Exercise 1: Use MP neurons to build a simple neural network that performs logical operations OR, AND and NOR

```
#AND GATE

x1 = [0,0,1,1]
x2 = [0,1,0,1]
w1 = [1,1,1,1]
w2 = [1,1,1,1]

t = 2

print("x1   x2   w1   w2   t   0")
for i in range(len(x1)):
  if(x1[i]*w1[i] + x2[i]*w2[i]) >= t:
    print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
  else:
    print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1   x2   w1   w2   t   0
0     0     1     1     2     0
0     1     1     1     2     0
1     0     1     1     2     0
1     1     1     1     2     1
```

```
#OR GATE

x1 = [0,0,1,1]
x2 = [0,1,0,1]
w1 = [1,1,1,1]
w2 = [1,1,1,1]

t = 1

print("x1   x2   w1   w2   t   0")
for i in range(len(x1)):
  if(x1[i]*w1[i] + x2[i]*w2[i]) >= t:
    print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
  else:
    print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1   x2   w1   w2   t   0
0     0     1     1     1     0
0     1     1     1     1     1
1     0     1     1     1     1
1     1     1     1     1     1
```

```
# NOR GATE

x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
```

```
w1 = [-1, -1, -1, -1]
w2 = [-1, -1, -1, -1]

t = 0

print("x1   x2   w1   w2   t   0")
for i in range(len(x1)):
    if (x1[i] * w1[i] + x2[i] * w2[i]) >= t:
        print(x1[i], '   ', x2[i], '   ', w1[i], '   ', w2[i], '   ', t, '   ', 1)
    else:
        print(x1[i], '   ', x2[i], '   ', w1[i], '   ', w2[i], '   ', t, '   ', 0)
```

```
x1   x2   w1   w2   t   0
0     0     -1     -1     0     1
0     1     -1     -1     0     0
1     0     -1     -1     0     0
1     1     -1     -1     0     0
```

Exercise 2: Implement an MP neuron for a binary classification problem using a breast cancer dataset.

• Analyze the effects of scaling on MP Neuron's decision-making process and accuracy. Apply different scaling techniques (min-max normalization, standardization) to the breast cancer dataset features. Train the MP Neuron with these scaled features and compare the model's performance with unscaled data.

• Compare the MP Neuron model's performance with a logistic regression model in accuracy.

