likelihood(beta_0, beta_1, X, Y)

plt.rcParams['figure.figsize']=(10, 10)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(beta_0, beta_1, output_array)

ax.set_title(r"3D Plot of Log Likelihood Function; $X = 1, Y = -1$", fontsize=24)
ax.set_xlabel(r'$\beta_0$', fontsize=18)
ax.set_ylabel(r'$\beta_1$', fontsize=18)
ax.set_zlabel(r'Output: $ln \mathcal{L}(\beta_0, \beta_1)$', fontsize=18)
plt.savefig("q1_b1.jpg")
plt.savefig("q1_b1.png")
plt.show()
```



Case 2: $X = 1, Y = 1$

```
X = 1.0
Y = 1.0
output_array = log_likelihood(beta_0, beta_1, X, Y)

plt.rcParams['figure.figsize']=(10, 10)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(beta_0, beta_1, output_array)

ax.set_title(r"3D Plot of Log Likelihood Function; $X = 1, Y = 1$", fontsize=24)
ax.set_xlabel(r'$\beta_0$', fontsize=18)
ax.set_ylabel(r'$\beta_1$', fontsize=18)
ax.set_zlabel(r'Output: $ln \mathcal{L}(\beta_0, \beta_1)$', fontsize=18)
plt.savefig("q1_b1.jpg")
plt.savefig("q1_b1.png")
plt.show()
```



Q: Based on the graph, is it possible to maximize this function?

As β0, β1 approach their upper, the log likelihood increases. Therefore, the functions are monotonic, and possible to find a global maxima. A caveat is that numerically, the asymptote of this graph is not feasible to reach unless we increase beta 0 and beta 1 infinitely.

PROBLEM 2 Derive the classification rule for threshold 0.5

$$P(Y=1 \mid X) = \frac{1}{1+e^{-y(\beta_0+\beta_1 x)}} \geq 0.5 \quad \text{where } y = 1$$

$$\Rightarrow \quad 1 \geq 0.5 + 0.5e^{-(\beta_0+\beta_1 x)}$$

$$\Rightarrow \quad 1 \geq e^{-(\beta_0+\beta_1 x)}$$

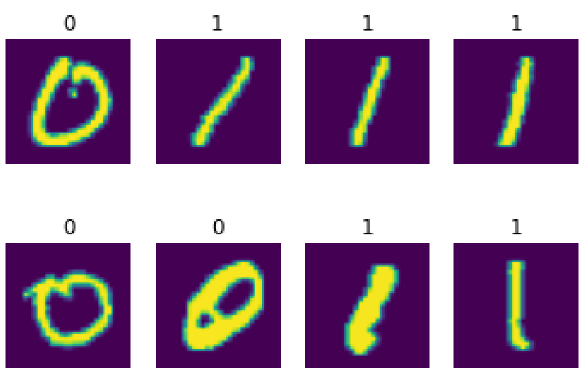
$$\Rightarrow \quad \beta_0 + \beta_1 x \geq 0$$ is the classification rule

```
import numpy as np
from matplotlib import pyplot as plt
```

```
raw_img_data = np.load('data.npy')
label_data = np.load('label.npy')
```

```
normalized_img_data = raw_img_data / 255.0
normalized_imshow_array = normalized_img_data.reshape(14780, 28, 28)
```

```
index = np.random.randint(0, len(normalized_imshow_array), size=8)
i = 0

fig, axs = plt.subplots(2, 4)
for r in range(2):
    for c in range(4):
        axs[r, c].imshow(normalized_imshow_array[index[i]].reshape(28, 28))
        axs[r, c].set_title(str(label_data[index[i]]))
        axs[r, c].axis('off')
        i+=1
```




```
relabeled_data = np.where(label_data == 1, -1, label_data)
relabeled_data = np.where(relabeled_data == 0, 1, relabeled_data)
```

```python
(unique, counts) = np.unique(relabeled_data, return_counts=True)
frequencies_relabeled = np.asarray((unique, counts)).T

(unique, counts) = np.unique(label_data, return_counts=True)
frequencies_original = np.asarray((unique, counts)).T

print("original frequency counts")
print(frequencies_original)
print("after relabeling (1 --> -1, 0 --> 1)")
print(frequencies_relabeled)
```

