CLASSIFICATION WITH MULTILAYER PERCEPTRON MLP

Neural Networks

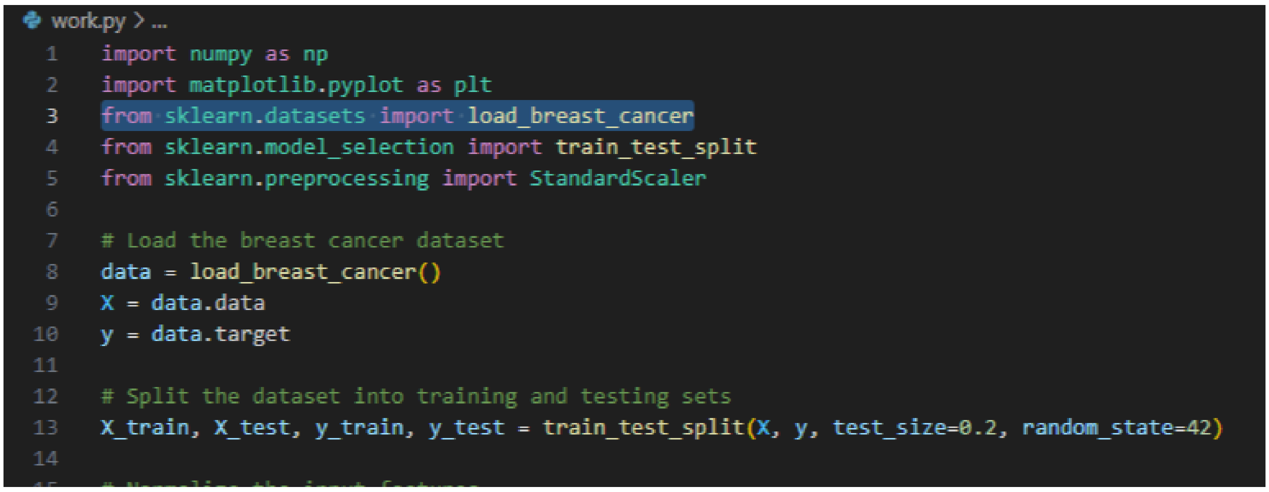
Presented to: Dr Huda Hakami
Prepared by: Ebtsam ALyzidi

MULTI-LAYER PERCEPTRON FOR CLASSIFICATION

Tasks:

-I loaded the instances in a different way, from a different library without loading the data
Once loaded, we can split the data into training and test sets

We will use the train_test_split() function from scikit-learn and use 80% of the data for training and 20% for testing.


work.py > ...

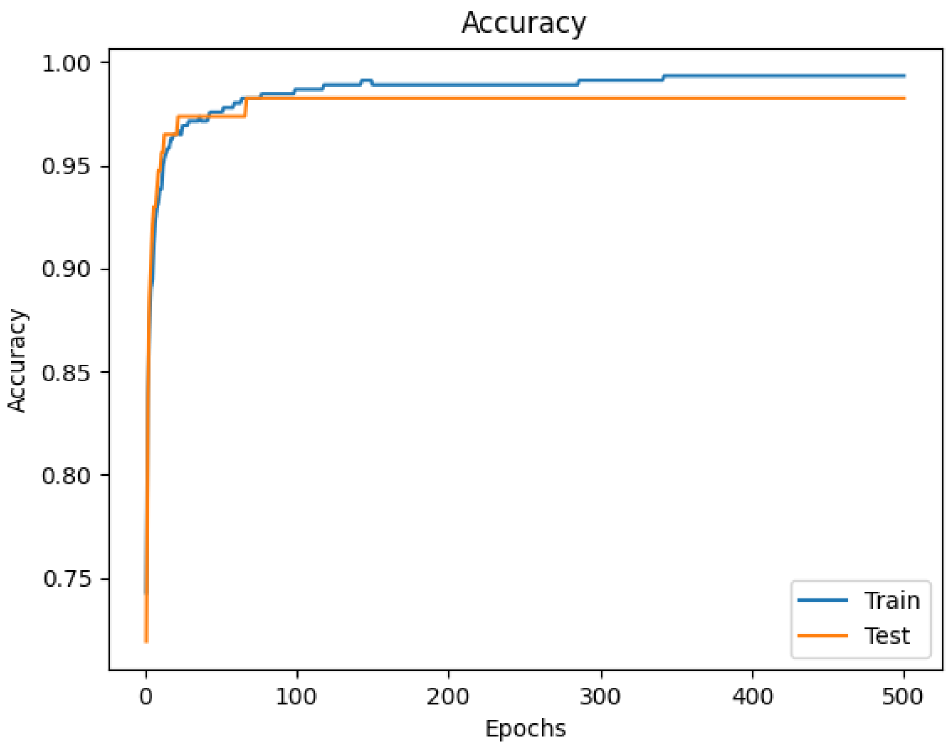
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

Implement the multi-layer perceptron
A network is organized into layers. The input layer is really just a row from our training dataset. The first real layer is the hidden layer. This is followed by the output layer that has one neuron for each class value.