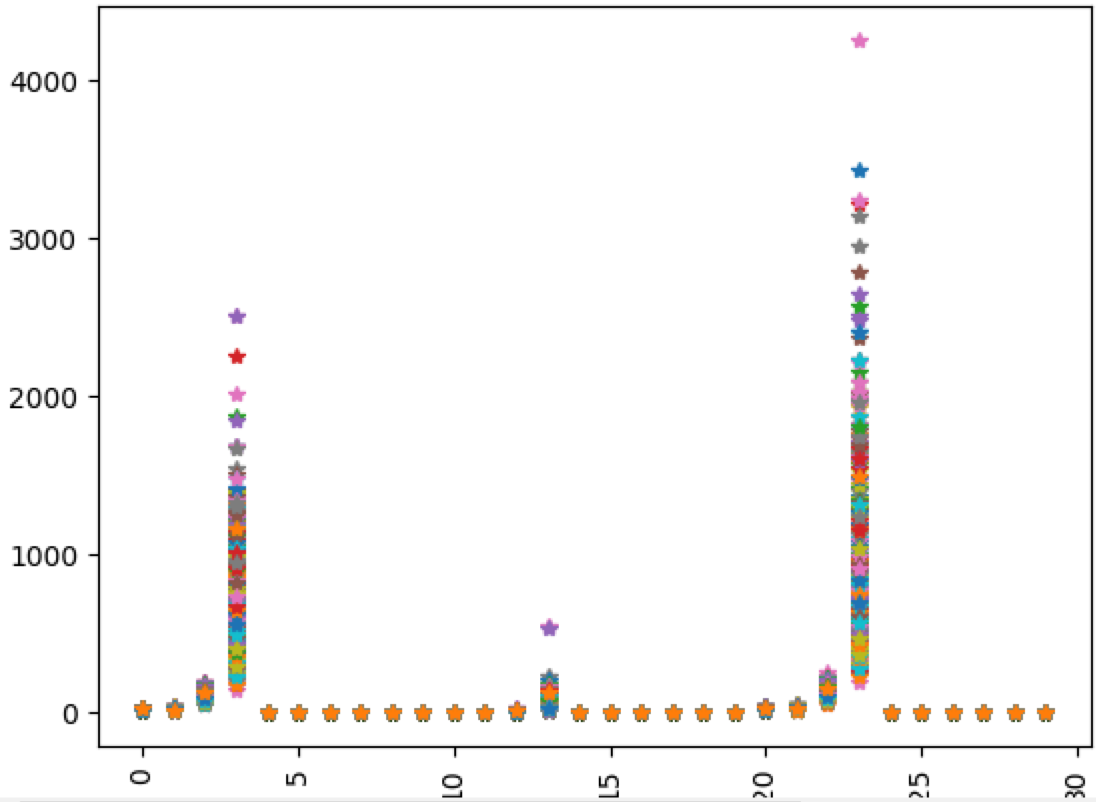
```
##LOAD DATA

import sklearn.datasets
breast_cancer = sklearn.datasets.load_breast_cancer()
X = breast_cancer.data
Y = breast_cancer.target
print(X.shape,Y.shape)
```

```
(569, 30) (569,)
```

```
#Test Train Split
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.1,stratify=Y,random_stat
print(Y.mean(),Y_test.mean(),Y_train.mean())
```

```
0.6274165202108963 0.631578947368421 0.626953125
```

```
import matplotlib.pyplot as plt
plt.plot(X_train.T,'*')
plt.xticks(rotation='vertical')
plt.show()
```




```
##IMPLEMENTING THE MP NEURON

class MPNeuron:
    def __init__(self, threshold=10):
        self.threshold = threshold

    def predict(self, X):
        # Binary step function based on threshold
        return np.where(np.sum(X, axis=1) >= self.threshold, 1, 0)
```

```
##APPLYING SCALING TECHNIQUES

from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-max normalization
scaler_minmax = MinMaxScaler()
X_train_minmax = scaler_minmax.fit_transform(X_train)
X_test_minmax = scaler_minmax.transform(X_test)

# Standardization
scaler_standard = StandardScaler()
X_train_standard = scaler_standard.fit_transform(X_train)
X_test_standard = scaler_standard.transform(X_test)
```

```
##COMPARING

import numpy as np
from sklearn.metrics import accuracy_score
```

```
mp_neuron = MPNeuron(threshold=10)

# Unscaled data
y_pred_train_unscaled = mp_neuron.predict(X_train)
y_pred_test_unscaled = mp_neuron.predict(X_test)

# Min-max scaled data
y_pred_train_minmax = mp_neuron.predict(X_train_minmax)
y_pred_test_minmax = mp_neuron.predict(X_test_minmax)

# Standardized data
y_pred_train_standard = mp_neuron.predict(X_train_standard)
y_pred_test_standard = mp_neuron.predict(X_test_standard)


accuracy_train_unscaled = accuracy_score(Y_train, y_pred_train_unscaled)
accuracy_test_unscaled = accuracy_score(Y_test, y_pred_test_unscaled)

accuracy_train_minmax = accuracy_score(Y_train, y_pred_train_minmax)
accuracy_test_minmax = accuracy_score(Y_test, y_pred_test_minmax)

accuracy_train_standard = accuracy_score(Y_train, y_pred_train_standard)
accuracy_test_standard = accuracy_score(Y_test, y_pred_test_standard)

print("MP Neuron Accuracy (Unscaled Data) - Train: ", accuracy_train_unscaled, " Test: ",
print("MP Neuron Accuracy (Min-Max Scaled) - Train: ", accuracy_train_minmax, " Test: ",
print("MP Neuron Accuracy (Standardized) - Train: ", accuracy_train_standard, " Test: ",
```

```
MP Neuron Accuracy (Unscaled Data) - Train:  0.626953125  Test:  0.631578947368421
MP Neuron Accuracy (Min-Max Scaled) - Train:  0.216796875  Test:  0.21052631578947367
MP Neuron Accuracy (Standardized) - Train:  0.134765625  Test:  0.12280701754385964
```

```
##LOGISTIC REGRESSION

from sklearn.linear_model import LogisticRegression

# Logistic regression model
logistic_reg = LogisticRegression(max_iter=10000)

# Train and test on unscaled data
logistic_reg.fit(X_train, Y_train)
y_pred_train_lr_unscaled = logistic_reg.predict(X_train)
y_pred_test_lr_unscaled = logistic_reg.predict(X_test)

# Train and test on min-max scaled data
logistic_reg.fit(X_train_minmax, Y_train)
y_pred_train_lr_minmax = logistic_reg.predict(X_train_minmax)
y_pred_test_lr_minmax = logistic_reg.predict(X_test_minmax)

# Train and test on standardized data
```

```
logistic_reg.fit(X_train_standard, Y_train)
y_pred_train_lr_standard = logistic_reg.predict(X_train_standard)
y_pred_test_lr_standard = logistic_reg.predict(X_test_standard)

# Calculate accuracies for logistic regression
accuracy_train_lr_unscaled = accuracy_score(Y_train, y_pred_train_lr_unscaled)
accuracy_test_lr_unscaled = accuracy_score(Y_test, y_pred_test_lr_unscaled)

accuracy_train_lr_minmax = accuracy_score(Y_train, y_pred_train_lr_minmax)
accuracy_test_lr_minmax = accuracy_score(Y_test, y_pred_test_lr_minmax)

accuracy_train_lr_standard = accuracy_score(Y_train, y_pred_train_lr_standard)
accuracy_test_lr_standard = accuracy_score(Y_test, y_pred_test_lr_standard)

print("\nLogistic Regression Accuracy (Unscaled Data) - Train: ", accuracy_train_lr_unsca
print("Logistic Regression Accuracy (Min-Max Scaled) - Train: ", accuracy_train_lr_minmax
print("Logistic Regression Accuracy (Standardized) - Train: ", accuracy_train_lr_standard
```

```
Logistic Regression Accuracy (Unscaled Data) - Train:  0.958984375  Test:  0.94736842
Logistic Regression Accuracy (Min-Max Scaled) - Train:  0.97265625  Test:  0.96491228
Logistic Regression Accuracy (Standardized) - Train:  0.990234375  Test:  0.98245614
```