```
original frequency counts
[[   0 6903]
 [   1 7877]]
after relabeling (1 --> -1, 0 --> 1)
[[  -1 7877]
 [   1 6903]]
```

```python
train_num_samples = int(0.8 * len(relabeled_data))
test_num_samples = len(relabeled_data) - train_num_samples
print("80-20 Test-Train Split = " + str(train_num_samples) + "-" + str(test_num_samples))

train_indices = np.random.randint(0, len(relabeled_data), size=train_num_samples)
test_indices = np.random.randint(0, len(relabeled_data), size=test_num_samples)
```

```
80-20 Test-Train Split = 11824-2956
```

```python
X_train, X_test = normalized_img_data[train_indices], normalized_img_data[test_indices]
Y_train, Y_test = relabeled_data[train_indices], relabeled_data[test_indices]

print("Size of Train vs Test = " + str(len(X_train)) + "-" + str(len(X_test)))
```

```
Size of Train vs Test = 11824-2956
```

```python
mu, sigma = 0, 1
d = 28*28
beta_0 = np.random.normal(mu, sigma, 1)[0]
beta_1 = np.random.normal(mu, sigma, d)
```

```python
def loss_function(beta_0, beta_1, X, Y):
    """
    :param beta_0, beta_1: Logistic Regression coefficients
    :return: Loss over Training set (X and Y are globals)
    """
    loss_function_sum = 0.0
    m = len(X)
    for i in range(m):
        loss_function_sum += np.log(1 + np.exp(-Y[i] * (beta_0 + np.dot(beta_1,X[i, :]))))
    loss_function_result = loss_function_sum / m
    return loss_function_result
```

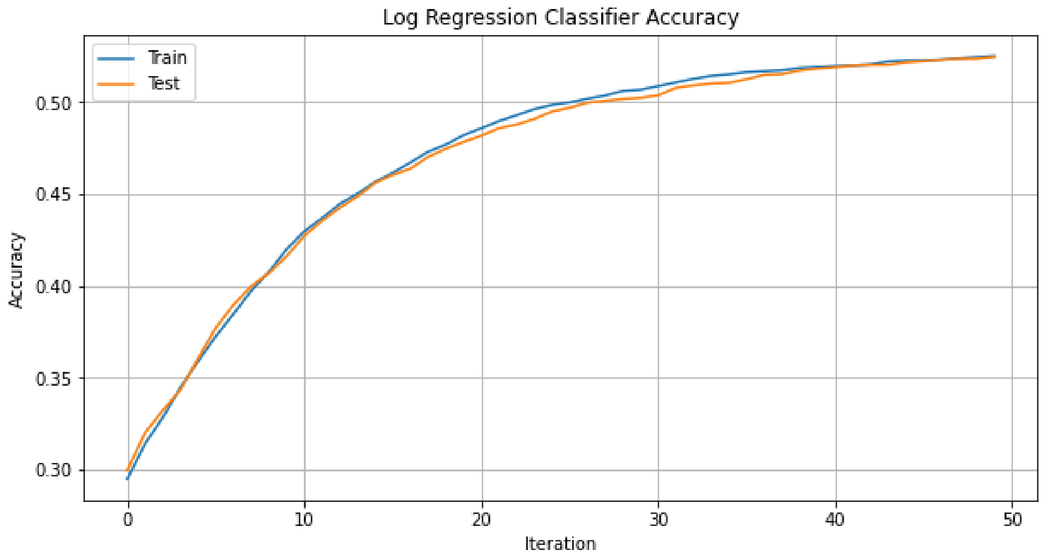
```python
def compute_gradients(beta_0, beta_1, X, Y):
    """
    :return: Gradient of beta_0, Gradient of beta_1
    """
    d_beta_0_sum = 0.0
    d_beta_1_sum = 0.0
    m = len(X)
    for i in range(m):
        exponent_term = np.exp(-Y[i] * (beta_0 + np.dot(beta_1.T,X[i, :])))
        d_beta_0_sum +=  Y[i] * exponent_term / (1 + exponent_term)
        d_beta_1_sum +=  Y[i] * X[i] * exponent_term / (1 + exponent_term)

    d_beta_0 = (-1.0 / m) * d_beta_0_sum
    d_beta_1 = (-1.0 / m) * d_beta_1_sum
    return d_beta_0, d_beta_1
```

