1a) Implementing Logistic Regression learning via gradient ascent.

As always start with importing what we need and finding our data locations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, roc_curve, auc,
roc_auc_score

import os

os.chdir("C:/Users/rique/Downloads/Gisette")
```

For part a) We're using the Gisette data to train our regressor. With w^(0) = 0, 300 gradient ascent iterations and a shrinkage of 0.0001.

load the data.

```
g_train_data = np.loadtxt("gisette_train.data")
g_train_labels = np.loadtxt("gisette_train.labels")
g_valid_data = np.loadtxt("gisette_valid.data")
g_valid_labels = np.loadtxt("gisette_valid.labels")
```

Normalize the features.

```
std=np.std(g_train_data, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
g_train_data = g_train_data[:, mask]
mean = np.mean(g_train_data, axis=0)
true_std = np.std(g_train_data, axis=0)

#standardize the features in both training and test datasets
g_train_data = (g_train_data-mean)/true_std
g_valid_data = g_valid_data[:, mask]
g_valid_data = (g_valid_data-mean)/true_std
```

```
#add a bias term
g_train_data = np.insert(g_train_data, 0, 1, axis=1)
g_valid_data = np.insert(g_valid_data, 0, 1, axis=1)

g_train_labels[g_train_labels == -1] = 0
g_valid_labels[g_valid_labels == -1] = 0
```

Initalize our needed parameters

```
iterations = 300

#note; this value may need to be adjusted
learning_rate = 0.01

#lambda
shrinkage = 0.0001

#weight
w = np.zeros(g_train_data.shape[1])

log_likelihoods = np.zeros(iterations)
train_misclass_errors = np.zeros(iterations)
valid_misclass_errors = np.zeros(iterations)

fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []

fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []
```

Train via gradient Ascent

```
for i in range(iterations):

    #dot product of train data and weight
    dot=np.sum(g_train_data*w, axis=1)
    exp=np.exp(dot)

    #gradient ascent
    gradient = np.sum((g_train_labels-exp/(1+exp))*(g_train_data).T,
axis=1)

    #update the weight with our gradient ascent
    w = (1 - learning_rate * shrinkage) * w +
learning_rate/g_train_data.shape[0] * gradient

    #recalculate dot product of train data and updated weight
    dot = np.sum(g_train_data * w, axis=1)
```

```
    #get the log likelihood
    ll = np.sum(g_train_labels * dot - np.log(1 + np.exp(dot)),
axis=0)
    log_likelihoods[i] = ll

    #prediction here is based on if the dot product of train/test sets
is greater than zero, and have it compared to corresponding labels
    y_pred = ((dot >=0) == g_train_labels)
    misclass_error_train = 1 - accuracy_score(g_train_labels, y_pred)
    train_misclass_errors[i] = misclass_error_train

    dotValid = np.sum(g_valid_data * w, axis=1)
    yTest_pred = ((dotValid >=0) == g_valid_labels)
    misclass_error_valid = 1 - accuracy_score(g_valid_labels,
yTest_pred)
    valid_misclass_errors[i] = misclass_error_valid

     # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(g_train_labels, 1 / (1 +
np.exp(-dot)))
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
    fpr_valid, tpr_valid, _ = roc_curve(g_valid_labels, 1 / (1 +
np.exp(-dotValid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)

    #print(f"Iteration: {i}'s Log Likelihood: {ll}\n mis class train
error: {misclass_error_train}\n mis class test error:
{misclass_error_valid}")
```

Plot the stuff

```
# Plot log likelihood vs iteration
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), log_likelihoods, label='Log Likelihood')
plt.xlabel('Iterations')
plt.ylabel('Log Likelihood')
plt.title('Log Likelihood vs. Iterations')

# Plot ROC curves for both training and validation sets
plt.subplot(1, 2, 2)
```

```python
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')

