Hiva Mohammadzadeh
3036919598

Question 1: Honor Code

“I certify that all solutions are entirely in my own words and that I have not looked at another student’s solutions. I have given credit to all external sources I consulted.”

Question 2: Logistic Regression with Newton’s Method

Question 3: Wine Classification with Logistic Regression

Preprocessing: Normalizing, shuffling and splitting:

```
Main data: (6000, 12)
Main labels: (6000, 1)
Main test: (497, 12)

Shapes after shuffle and split:
Main data: (5000, 12)
Main labels: (5000,)
Validation data: (1000, 12)
Validation labels:(1000,)

After normalizing training
Main data (5000, 13)
[[ 0.21238764 -0.42529608  0.28218208 ...  0.42684345 -0.92568693
   1.        ]
 [-0.4041222  -0.24219715  1.24063449 ... -1.32978738  0.21432161
   1.        ]
 [ 1.59953477  0.91742937  0.28218208 ...  0.34319436  0.21432161
   1.        ]
 ...
 [-0.24999474 -1.15769178 -0.19704412 ...  0.59414162  1.35433015
   1.        ]
 [-0.24999474 -0.79149393  0.07679942 ... -0.15870016  0.21432161
   1.        ]
 [ 0.13532391  3.81649567 -2.18240982 ... -0.24234925 -0.92568693
   1.        ]]
```

. . .

```
After normalizing test
Test data (497, 13)
[[ 0.46904252 -0.5677573   0.3362711  ...  0.77846078 -0.90726388
   1.        ]
 [ 2.13766491  0.85572333 -0.8290917  ... -1.06263446  0.26089713
   1.        ]
 [ 0.84827488  1.1522818  -0.28068568 ... -0.39314528  0.26089713
   1.        ]
 ...
 [-1.65465871  0.32191809  1.22743088 ...  2.61955602  0.26089713
   1.        ]
 [ 0.31734958  0.2626064  -0.07503342 ... -0.64420373 -0.90726388
   1.        ]
 [-0.82034751 -0.50844561  1.29598163 ... -0.56051758  0.26089713
   1.        ]]
```

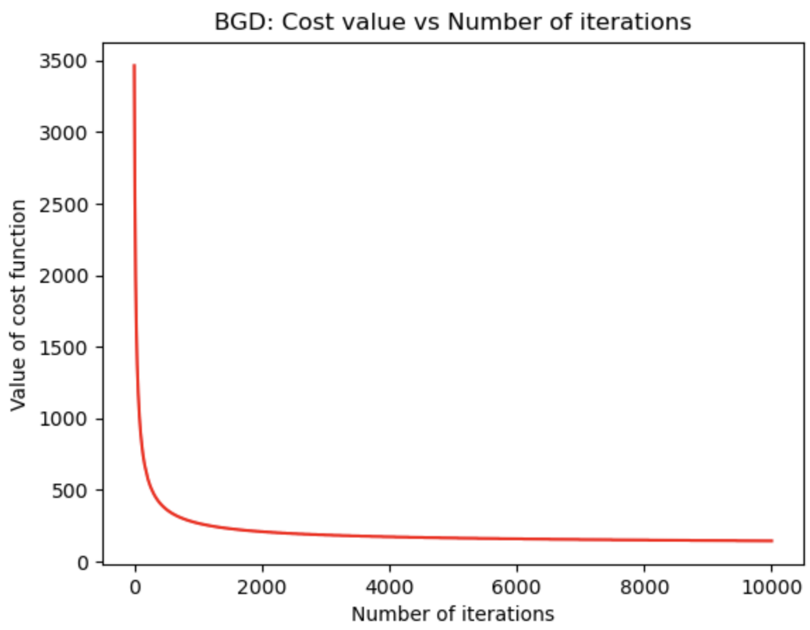
Part 1: Batch Gradient Descent Update Rule:

$$\boxed{w \leftarrow w + (X^T(y - \sigma(Xw))) - \frac{\lambda}{n} \sum_{j=1}^{d} w_j}$$

$\rightarrow$ update rule

Part 2: Batch Gradient Descent Code:

```
---- Training Batch gradient Descent---
Training accuracy: 0.9928
Validation accuracy: 0.995
```



Setting hyperparameters:

```
Setting Hyperparameters for Batch Gradient Descent:
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.537
Parameters(1e-07, 0.001, 500): Validation accuracy: 0.568
Parameters(1e-07, 0.001, 1000): Validation accuracy: 0.595
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
Parameters(1e-07, 0.001, 10000): Validation accuracy: 0.889
```

■ ■ ■

```
Best validation accuracy:  0.992

Best hyperparameters combo:(learning rate, regularization parameter, number of iteration):  (1e-05, 0.001, 10000)
```