```python
def evaluate_accuracy(beta_0, beta_1, X, Y):
    correct = 0
    for i in range(len(X)):
        predicted_label_prob = 1.0 / (1.0 + np.exp(Y[i] * (beta_0 + np.dot(beta_1.T,X[i]))))
        prediction = 1 if predicted_label_prob > 0.5 else -1
        label = Y[i]
        is_equal = np.array_equal(prediction, label)
        if is_equal:
            correct += 1

    return (correct/len(X))
```

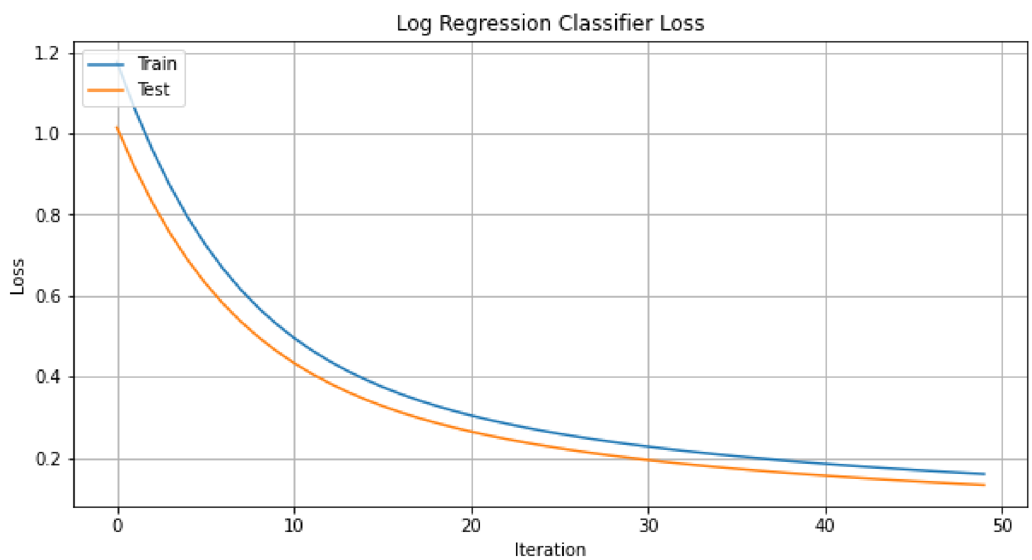
```python
num_iterations = 50
learning_rate = 0.05

training_loss_array, training_accuracy_array = [], []
test_loss_array, test_accuracy_array = [], []
for iter in range(num_iterations):

    loss = loss_function(beta_0, beta_1, X_train, Y_train)
    d_beta_0, d_beta_1 = compute_gradients(beta_0, beta_1, X_train, Y_train)

    beta_0 = beta_0 - learning_rate * d_beta_0
    beta_1 = beta_1 - learning_rate * d_beta_1

    train_accuracy = evaluate_accuracy(beta_0, beta_1, X_train, Y_train)
    test_accuracy = evaluate_accuracy(beta_0, beta_1, X_test, Y_test)
    test_loss = loss_function(beta_0, beta_1, X_test, Y_test)

    training_loss_array.append(loss)
    training_accuracy_array.append(train_accuracy)
    test_loss_array.append(test_loss)
    test_accuracy_array.append(test_accuracy)

    if(iter % 5 == 0):
        print("[{:2d}] Accuracy on test set: {:.4f}".format(iter, test_accuracy))
```

```
[ 0] Accuracy on test set: 0.2997
[ 5] Accuracy on test set: 0.3772
[10] Accuracy on test set: 0.4273
[15] Accuracy on test set: 0.4604
[20] Accuracy on test set: 0.4817
[25] Accuracy on test set: 0.4970
[30] Accuracy on test set: 0.5037
[35] Accuracy on test set: 0.5125
[40] Accuracy on test set: 0.5189
[45] Accuracy on test set: 0.5223
```

```python
plt.rcParams['figure.figsize']=(10, 5)
plt.plot(training_accuracy_array)
plt.plot(test_accuracy_array)
plt.title('Log Regression Classifier Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Iteration')
plt.legend(['Train', 'Test'], loc='upper left')
plt.grid()
plt.show()
```