# Add a dashed diagonal line (no-discrimination ROC curve)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()

plt.show()
```

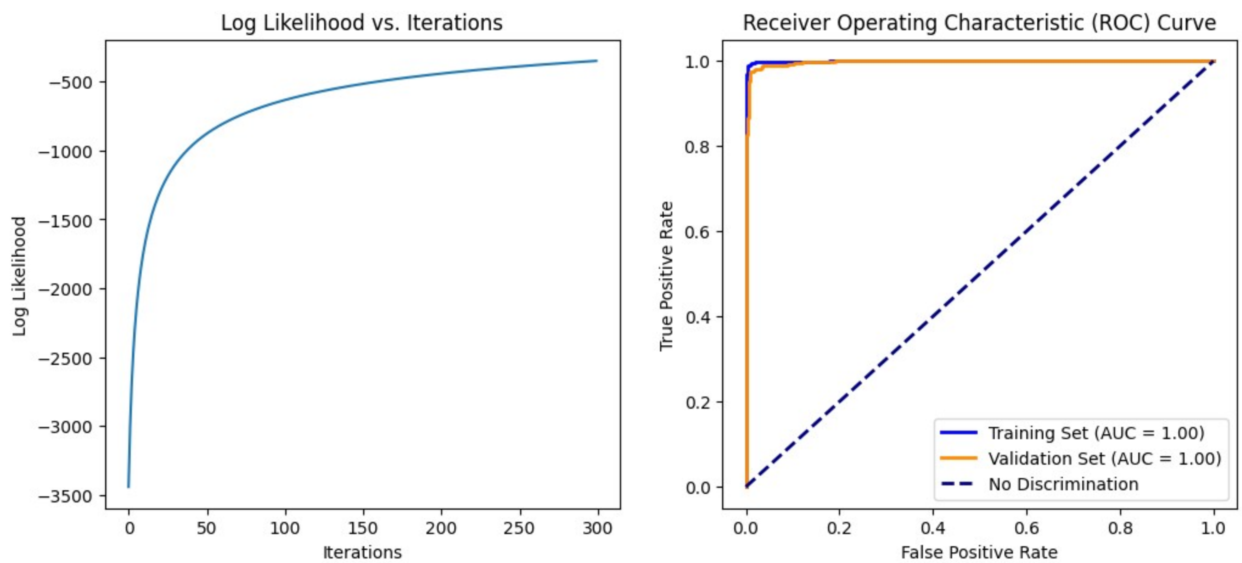


Table with the Misclass errors on test and training sets

```python
results = pd.DataFrame({
    'Iteration': range(iterations),
    'Training Misclassification Error': train_misclass_errors,
    'Validation Misclassification Error': valid_misclass_errors
})
print(results)
```

```
     Iteration  Training Misclassification Error  \
0            0                          0.471000
1            1                          0.474833
2            2                          0.477000
3            3                          0.478000
4            4                          0.477833
..         ...                               ...
295        295                          0.501500
```

```
296         296                           0.501667
297         297                           0.501667
298         298                           0.501667
299         299                           0.501667

     Validation Misclassification Error
0                                 0.456
1                                 0.464
2                                 0.466
3                                 0.469
4                                 0.471
..                                  ...
295                               0.500
296                               0.500
297                               0.500
298                               0.500
299                               0.500

[300 rows x 3 columns]
```

1b) Repeating a) but on the hill-valley set.

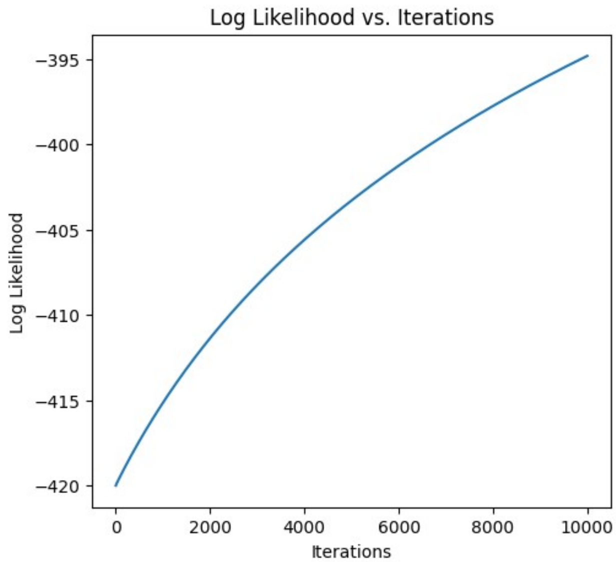
Copy & Pasting above code, with only differences being the dataset, and iterations going up to 10k.

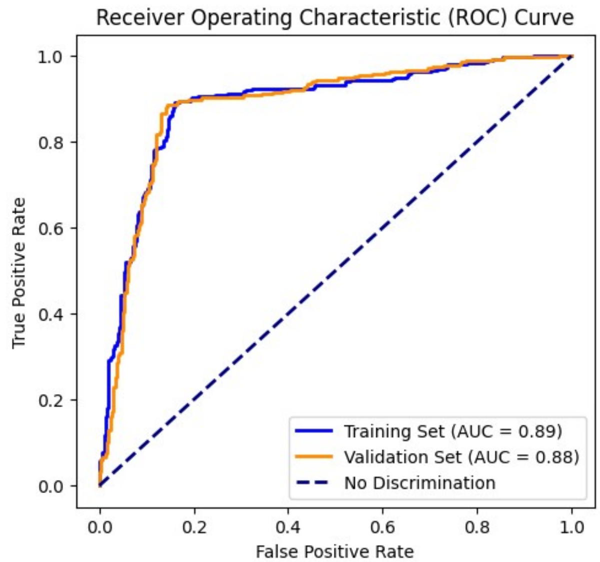
```
os.chdir("C:/Users/rique/Downloads/HV")