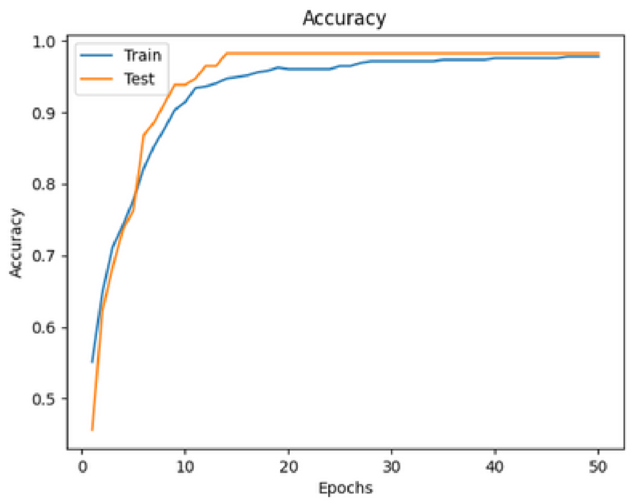
It is good practice to initialize the network weights and biases to small random numbers

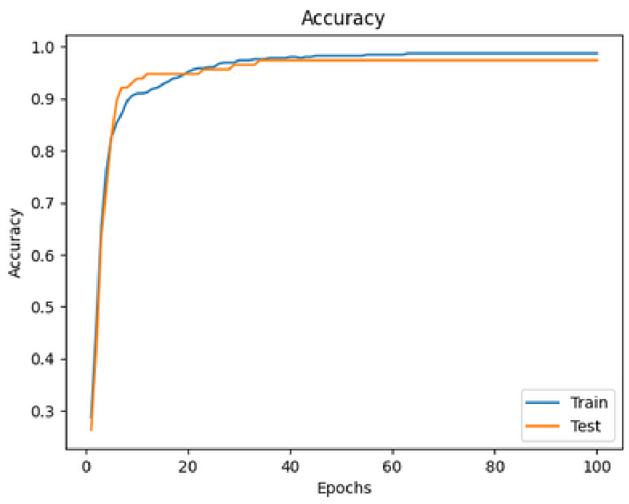
The first step is Define the activation function (sigmoid) and Gradient Descent as function
Second, create a loop for each epochs that calculates Forward propagation, Backward propagation, Update the weights and biases, and Evaluate the training-testing set ,respectively

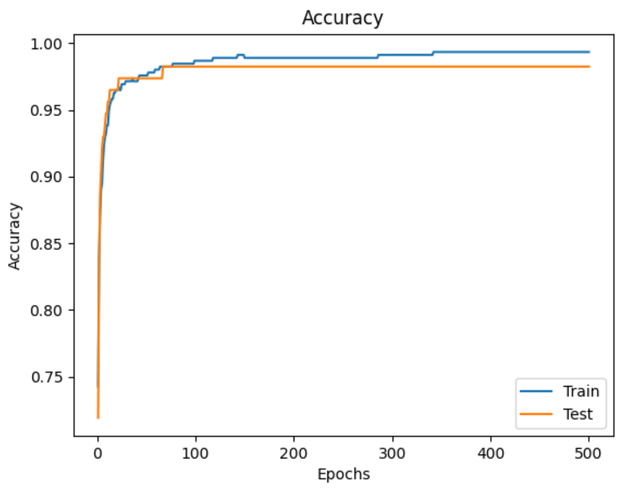
Accuracy data is also represented in a figure

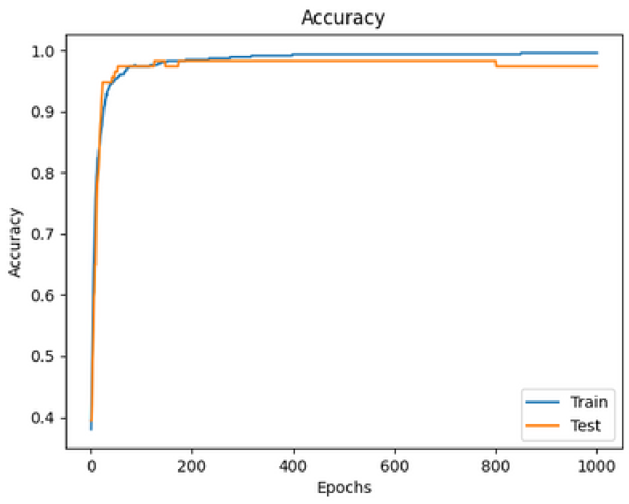


I have tried a large number of iterations, and I do not think that there is an ideal number of iterations, but the more it is, the greater the accuracy and the change becomes minimal