Exercise 3: Implement Perceptron for Breast Cancer Classification

a. Load and Explore the Dataset

b. Split the Data into Training and Testing Sets

c. Standardize the Features : Standardize the features (i.e., transform them to have a mean of 0 and a variance of 1) to ensure faster convergence of the Perceptron algorithm.

d. Train the Perceptron Model

e. Make Predictions

f. Evaluate the Model

```
import sklearn.datasets
import numpy as np
from sklearn.metrics import accuracy_score
```

Loading dataset

```
breast_cancer = sklearn.datasets.load_breast_cancer()
```

```
X = breast_cancer.data
Y = breast_cancer.target
```

```
print(X)
print(Y)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

```
print(X.shape, Y.shape)
```

```
(569, 30) (569,)
```

```
import pandas as pd
```

```
data = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
```

```
data['class'] = breast_cancer.target
```

```
data.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

5 rows × 31 columns

```
data.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	conca
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.00
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.08
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.07
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.00
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.02
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.06
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.13
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.42

8 rows × 31 columns

```
print(data['class'].value_counts())
```

```
class
1    357
0    212
Name: count, dtype: int64
```

```
print(breast_cancer.target_names)
```

```
['malignant' 'benign']
```

```
data.groupby('class').mean()
```

class	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mear concavity
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058

2 rows × 30 columns

Train test split

```
from sklearn.model_selection import train_test_split
```

```
X = data.drop('class', axis=1)
Y = data['class']
```

```
type(X)
```

```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None,
dtype: Dtype | None=None, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y)
```

```
print(Y.shape, Y_train.shape, Y_test.shape)
```

```
(569,) (426,) (143,)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
```