hv_train_data = np.loadtxt("X.dat")
hv_train_labels = np.loadtxt("Y.dat")
hv_test_data = np.loadtxt("Xtest.dat")
hv_test_labels = np.loadtxt("Ytest.dat")

std=np.std(hv_train_data, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
hv_train_data = hv_train_data[:, mask]
mean = np.mean(hv_train_data, axis=0)
true_std = np.std(hv_train_data, axis=0)

#standardize the features in both training and test datasets
hv_train_data = (hv_train_data-mean)/true_std
hv_test_data = hv_test_data[:, mask]
hv_test_data = (hv_test_data-mean)/true_std

#add a bias term
```

```
hv_train_data = np.insert(hv_train_data, 0, 1, axis=1)
hv_test_data = np.insert(hv_test_data, 0, 1, axis=1)

hv_train_labels[hv_train_labels == -1] = 0
hv_test_labels[hv_test_labels == -1] = 0

iterations = 10000

#note; this value may need to be adjusted
learning_rate = 0.01

#lambda
shrinkage = 0.0001

#weight
w = np.zeros(hv_train_data.shape[1])

log_likelihoods = np.zeros(iterations)
train_misclass_errors = np.zeros(iterations)
valid_misclass_errors = np.zeros(iterations)

fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []

fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

for i in range(iterations):

    #dot product of train data and weight
    dot=np.sum(hv_train_data*w, axis=1)
    exp=np.exp(dot)

    #gradient ascent
    gradient = np.sum((hv_train_labels-exp/(1+exp))*(hv_train_data).T,
axis=1)

    #update the weight with our gradient ascent
    w = (1 - learning_rate * shrinkage) * w +
learning_rate/hv_train_data.shape[0] * gradient

    #recalculate dot product of train data and updated weight
    dot = np.sum(hv_train_data * w, axis=1)

    #get the log likelihood
    ll = np.sum(hv_train_labels * dot - np.log(1 + np.exp(dot)),
axis=0)
    log_likelihoods[i] = ll
```

```
    #prediction here is based on if the dot product of train/test sets
is greater than zero, and have it compared to corresponding labels
    y_pred = ((dot >=0) == hv_train_labels)
    misclass_error_train = 1 - accuracy_score(hv_train_labels, y_pred)
    train_misclass_errors[i] = misclass_error_train

    dotValid = np.sum(hv_test_data * w, axis=1)
    yTest_pred = ((dotValid >=0) == hv_test_labels)
    misclass_error_valid = 1 - accuracy_score(hv_test_labels,
yTest_pred)
    valid_misclass_errors[i] = misclass_error_valid

     # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(hv_train_labels, 1 / (1 +
np.exp(-dot)))
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
    fpr_valid, tpr_valid, _ = roc_curve(hv_test_labels, 1 / (1 +
np.exp(-dotValid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)

# Plot log likelihood vs iteration
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), log_likelihoods, label='Log Likelihood')
plt.xlabel('Iterations')
plt.ylabel('Log Likelihood')
plt.title('Log Likelihood vs. Iterations')