Part 3: Stochastic Gradient Descent (SGD) Update Rule:

3) $w \leftarrow w + \alpha [(y_i - \sigma(x_i; w)) x_i - \frac{\lambda}{n} \sum_{j=1}^{d} w_j$ → update rule

Part 4: Stochastic Gradient Descent Code:

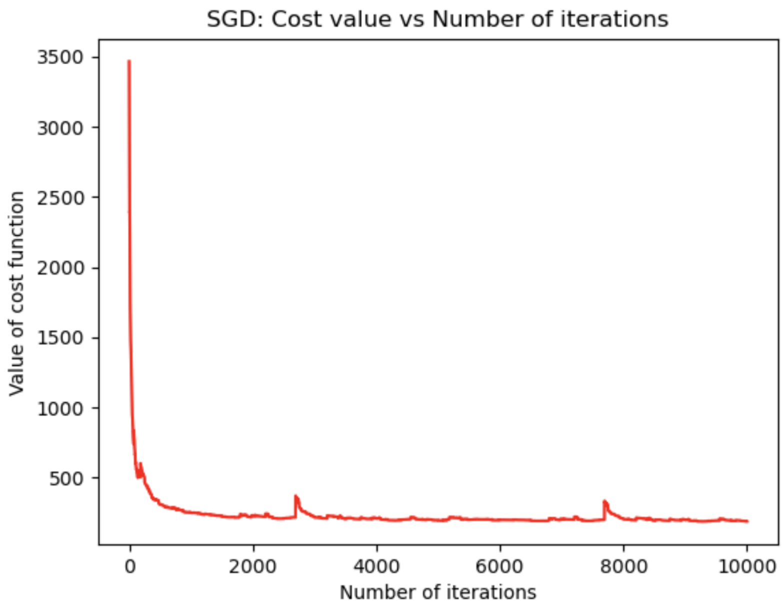
```
---- Training Stochastic Gradient Descent ---
Training accuracy: 0.993
Validation accuracy: 0.995
```

Setting hyperparameters:

```
Setting Hyperparameters for Stochastic Gradient Descent (SGD):
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
Parameters(1e-07, 0.001, 100): Validation accuracy: 0.945
```

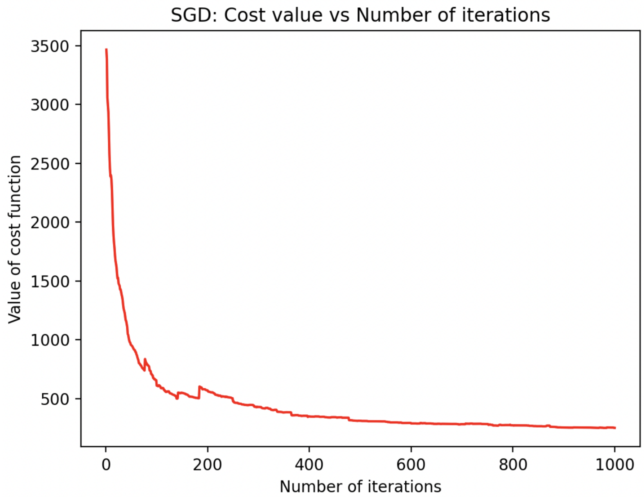
...

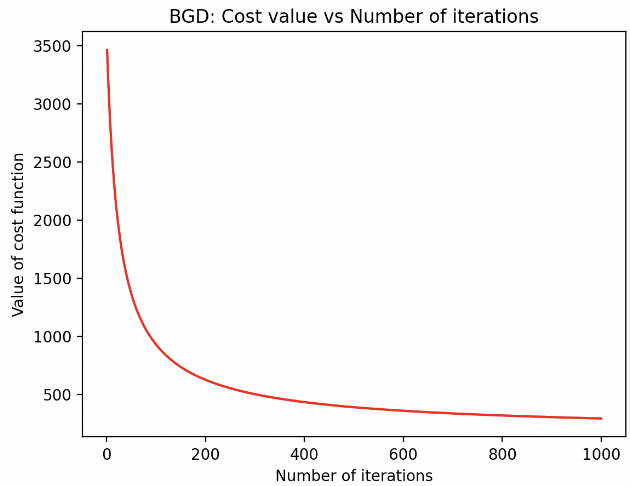
```
Parameters(0.1, 0.001, 10000): Validation accuracy: 0.992
Parameters(0.1, 0.001, 10000): Validation accuracy: 0.992
Parameters(0.1, 0.001, 10000): Validation accuracy: 0.992
Parameters(0.1, 0.001, 10000): Validation accuracy: 0.992
best validation accuracy:  0.992
best hyperparameters combo:(learning rate, regularization parameter, number of iteration):  (0.1, 0.001, 10000)
```



Compare your plot here with that of question 3.2. Which method converges more quickly?

