Iris\Iris_main.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from Iris_functions import *
import time

# Parameters
D = 4                               # Number of features
C = 3                               # Number of classes
N_train = 30                        # Number of training data
N_test  = 20                        # Number of test data
first_30_to_train = True            # Use first 30 data points for training and last 20 for
testing
disabled_features = []              # Which features to disable
D = D - len(disabled_features)      # Update D

# Plot parameters
visualize_histogram = True         # Plot histograms of the data
visualize_confusion_matrix = False  # Plot confusion matrix
visualize_MSE = False               # Plot MSE vs iteration

# Load seperate iris data
setosa      = pd.read_csv('Iris_raw_data/class_1.csv', header=0)
versicolour = pd.read_csv('Iris_raw_data/class_2.csv', header=0)
verginica   = pd.read_csv('Iris_raw_data/class_3.csv', header=0)

# Remove unwanted features
setosa      = remove_features(setosa, disabled_features)
versicolour = remove_features(versicolour, disabled_features)
verginica   = remove_features(verginica, disabled_features)

# Create training and test data
train_data, test_data = create_train_test_data(setosa, versicolour, verginica, N_train, N_test,
first_30_to_train)

# Normalizing the data
max_features_val = np.array([train_data[:,i].max() for i in range(D)])
normal_train_data = train_data/max_features_val

# Save max values to file
np.savetxt("Iris_max_values.txt", max_features_val)

# Target vectors one hot encoded
t1 = np.array([1, 0, 0])
t2 = np.array([0, 1, 0])
t3 = np.array([0, 0, 1])
label_train = np.vstack((np.tile(t1, (N_train, 1)), np.tile(t2, (N_train, 1)), np.tile(t3,
(N_train, 1))))
label_test = np.vstack((np.tile(t1, (N_test, 1)), np.tile(t2, (N_test, 1)), np.tile(t3, (N_test,
1))))

# Training the LDC
W = np.zeros((C, D+1))    # Initial Weights
```

```
training = True
iterations = 1000
learning_rate = 0.005

MSE_list = []
print("\nStarting training")
start_time = time.time()

for i in range(iterations):
    grad_W_MSE = 0
    MSE = 0
    for i in range(C*N_train):
        # Using 3.2 in compendium
        x_k  = np.array(train_data[i, :])   # Get data point
        x_k = np.append(x_k, 1)             # Add bias
        z_k = W @ x_k                       # Calculate weighted sum
        g_k = sigmoid(z_k)                  # Activation function
        t_k = label_train[i, :]             # Get data point label
        grad_W_MSE += grad_W_MSE_func(g_k, t_k, x_k, D)
        MSE += 0.5*(g_k-t_k).T @ (g_k-t_k)

    MSE_list.append(MSE)
    W = W - learning_rate*grad_W_MSE

end_time = time.time()
elapsed_time = round(end_time - start_time, 2)
print("Training time: ", elapsed_time, "s")
print("Training done\n")

# Set print options
np.set_printoptions(precision=2, suppress=True)
print("Weights:")
print(W)

# Plot functions
if visualize_MSE:
    plt.plot(MSE_list)
    plt.title("MSE vs iteration\nLearning rate: " + str(learning_rate) + ", Iterations: " +
str(iterations) + ", Time: " + str(elapsed_time) + "s")
    plt.xlabel("Iteration")
    plt.ylabel("MSE")
    plt.show()

# Find confusion matrix for training data
confusion_matrix_train = np.zeros((C, C))
for i in range(C*N_train):
    x_k  = np.array(train_data[i, :])
    x_k = np.append(x_k, 1)
    z_k = W @ x_k
    g_k = sigmoid(z_k)
    t_k = label_train[i, :]
    if np.argmax(g_k) == np.argmax(t_k):
        confusion_matrix_train[np.argmax(t_k), np.argmax(t_k)] += 1
    else:
        confusion_matrix_train[np.argmax(t_k), np.argmax(g_k)] += 1
```

```python
print("Confusion matrix for training data: ")
print(confusion_matrix_train)

# Find confusion matrix for test data
confusion_matrix_test = np.zeros((C, C))
for i in range(C*N_test):
    x_k  = np.array(test_data[i, :])
    x_k = np.append(x_k, 1)
    z_k = W @ x_k
    g_k = sigmoid(z_k)
    t_k = label_test[i, :]
    if np.argmax(g_k) == np.argmax(t_k):
        confusion_matrix_test[np.argmax(t_k), np.argmax(t_k)] += 1
    else:
        confusion_matrix_test[np.argmax(t_k), np.argmax(g_k)] += 1

print("Confusion matrix for test data:")
print(confusion_matrix_test)

# Print accuracy for traing and test data
accuracy_train = np.sum(np.diag(confusion_matrix_train))/np.sum(confusion_matrix_train)
accuracy_test = np.sum(np.diag(confusion_matrix_test))/np.sum(confusion_matrix_test)
print_accuracy_for_confusion_matrix(confusion_matrix_train,"training")
print_accuracy_for_confusion_matrix(confusion_matrix_test,"test")

# END OF TASK 1
#-----------------------------------------------------------------------------------------------------------------
# START OF TASK 2

# Plotting confusion matrices for training and test data
error_rate_train_percent = round((1 - accuracy_train)*100, 2)
error_rate_test_percent = round((1 - accuracy_test)*100, 2)

label_names = ["Setosa", "Versicolour", "Verginica"]
if visualize_confusion_matrix:
    plot_confusion_matrix(confusion_matrix_train,confusion_matrix_test, label_names,
first_30_to_train,error_rate_train_percent,error_rate_test_percent)

if visualize_histogram and D == 4:
    #Plot 3 histograms for feature x for all classes
    plot_histograms(train_data,N_train)

# Show all figures
plt.show()
```