# Plot ROC curves for both training and validation sets
plt.subplot(1, 2, 2)
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')

# Add a dashed diagonal line (no-discrimination ROC curve)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
```

```
plt.show()
```






```
results = pd.DataFrame({
    'Iteration': range(iterations),
    'Training Misclassification Error': train_misclass_errors,
    'Validation Misclassification Error': valid_misclass_errors
})
print(results)
```

```
      Iteration  Training Misclassification Error  \
0             0                          0.795380
1             1                          0.795380
2             2                          0.797030
3             3                          0.797030
4             4                          0.797030
...         ...                               ...
9995       9995                          0.851485
9996       9996                          0.851485
9997       9997                          0.851485
9998       9998                          0.851485
9999       9999                          0.851485

      Validation Misclassification Error
0                               0.767327
1                               0.767327
2                               0.767327
3                               0.767327
4                               0.767327
...                                  ...
9995                            0.839934
9996                            0.839934
```

```
9997                                0.839934
9998                                0.839934
9999                                0.839934

[10000 rows x 3 columns]
```

1c) again, just like a) and b) same thing. But using dexter dataset.

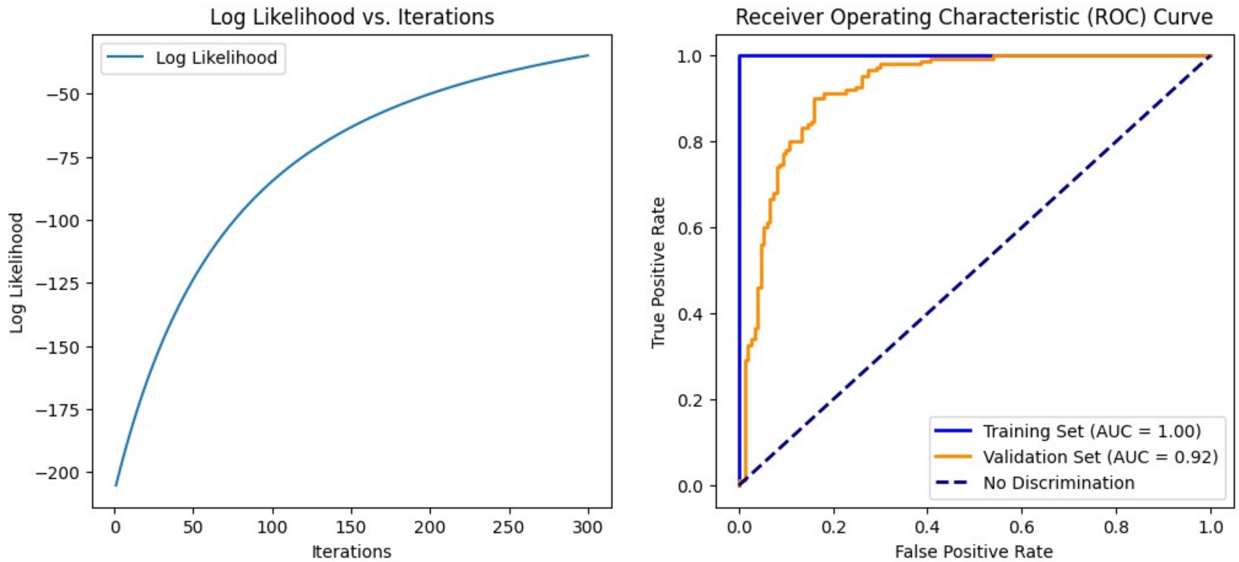
again, just gnna copy and past above code and change the dataset.

Fun fact, i hated this problem because I downloaded the wrong dataset. and it took me over 24 hours to realize that mistake, I thought we werent given labels, but we were. IN A DIFFERENT ZIP FOLDER

```
os.chdir("C:/Users/rique/Downloads/dexter")

X=np.genfromtxt('dexter_train.csv', delimiter=',')
y=np.loadtxt('dexter_train.labels')
Xtest=np.genfromtxt('dexter_valid.csv', delimiter=',')
yTest=np.loadtxt('dexter_valid.labels')

std=np.std(X, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
X = X[:, mask]
mean = np.mean(X, axis=0)
true_std = np.std(X, axis=0)