I trained both BGD and SGD with 1000 training samples to see the difference in their convergence. As we can see Stochastic Gradient Descent converges faster. In SGD, we sample with replacement. So, we pick a random point and label every iteration.





Part 5: SGD with Decreasing Learning Rate

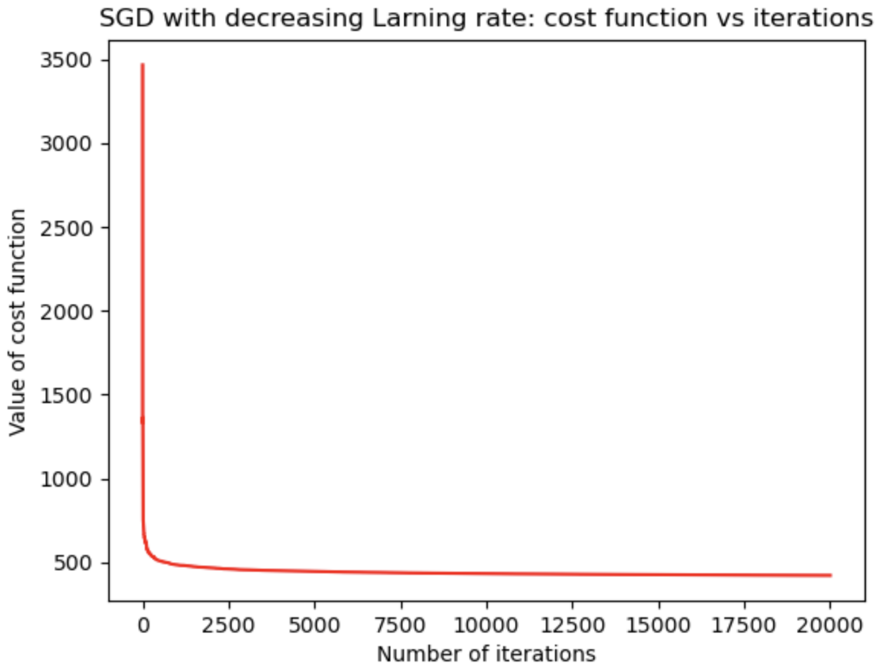
```
---- Training Stochastic Gradient Descent with decaying learning rate ---
Training accuracy: 0.9848
Validation accuracy: 0.985
```

Setting hyperparameters:

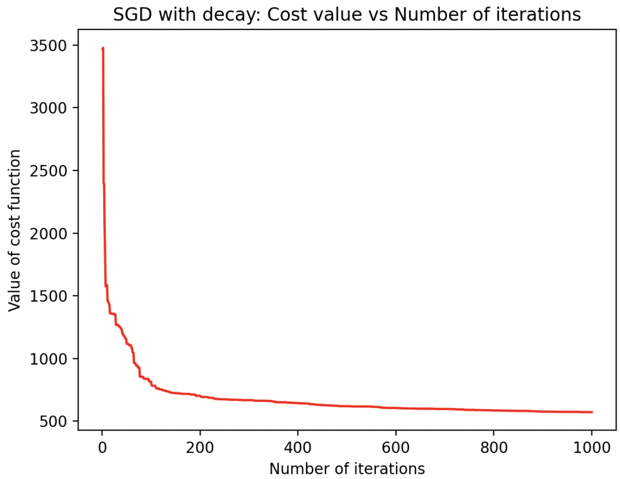
```
Setting Hyperparameters for Stochastic Gradient Descent (SGD) with decaying learning rate:
Parameters(1e-07, 0.001, 100, 0.1): Validation accuracy: 0.927
Parameters(1e-07, 0.001, 500, 0.1): Validation accuracy: 0.936
Parameters(1e-07, 0.001, 1000, 0.1): Validation accuracy: 0.941
Parameters(1e-07, 0.001, 10000, 0.1): Validation accuracy: 0.952
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
Parameters(1e-07, 0.001, 20000, 0.1): Validation accuracy: 0.954
```