```
plt.plot(training_loss_array)
plt.plot(test_loss_array)
plt.title('Log Regression Classifier Loss')
plt.ylabel('Loss')
plt.xlabel('Iteration')
plt.legend(['Train', 'Test'], loc='upper left')
plt.grid()
plt.show()
```



PROBLEM 3

Design $P(Y=y|X=x)$ such that i) $P(y=0) = P(y=1) = 0.5$

ii) Classification accuracy of any classifier ≤ 0.9

iii) Accuracy of Bayes optimal classifier is ≥ 0.8

$$P(Y=1|X) = \begin{cases} 0.85 & \text{if } x \leq 0.5 \\ 0.15 & \text{if } x \geq 0.5 \end{cases}$$

$$P(Y=0|X) = \begin{cases} 0.15 & \text{if } x \cancel{\leq} 0.5 \\ 0.85 & \text{if } x \geq 0.5 \end{cases}$$

Are results in $n=1000$ vs $n=100$ any different?

⇒ For Bayes optimal, we get ≈ 0.85 accuracy in both cases, since that is the true probability of the distribution generating a label

⇒ Logistic regression does not get better because data's underlying distribution does not match with what the logistic regression model is trying to fit.

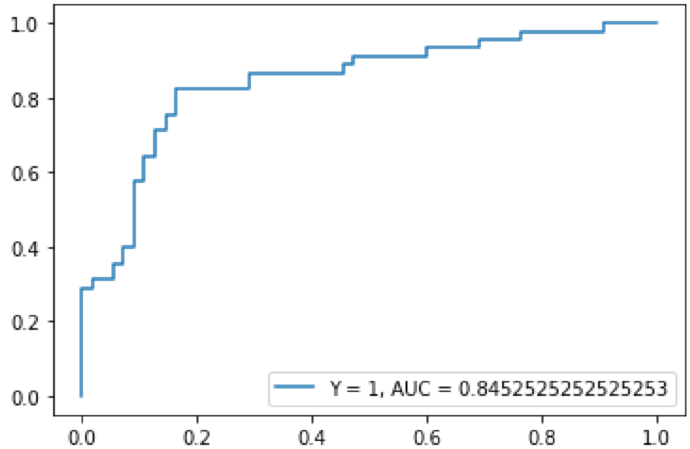
Sheet

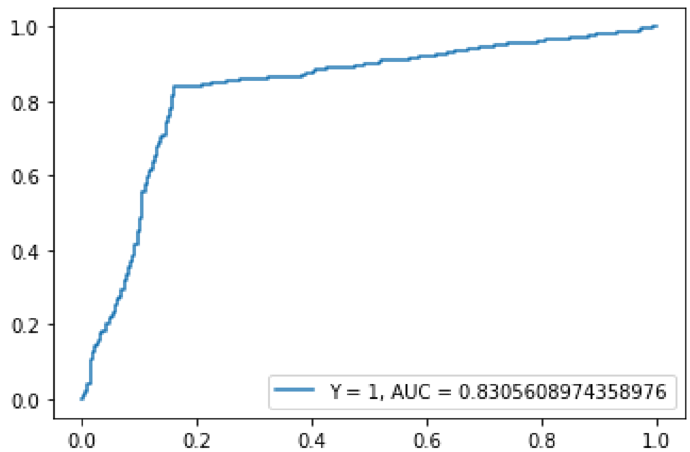
```python
import numpy as np
from sklearn.linear_model import LogisticRegression as LogReg
from sklearn.metrics import accuracy_score as accuracy
from sklearn.metrics import roc_curve, roc_auc_score
from matplotlib import pyplot as plt
```

```python
def generate_label(x):
    prob_y_equals_one = 0.85 if x < 0.5 else 0.15
    return np.random.choice([1, 0], p=[prob_y_equals_one, 1 - prob_y_equals_one])

def bayes_optimal_classifier(x):
    return 1 if x < .5 else 0

vectorized_generate_labels = np.vectorize(generate_label)
vectorized_bayes_classifier = np.vectorize(bayes_optimal_classifier)
```