#standardize the features in both training and test datasets
X = (X-mean)/true_std
Xtest = Xtest[:, mask]
Xtest = (Xtest-mean)/true_std

#add a bias term
X = np.insert(X, 0, 1, axis=1)
```

```
Xtest = np.insert(Xtest, 0, 1, axis=1)

y[y == -1] = 0
yTest[yTest == -1] = 0

iterations = 300

#note; this value may need to be adjusted
learning_rate = 0.001

#lambda
shrinkage = 0.0001

#weight
w = np.zeros(X.shape[1])

log_likelihoods = np.zeros(iterations)
train_misclass_errors = np.zeros(iterations)
valid_misclass_errors = np.zeros(iterations)

fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []

fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

for i in range(iterations):

   #dot product of train data and weight
    dot=np.sum(X*w, axis=1)
    exp=np.exp(dot)

    #gradient ascent
    gradient = np.sum((y-exp/(1+exp))*(X).T, axis = 1)

    #update the weight with our gradient ascent
    w = (1 - learning_rate * shrinkage) * w + learning_rate/X.shape[0]
* gradient

    #recalculate dot product of train data and updated weight
    dot = np.sum(X * w, axis=1)

    #get the log likelihood
    log_likelihoods[i] = np.sum(y*dot-np.log(1+np.exp(dot)), axis=0)

    #prediction here is based on if the dot product of train/test sets
is greater than zero, and have it compared to corresponding labels
    y_pred = ((dot >=0) == y)
    misclass_error_train = 1 - accuracy_score(y, y_pred)
```

```
    train_misclass_errors[i] = misclass_error_train

    dotValid = np.sum(Xtest * w, axis=1)
    yTest_pred = ((dotValid >=0) == yTest)
    misclass_error_valid = 1 - accuracy_score(yTest, yTest_pred)
    valid_misclass_errors[i] = misclass_error_valid

   # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(y, 1 / (1 + np.exp(-dot)))
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
    fpr_valid, tpr_valid, _ = roc_curve(yTest, 1 / (1 + np.exp(-
dotValid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)


# Plot log likelihood vs. iteration
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, 301), log_likelihoods, label='Log Likelihood')
plt.xlabel('Iterations')
plt.ylabel('Log Likelihood')
plt.title('Log Likelihood vs. Iterations')
plt.legend()

# Plot ROC curves for both training and validation sets
plt.subplot(1, 2, 2)
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')

# Add a dashed diagonal line (no-discrimination ROC curve)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()

plt.show()
```




```
results = pd.DataFrame({
    'Iteration': range(iterations),
    'Training Misclassification Error': train_misclass_errors,
    'Validation Misclassification Error': valid_misclass_errors
})
print(results)
```

```
     Iteration  Training Misclassification Error  \
0            0                               0.5
1            1                               0.5
2            2                               0.5
3            3                               0.5
4            4                               0.5
..         ...                               ...
295        295                               0.5
296        296                               0.5
297        297                               0.5
298        298                               0.5
299        299                               0.5

     Validation Misclassification Error
0                              0.446667
1                              0.446667
2                              0.446667
3                              0.446667
4                              0.446667
..                                  ...
295                            0.453333
296                            0.453333
297                            0.453333
298                            0.453333
299                            0.453333
```