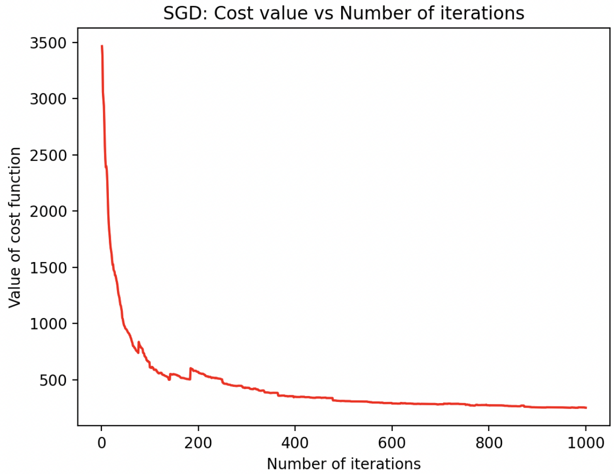
■ ■ ■

```
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
Parameters(1e-07, 0.01, 20000, 2): Validation accuracy: 0.985
best validation accuracy:  0.985
best hyperparameters combo:(learning rate, regularization parameter, number of iteration, delta):  (1e-07, 0.01, 20000, 2)
---- Training Stochastic Gradient Descent with decaying learning rate ---
Training accuracy: 0.9848
Validation accuracy: 0.985
```



How does this compare to the convergence of your previous SGD code?





I trained both SGD and SGD with decaying learning rate with 1000 training samples to see the difference in their convergence. As we can see SGD with decaying learning rate converges faster. The delta decreases the learning rate allowing it to converge more quickly.

Part 6: Kaggle Submission

```
---- Training Best Performing Batch gradient Descent for testing---
Tested the data and Saved the predictions
```

Kaggle username: **Hiva Mohammadzadeh**

Kaggle Scores:

a) WINE: Score: 0.97580 = 97.6%

Briefly describe what your best classifier does to achieve that score.

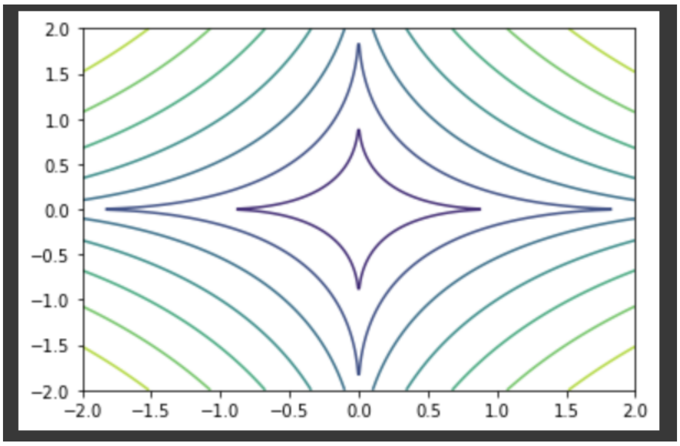
I did hyperparameter tuning of learning rate, regularization parameter and the number of training iterations (I tuned the number of training iterations because I was getting different accuracies when I was using different ones so I just thought I tune it to the best one.) by just having 3 for loops (4 in the case of SGD with decaying learning rate) and took the combination that gave the highest validation accuracy. Then I trained the model that had the highest validation accuracy on the entire dataset and tested it on the test set.

Question 4: A Bayesian Interpretation of Lasso

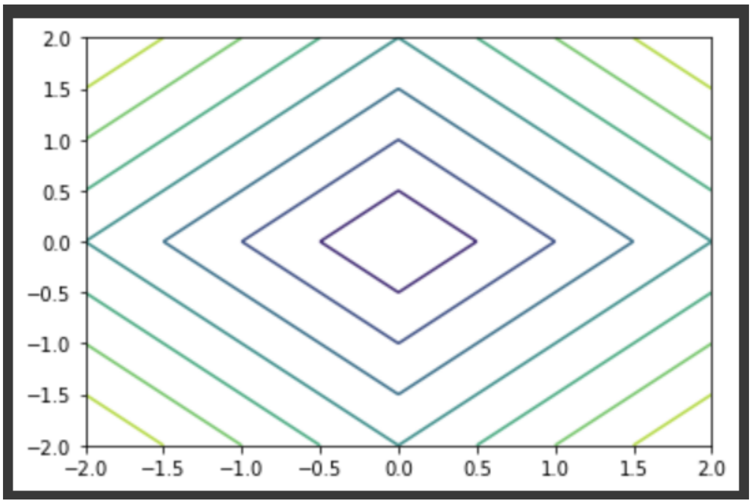
Question 5: L_1- regularization, L_2- regularization, and Sparsity

Part 1:

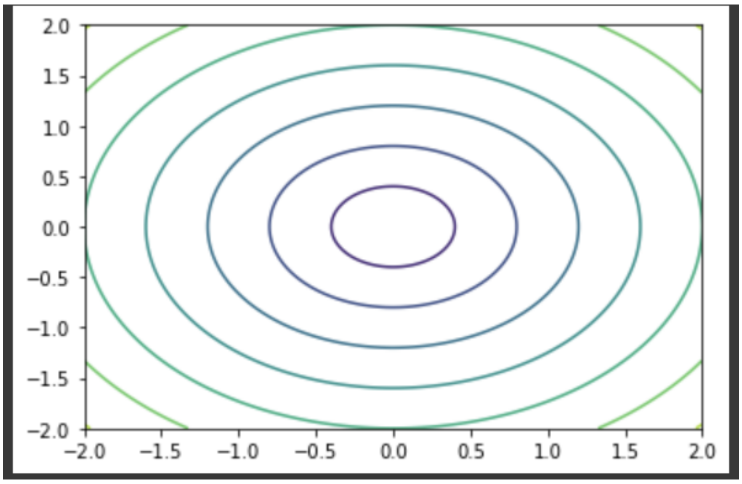
a) L_0.5



b) L_1



c) L_2



CODE APPENDIX:

Question 3: Wine Classification with Logistic Regression

Random Seed = 100

Preprocessing: Normalizing, shuffling and splitting:

```
import numpy as np
import scipy.io as sio
import scipy
import matplotlib.pyplot as plt
from save_csv import results_to_csv
from scipy.special import *
import math

# Random seed
np.random.seed(100)

data = sio.loadmat('data.mat')
# print(data)

train_data = data['X']
print("Main data: {}" .format(train_data.shape))
train_labels = data['y']
print("Main labels: {}" .format(train_labels.shape))
test_data = data['X_test']
print("Main test: {}" .format(test_data.shape))

# Shuffle the data and split and set aside 1000 samples aside for the validation
full_set = np.concatenate((train_data, train_labels), axis=1)
np.random.shuffle(full_set)

training_data_full = full_set[:5000, :-1]
training_labels = full_set[:5000, -1:].reshape(-1,)

validation_data_full = full_set[5000:, :-1]
validation_labels = full_set[5000:, -1:].reshape(-1,)

print("\nShapes after shuffle and split:")
print("Main data: {}" .format(training_data_full.shape))
print("Main labels: {}" .format(training_labels.shape))
print("Validation data: {}" .format(validation_data_full.shape))
print("Validation labels:{}" .format(validation_labels.shape))

# Begin by normalizing the data with each feature’s mean and standard deviation.
# You should use training data statistics to normalize both training and validation/test data.
# Then add a fictitious dimension. Whenever required, it is recommended that you tune
hyperparameter values with cross-validation.
```

```
# Function to normalize the training and test data with each feature’s mean and standard deviation
mean_f = None
std_f = None
def normalize(data, mean=None, standard_deviation=None):
   samples, features = data.shape
   if not mean:
       mean = data.mean(axis=0)
       standard_deviation = data.std(axis=0)
   data = (data - mean) / standard_deviation
   # Add a fictitious dimension by adding a vector of 1s at the end of the training data set.
   data = np.concatenate((data, np.ones((samples, 1))), axis=1)
   return data
# Normalize the training and test data
training_data = normalize(training_data_full)
print("\nAfter normalizing training")
print("Main data {}" .format(training_data.shape))
print(training_data)

test_data = normalize(test_data)
print("\nAfter normalizing test")
print("Test data {}" .format(test_data.shape))
print(test_data)
```