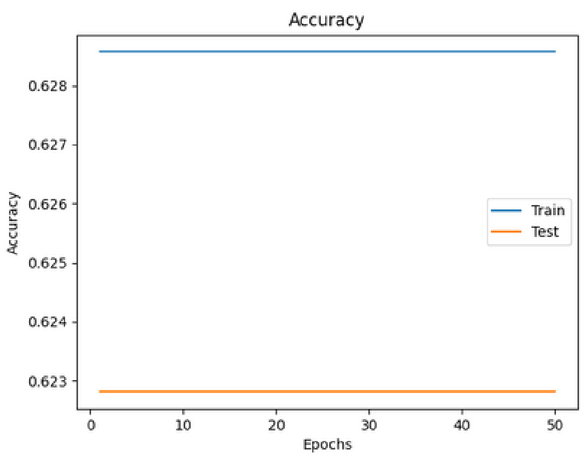


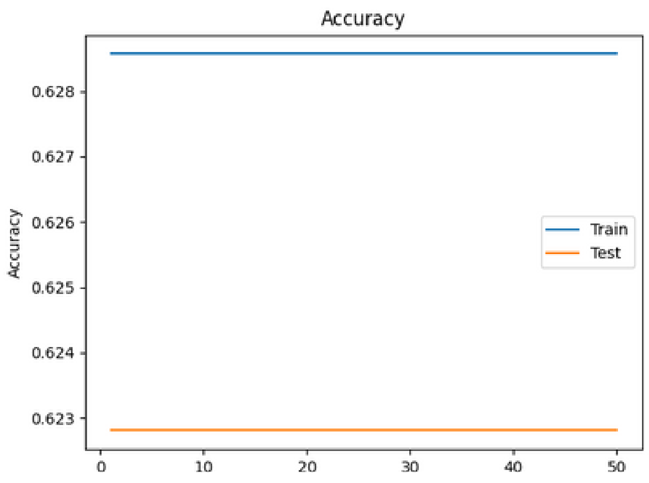




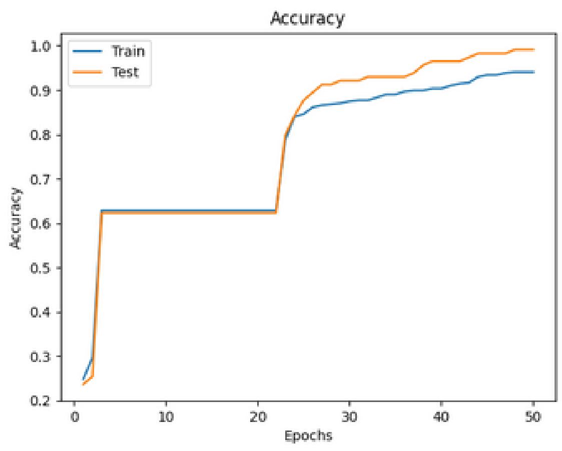
The learning rate hyperparameter controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated, such as at the end of each batch of training

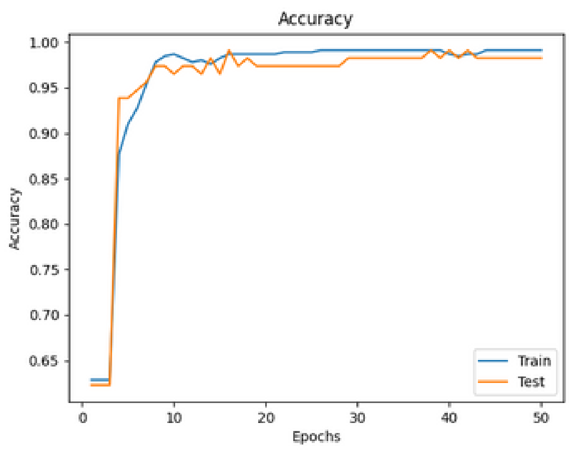
Train the MLP with one hidden layer with different learning rates [1.0, 0.5, 0.1, 0.01] and plot the train/test loss for each value.
in 1.0 and 0.5 there is no change -i think the problem from me





here when use 0.1 and 0.001 The learning rate seemed to be affected





References:

https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/

https://www.askpython.com/python/examples/backpropagation-in-python

https://www.google.com.sa/books/edition/Learning_OpenCV_4_Computer_Vision_with_P/ef_RDwAAQBAJ?hl=ar&gbpv=0

https://www.google.com.sa/books/edition/Proceedings_of_International_Conference/PzItEAAAQBAJ?hl=ar&gbpv=0