```
[300 rows x 3 columns]
```

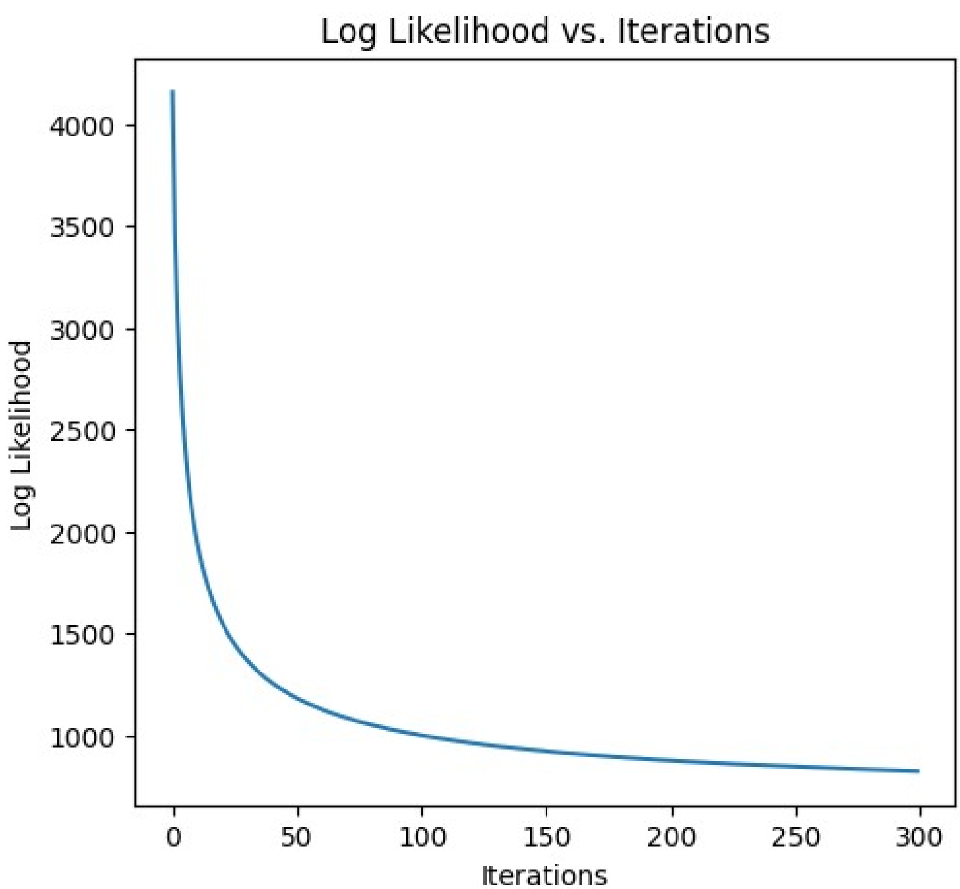
2a Minimize Gisette dataset via gradient descent.

Starts off the same as previous problems, load, normalize, initalize params.

```
os.chdir("C:/Users/rique/Downloads/Gisette")

g_train_data = np.loadtxt("gisette_train.data")
g_train_labels = np.loadtxt("gisette_train.labels")
g_valid_data = np.loadtxt("gisette_valid.data")
g_valid_labels = np.loadtxt("gisette_valid.labels")

std=np.std(g_train_data, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
g_train_data = g_train_data[:, mask]
mean = np.mean(g_train_data, axis=0)
true_std = np.std(g_train_data, axis=0)

#standardize the features in both training and test datasets
g_train_data = (g_train_data-mean)/true_std
g_valid_data = g_valid_data[:, mask]
g_valid_data = (g_valid_data-mean)/true_std

#add a bias term
g_train_data = np.insert(g_train_data, 0, 1, axis=1)
g_valid_data = np.insert(g_valid_data, 0, 1, axis=1)

g_train_labels[g_train_labels == -1] = 0
g_valid_labels[g_valid_labels == -1] = 0