Part 2: Batch Gradient Descent Code:

```
#### QUESTION 3.2: Batch Gradient Descent Code.

# Batch Gradient Descent Code. Implement your batch gradient descent algorithm for logistic
regression and include your code here.

# Choose reasonable values for the regularization parameter and step size (learning rate),
specify your chosen values in the write-up, and train your model from question 3.1.

# Shuffle and split your data into training/validation sets and mention the random seed used
in the write-up.

# Plot the value of the cost function versus the number of iterations spent in training.

# Random seed = 100

def sigmoid_func(X, w):

   # print('hi')

   # print(X.shape)

   # # print(w.shape)

   # print(w.shape)

   return scipy.special.expit(np.matmul(X, w.T))

# Loss function of the logistic regression for BGD

def loss_func(data, label, weight, reg):

   regularizer = (reg / 2) * weight.T.dot(weight)

   # w = np.reshape(w, (w.shape[0], 1))
```

```
    # print(X.shape)
    # print(w.shape)
    loss_1 = np.dot(label.T, np.log(sigmoid_func(data, weight)))
    loss_2 = np.dot((1 - label).T, np.log(1 - sigmoid_func(data, weight)))
    # print(regularizer - loss_1 - loss_2)
    return regularizer - loss_1 - loss_2
# Update rule of BGD
def update_rule(data, label, weight, reg, lr):
    # print('Hoala')
    # print(X.T.shape)
    # print(y.shape)
    sigmoid = sigmoid_func(data, weight)
    # print('bro')
    # print((y - sigmoind.T).shape)
    # print(X.shape)
    gradient = np.matmul(data.T, (label - sigmoid))
    # print(gradient.shape)
    # print('hi')
    # print(w.shape)
    # w = np.reshape(w, (13,1))
    # print(w.shape)
    step = reg * weight - gradient
    return weight - (lr * step)
X_train = normalize(training_data_full)
# Function to train the BGD Logistic Regression
def train_BGD(num_iterations, X_train, y_train, w_train, regularization_param, lr):
    training_loss = []
    for iteration in range(num_iterations):
        loss = loss_func(X_train, y_train, w_train, regularization_param)
        next_w = update_rule(X_train, y_train, w_train, regularization_param, lr)
        # print(loss)
        if math.isnan(loss):
            loss = 0
        training_loss.append(int(loss))
        w_train = next_w
    return w_train, training_loss
print("\nSetting Hyperparameters for Batch Gradient Descent:")
```

```python
### Setting the hyperparameters of the BGD.

w_train = np.zeros(13)

num_iterations = [100, 500, 1000, 10000]

regularization_param = [0.001, 0.01, 0.1, 1.0, 10]

learningRates = [1e-7, 1e-6,1e-5, 1e-3, 0.1, 1, 10]

best_validation_accuracy_so_far = 0.0

best_hyperparameters_so_far = None

for lr in learningRates:

   for l in regularization_param:

       for i in num_iterations:

           batch_best_w, batch_tl = train_BGD(i, X_train, training_labels, w_train, l, lr)

           val_data = normalize(validation_data_full, mean_f, std_f)

           out_labels = sigmoid_func(val_data, batch_best_w)

           predicted_labels = (out_labels > 0.5).astype(np.int32)

           predicted_labels = predicted_labels.reshape(-1,)

           val_accuracy = (validation_labels == predicted_labels).astype(np.int32).mean()

           if val_accuracy > best_validation_accuracy_so_far:

               best_validation_accuracy_so_far = val_accuracy

               best_hyperparameters_so_far = (lr, l, i)

           print("Parameters{}: Validation accuracy: {}" .format(best_hyperparameters_so_far,
best_validation_accuracy_so_far))


best_validation_acc = best_validation_accuracy_so_far

best_hyperparams = best_hyperparameters_so_far

print("\nBest validation accuracy: ",best_validation_acc)

print("\nBest hyperparameters combo:(learning rate, regularization parameter, number of
iteration): ",best_hyperparams)

## Training BGD

print("\n---- Training Batch gradient Descent---")

w_train = np.zeros(13)

# Were set by hyperparameter tuning above

num_iterations = 10000

regularization_param = 0.001

lerningRate = 1e-5

batch_GD_best_w, batch_trainingloss = train_BGD(num_iterations, X_train, training_labels,
w_train, regularization_param, lerningRate)

# Plot the value of the cost function versus the number of iterations spent in training.

iterations = list(np.arange(1, num_iterations + 1, 1))

plt.figure(1)
```

```
plt.plot(iterations, batch_trainingloss, 'r-')

plt.title("BGD: Cost value vs Number of iterations")

plt.xlabel("Number of iterations")

plt.ylabel("Value of cost function")

plt.savefig('Batch Gradient Descent.png')

# plt.show()

out_labels = sigmoid_func(X_train, batch_GD_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)

predicted_labels = predicted_labels.reshape(-1,)

(training_labels == predicted_labels).astype(np.int32).mean()

print("Training accuracy: {}" .format((training_labels ==
predicted_labels).astype(np.int32).mean()))

val_data = normalize(validation_data_full, mean_f, std_f)

out_labels = sigmoid_func(val_data, batch_GD_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)

predicted_labels = predicted_labels.reshape(-1,)

(validation_labels == predicted_labels).astype(np.int32).mean()

print("Validation accuracy: {}" .format((validation_labels ==
predicted_labels).astype(np.int32).mean()))
```