```
print(Y.mean(), Y_train.mean(), Y_test.mean())
```

```
0.6274165202108963 0.630859375 0.5964912280701754
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, stratify = Y)
```

```
print(X_train.mean(), X_test.mean(), X.mean())
```

```
mean area                  692.014035
mean smoothness              0.094302
mean compactness             0.103559
mean concavity               0.097553
mean concave points          0.052743
mean symmetry                0.175105
mean fractal dimension       0.061632
radius error                 0.381686
texture error                1.167325
perimeter error              2.671591
area error                  37.450965
smoothness error             0.006856
compactness error            0.025942
concavity error              0.035304
concave points error         0.012872
symmetry error               0.019152
fractal dimension error      0.003660
worst radius                16.564561
worst texture               25.510526
worst perimeter            108.950877
worst area                 903.733333
worst smoothness             0.129407
worst compactness            0.245783
worst concavity              0.289601
worst concave points         0.122566
worst symmetry               0.279516
worst fractal dimension      0.081522
dtype: float64 mean radius                 14.127292
mean texture                19.289649
mean perimeter              91.969033
mean area                  654.889104
mean smoothness              0.096360
mean compactness             0.104341
mean concavity               0.088799
mean concave points          0.048919
mean symmetry                0.181162
mean fractal dimension       0.062798
radius error                 0.405172
texture error                1.216853
perimeter error              2.866059
area error                  40.337079
smoothness error             0.007041
compactness error            0.025478
concavity error              0.031894
concave points error         0.011796
symmetry error               0.020542
fractal dimension error      0.003795
worst radius                16.269190
worst texture               25.677223
worst perimeter            107.261213
worst area                 880.583128
worst smoothness             0.132369
worst compactness            0.254265
worst concavity              0.272188
worst concave points         0.114606
worst symmetry               0.290076
worst fractal dimension      0.083946
dtype: float64
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, stratify = Y, ra
```

```
print(X_train.mean(), X_test.mean(), X.mean())
```

```
mean radius                 14.058656
mean texture                19.309668
mean perimeter              91.530488
mean area                  648.097266
mean smoothness              0.096568
mean compactness             0.105144
mean concavity               0.089342
mean concave points          0.048892
mean symmetry                0.181961
mean fractal dimension       0.062979
radius error                 0.403659
texture error                1.206856
perimeter error              2.861173
area error                  39.935506
smoothness error             0.007067
compactness error            0.025681
concavity error              0.032328
concave points error         0.011963
symmetry error               0.020584
fractal dimension error      0.003815
worst radius                16.194275
worst texture               25.644902
worst perimeter            106.757715
worst area                 871.647852
worst smoothness             0.132592
worst compactness            0.257415
worst concavity              0.275623
worst concave points         0.115454
worst symmetry               0.291562
worst fractal dimension      0.084402
dtype: float64 mean radius                 14.743807
mean texture                19.109825
mean perimeter              95.908246
mean area                  715.896491
mean smoothness              0.094496
mean compactness             0.097130
mean concavity               0.083923
mean concave points          0.049159
mean symmetry                0.173981
mean fractal dimension       0.061169
radius error                 0.418767
texture error                1.306656
perimeter error              2.909946
area error                  43.944193
smoothness error             0.006809
compactness error            0.023659
concavity error              0.027989
concave points error         0.010293
symmetry error               0.020169
fractal dimension error      0.003618
worst radius                16.942105
worst texture               25.967544
worst perimeter            111.783860
worst area                 960.843860
```

```
worst smoothness           0.130357
worst compactness          0.225973
worst concavity            0.241340
worst concave points       0.106994
```

```
import matplotlib.pyplot as plt
```

Perceptron Class

Defines a Perceptron model with methods for initialization, prediction, and fitting the model to the data.

Initializes the weights (w) and bias (b) of the Perceptron to None

Computes the dot product of the weights and the input vector. If the result is greater than or equal to the bias, it returns 1 (positive class); otherwise, it returns 0 (negative class).

Applies the model to each input in the dataset X to generate predictions for an entire batch.

Trains the Perceptron model using the input features X and labels Y over a specified number of epochs with a learning rate lr. Adjusts the weights and bias based on prediction errors, striving to increase the model's accuracy over iterations.