iterations = 300

#note; this value may need to be adjusted
learning_rate = 0.01

#lambda
shrinkage = 0.01

#weight
```

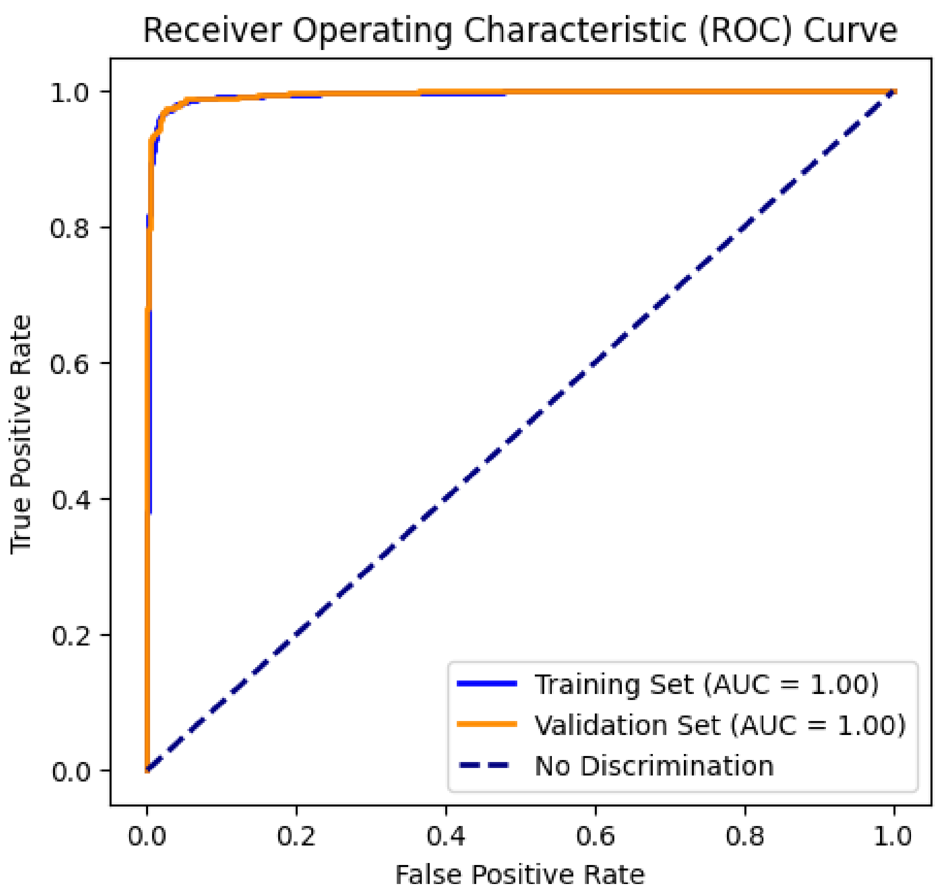
```
w = np.zeros(g_train_data.shape[1])

log_likelihoods = np.zeros(iterations)
train_misclass_errors = np.zeros(iterations)
valid_misclass_errors = np.zeros(iterations)

fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []

fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

## Gradient Descent

for i in range(iterations):
    # Calculate the sigmoid function
    z = np.dot(g_train_data, w)
    predictions = 1 / (1 + np.exp(-z))

    # Calculate the gradient of the logistic loss with L1
regularization
    gradient = np.dot(g_train_data.T, (predictions - g_train_labels))
/ len(g_train_labels)
    l1_penalty = shrinkage * np.sign(w)
    gradient += l1_penalty

    # Update weights using gradient descent
    w -= learning_rate * gradient

    # Calculate log-likelihood and misclassification error for
training data
    log_likelihood = np.sum(g_train_labels * np.log(predictions) + (1
- g_train_labels) * np.log(1 - predictions))
    log_likelihoods[i] = log_likelihood
    train_predictions = predictions >= 0.5
    train_misclass_errors[i] = np.mean(train_predictions !=
g_train_labels)

    # Calculate misclassification error for validation data
    z_valid = np.dot(g_valid_data, w)
    valid_predictions = 1 / (1 + np.exp(-z_valid))
    valid_predictions = valid_predictions >= 0.5
    valid_misclass_errors[i] = np.mean(valid_predictions !=
g_valid_labels)

    # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(g_train_labels, 1 / (1 +
np.exp(-z)))
```

```
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
    fpr_valid, tpr_valid, _ = roc_curve(g_valid_labels, 1 / (1 +
np.exp(-z_valid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)

# Plot log likelihood vs iteration
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), -log_likelihoods, label='Log Likelihood')
plt.xlabel('Iterations')
plt.ylabel('Log Likelihood')
plt.title('Log Likelihood vs. Iterations')

plt.show()
```



2b) Plotting the ROC curve from a)

```
plt.figure(figsize=(12, 5))
# Plot ROC curves for both training and validation sets for the last
iteration (or any specific iteration)
plt.subplot(1, 2, 2)
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')

# Add a dashed diagonal line (no-discrimination ROC curve)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()

plt.show()
```



2c) How many nonzero entries are in w? How many values in w satisfy |W_i| > lambda

```
nonzero_count = np.count_nonzero(w)
weights_above_lambda = np.sum(np.abs(w) > shrinkage)

print(f"nonzero's in w = {nonzero_count}")
print(f"values satisfy w > lambda = {weights_above_lambda}")
```

```
nonzero's in w = 4956
values satisfy w > lambda = 391
```