Part 4: Stochastic Gradient Descent Code

```
#### QUESTION 3.4 : Stochastic Gradient Descent Code.

# Stochastic Gradient Descent Code. Implement your stochastic gradient descent algorithm for
logistic regression and include your code here.

# Choose a suitable value for the step size (learning rate), specify your chosen value in the
write-up, and run your SGD algorithm from question 3.3.

# Shuffle and split your data into training/validation sets and mention the random seed used
in the write-up.

# Plot the value of the cost function versus the number of iterations spent in training.

# Compare your plot here with that of question 3.2. Which method converges more quickly?
Briefly describe what you observe.

def sigmoid_single(x_i, w):

   return 1 / (1 + np.exp(-np.dot(x_i, w)))

# Update rule for SGD

def sgd_compute_update(x_i, y_i, w, reg, lr):

   sig = sigmoid_single(x_i.T,w)

   gradient = np.dot(x_i, (y_i-sig))

   step = reg * w - gradient

   return w - (lr * step)

X_train = normalize(training_data_full)

# Function to train the SGD Logistic Regression
```

```python
def train_SGD(num_iterations, X_train, y_train, w_train, regularization_param, lr):
    training_loss_history = []
    N, d = X_train.shape
    for curr_iter in range(num_iterations):
        training_sample = X_train[curr_iter % N]
        labels_sample = y_train[curr_iter % N]
        loss = loss_func(X_train, y_train, w_train, regularization_param)
        w_train = sgd_compute_update(training_sample, labels_sample, w_train, regularization_param, lr)
        training_loss_history.append(float(loss))

    return w_train, training_loss_history
print("Setting Hyperparameters for Stochastic Gradient Descent (SGD):")
### Setting the hyperparameters of the best performed model. Which is the  Stochastic Gradient Descent (SGD)
w_train = np.zeros(13)
num_iterations = [100, 500, 1000, 10000]
regularization_param = [0.001, 0.01, 0.1, 1.0, 10]
learningRates = [1e-7, 1e-6, 1e-5, 1e-3, 0.1, 1, 10]
best_validation_accuracy_so_far = 0.0
best_hyperparameters_so_far = None
for a in learningRates:
    for l in regularization_param:
        for i in num_iterations:
            batch_best_w, batch_tl = train_SGD(i, X_train, training_labels, w_train, l, a)
            data_val = normalize(validation_data_full, mean_f, std_f)
            out_labels = sigmoid_func(data_val, batch_best_w)
            pred_label = (out_labels > 0.5).astype(np.int32)
            pred_label= pred_label.reshape(-1,)
            val_accuracy = (validation_labels == pred_label).astype(np.int32).mean()
            if val_accuracy > best_validation_accuracy_so_far:
                best_validation_accuracy_so_far = val_accuracy
                best_hyperparameters_so_far = (a, l, i)
            print("Parameters{}: Validation accuracy: {}" .format(best_hyperparameters_so_far, best_validation_accuracy_so_far))
best_valudation_acc = best_validation_accuracy_so_far
best_hyperparams = best_hyperparameters_so_far
print("best validation accuracy: ",best_valudation_acc)
```

```python
print("best hyperparameters combo:(learning rate, regularization parameter, number of iteration): ",best_hyperparams)

## Training SGD

print("---- Training Stochastic Gradient Descent ---")

w_train = np.zeros(13)

# Set by hyperparameter tuning

regularization_param = 0.001

lr = 0.1

num_iterations = 10000

sgd_best_w, sgd_trainingloss = train_SGD(num_iterations, X_train, training_labels, w_train, regularization_param, lr)

iterations = list(np.arange(1, num_iterations + 1, 1))

plt.figure(2)

plt.plot(iterations, sgd_trainingloss, 'r-')

plt.title("SGD: Cost value vs Number of iterations")

plt.xlabel("Number of iterations")

plt.ylabel("Value of cost function")

plt.savefig('Stochastic Gradient Descent.png')

# plt.show()

out_labels = sigmoid_func(X_train, sgd_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)

predicted_labels= predicted_labels.reshape(-1,)

(training_labels == predicted_labels).astype(np.int32).mean()

print("Training accuracy: {}" .format((training_labels == predicted_labels).astype(np.int32).mean()))

out_labels = normalize(validation_data_full, mean_f, std_f)

out_labels = sigmoid_func(out_labels, sgd_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)

predicted_labels = predicted_labels.reshape(-1,)

(validation_labels == predicted_labels).astype(np.int32).mean()

print("Validation accuracy: {}" .format((validation_labels == predicted_labels).astype(np.int32).mean()))
```