```python
n_array = [100, 1000]

for n in n_array:
    X_train = np.random.uniform(0, 1, size=n).reshape(-1, 1)
    Y_train = vectorized_generate_labels(X_train).ravel()

    logistic_regression = LogReg()
    _ = logistic_regression.fit(X_train, Y_train)

    X_test = np.random.uniform(0, 1, size=n).reshape(-1, 1)
    Y_test = vectorized_generate_labels(X_test).ravel()

    Y_bayes_test = vectorized_bayes_classifier(X_test)
    Y_pred = logistic_regression.predict(X_test)

    print("===========      n = {:5d}       ===========".format(n))
    print("Logistic Regression Classifier Accuracy  : " + str(accuracy(Y_test, Y_pred)))
    print("Bayes Optimal Classifier Accuracy        : " + str(accuracy(Y_test, Y_bayes_test)))

    Y_pred_probability = logistic_regression.predict_proba(X_test)[::,1]
    false_positive_rate, true_positive_rate, _ = roc_curve(Y_test,  Y_pred_probability)
    auc = roc_auc_score(Y_test, Y_pred_probability)
    plt.plot(false_positive_rate,true_positive_rate,label="Y = 1, AUC = "+str(auc))
    plt.legend(loc=4)
    plt.show()
```

```
===========      n =   100       ===========
Logistic Regression Classifier Accuracy  : 0.7
Bayes Optimal Classifier Accuracy        : 0.83
===========      n =  1000       ===========
Logistic Regression Classifier Accuracy  : 0.82
Bayes Optimal Classifier Accuracy        : 0.839
```


Y = 1, AUC = 0.8452525252525253
0.0
0.2
0.4
0.6
0.8
1.0


Y = 1, AUC = 0.8305608974358976
0.0
0.2
0.4
0.6
0.8
1.0

PROBLEM 4

TODO: Determine an update rule for the centroid c_k of the k-th cluster C_k

Find optimal c_k that minimizes the objective function.

The data x contains p features

$\boxed{4.1}$ To prove: $\sum_{k=1}^{K} \sum_{x \in C_k} \sum_{i=1}^{p} (c_{k_i} - x_i)^2$ results in an update rule

where the optimal centroid is the mean of the points in the cluster.

Proof: Within a given centroid (independent & non-overlapping with other centroids), we have:

$$\sum_{x \in C_k} \sum_{i=1}^{p} (c_{k_i} - x_i)^2 \quad \Leftarrow \text{update rule}$$

To simplify, first consider the case of a single feature. We can later generalize.

$$\Rightarrow \sum_{x \in C_k} (c_{k_i} - x_i)^2 \quad \Leftarrow \text{update rule.}$$

First derivative set equal to $0 \Rightarrow \quad 2 \sum_{x \in C_k} (c_{k_i} - x_i) = 0$

$$\Rightarrow \sum_{x \in C_k} c_{k_i} = \sum_{x \in C_k} x_i$$

Assuming centroid c_k has n_k data points $\Rightarrow \quad n_k c_{k_i} = \frac{n_k}{n_k} \sum x_i$

$$\Rightarrow c_{k_i} = \bar{X} \quad \text{(average of points in centroid k)}$$

$\boxed{4.2}$ $\sum_{k=1}^{K} \sum_{x \in C_k} \sum_{i=1}^{P} |c_{ki} - x_i|$ result in centroid as median of cluster

Following same simplifications as 4.1

$$\Rightarrow \sum_{x \in C_k} \sum_{i=1}^{P} \text{sign}(c_{ki} - x_i) = 0$$

(first derivative of $|c_{ki} - x_i|$)

$$\text{sign}(c_{ki} - x_i) = \begin{cases} -1 & \text{if } c_{ki} < \cancel{x_i}\ x_i \\ 1 & \text{if } c_{ki} \geq \cancel{x_i}\ x_i \end{cases}$$

To ~~minimize~~ set ~~this~~ the LHS function equal to zero,

the # of -1s & $+1$s must be equal (negating each other & resulting in a 0 sum)

$$\Rightarrow \# \text{ of } x_i\text{'s} > c_{ki} = \# \text{ of } x_i\text{s} \leq c_{ki}$$

$\Rightarrow$ c_{ki} by definition must be the median of all the points in the ~~centroid~~. cluster.