Part 5: Stochastic Gradient Descent with decaying learning rate

```python
#### QUESTION 3-5: Stochastic Gradient Descent with decaying learning rate

# Instead of using a constant step size (learning rate) in SGD, you could use a step size that slowly shrinks from iteration to iteration.

# Run your SGD algorithm from question 3.3 with a step size εt = δ/t where t is the iteration number and δ is a hyperparameter you select
```

```
# empirically. Mention the value of δ chosen. Plot the value of cost function versus the
number of iterations spent in training.

# How does this compare to the convergence of your previous SGD code?

X_train = normalize(training_data_full)

# Function to train the SGD Logistic Regression with decaying learning rate.

def train_sgd_decay(num_iterations, X_train, y_train, w_train, lr, regularization_param,
delta):

   training_loss_history = []

   N, d = X_train.shape

   for iteration in range(num_iterations):

       lr = delta / (iteration + 1)

       X_sample = X_train[iteration % N]

       y_sample = y_train[iteration % N]

       loss = loss_func(X_train, y_train, w_train, regularization_param)

       w_train = sgd_compute_update(X_sample, y_sample, w_train, regularization_param, lr)

       training_loss_history.append(float(loss))

   return w_train, training_loss_history


print("\nSetting Hyperparameters for Stochastic Gradient Descent (SGD) with decaying learning
rate:")

### Setting the hyperparameters of the best performed model. Which is the  Stochastic Gradient
Descent (SGD)

w_train = np.zeros(13)

deltas = [0.1, 0.2, 0.5, 0.7, 0.9, 1.0, 2, 5]

num_iterations = [100, 500, 1000, 10000, 20000]

regularization_param = [ 0.001, 0.01, 0.1, 1.0, 10]

learningRates = [1e-7, 1e-6, 1e-5, 1e-3, 0.1, 1, 10]

best_validation_accuracy_so_far = 0.0

best_hyperparameters_so_far = None

for d in deltas:

   for a in learningRates:

       for l in regularization_param:

           for i in num_iterations:

               batch_best_w, batch_tl = train_sgd_decay(i, X_train, training_labels, w_train,
lr, l, d)

               val_data = normalize(validation_data_full, mean_f, std_f)

               out_labels = sigmoid_func(val_data, batch_best_w)

               predicted_labels = (out_labels > 0.5).astype(np.int32)

               predicted_labels = predicted_labels.reshape(-1,)

               val_accuracy = (validation_labels == predicted_labels).astype(np.int32).mean()
```

```
                if val_accuracy > best_validation_accuracy_so_far:

                    best_validation_accuracy_so_far = val_accuracy

                    best_hyperparameters_so_far = (a, l, i, d)

                print("Parameters{}: Validation accuracy: {}"
.format(best_hyperparameters_so_far, best_validation_accuracy_so_far))


best_val_acc = best_validation_accuracy_so_far

best_hyperparams = best_hyperparameters_so_far

print("best validation accuracy: ",best_val_acc)

print("best hyperparameters combo:(learning rate, regularization parameter, number of
iteration, delta): ",best_hyperparams)

## Training SGD with decaying learning rate

print("---- Training Stochastic Gradient Descent with decaying learning rate ---")

w_train = np.zeros(13)

# Set by hyperparameter tuning

regularization_param = 0.01

lr = 1e-07

delta = 2.0

num_iterations = 20000

sgd_decay_best_w, sgd_decay_tl = train_sgd_decay(num_iterations, X_train, training_labels,
w_train, lr, regularization_param, delta)


iterations = list(np.arange(1, num_iterations+1, 1))

plt.figure(3)

plt.plot(iterations, sgd_decay_tl, 'r-')

plt.title("SGD with decreasing Learning rate: cost function vs iterations")

plt.xlabel("Number of iterations")

plt.ylabel("Value of cost function")

plt.savefig('SGD with decreasing learning rate.png')

# plt.show()

out_labels = sigmoid_func(X_train, sgd_decay_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)

predicted_labels = predicted_labels.reshape(-1,)

(training_labels == predicted_labels).astype(np.int32).mean()

print("Training accuracy: {}" .format((training_labels ==
predicted_labels).astype(np.int32).mean()))

out_labels = normalize(validation_data_full, mean_f, std_f)

out_labels = sigmoid_func(out_labels, sgd_decay_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)
```

```
predicted_labels = predicted_labels.reshape(-1,)

(validation_labels == predicted_labels).astype(np.int32).mean()

print("Validation accuracy: {}" .format((validation_labels ==
predicted_labels).astype(np.int32).mean()))
```

Part 6: Kaggle

```
#### QUESTION 3.6: Kaggle

# Kaggle. Train your best classifier on the entire training set and submit your prediction on
the test sample

# points to Kaggle. As always for Kaggle competitions, you are welcome to add or remove
features, tweak the algorithm,

# and do pretty much anything you want to improve your Kaggle leaderboard performance except
that you may not replace or

# augment logistic regression with a wholly different learning algorithm. Your code should
output the predicted labels in a CSV file.

## My Best classifier is the batch gradient Descent classifier which got 99.5% validation
accuracy

## Training with the whole training set

X_train = normalize(training_data_full)

print("\n---- Training Best Performing Batch gradient Descent for testing---")

w_train = np.zeros(13)

#set by hyperparameter tuning

num_iterations = 10000

regularization_param = 0.001

lr = 1e-5

batch_best_w, batch_train_loss = train_BGD(num_iterations, X_train, training_labels, w_train,
regularization_param, lr)

test_data_full = data["X_test"]

X_test = normalize(test_data_full, mean_f, std_f)

out_labels = sigmoid_func(X_test, batch_best_w)

predicted_labels = (out_labels > 0.5).astype(np.int32)

predicted_labels = predicted_labels.reshape(-1,)

results_to_csv(predicted_labels)

print("Tested the data and Saved the predictions")
```