```
X_train = X_train.values
X_test = X_test.values
```

$$y = 1, \text{if} \sum_i w_i x_i >= b$$

$$y = 0, \text{otherwise}$$

```
print (Y_train)
```

```
430    0
48     1
105    0
467    1
547    1
      ..
201    0
183    1
285    1
49     1
161    0
Name: class, Length: 512, dtype: int64
```

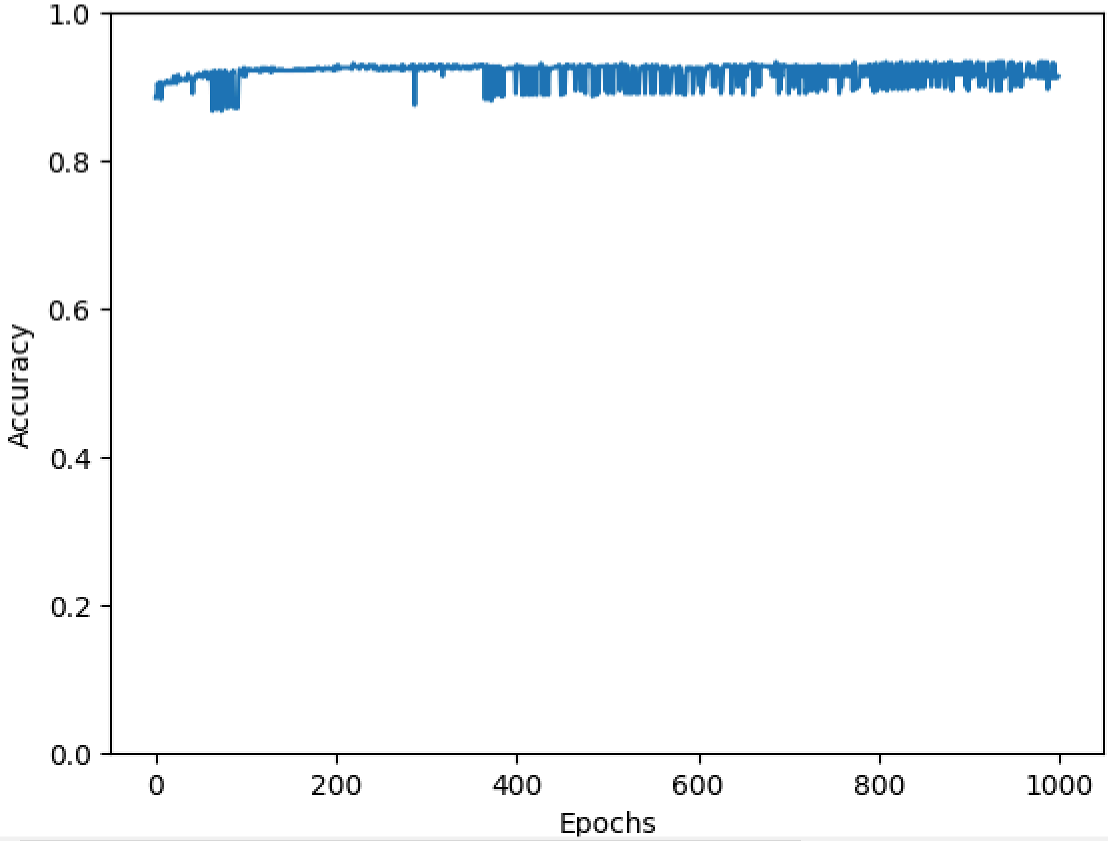
```
class Perceptron:

  def __init__ (self):
    self.w = None
    self.b = None
```

```
  def model(self, x):
    return 1 if (np.dot(self.w, x) >= self.b) else 0

  def predict(self, X):
    Y = []
    for x in X:
      result = self.model(x)
      Y.append(result)
    return np.array(Y)

  def fit(self, X, Y, epochs = 1, lr = 1):

    self.w = np.ones(X.shape[1])
    self.b = 0

    accuracy = {}
    max_accuracy = 0


    for i in range(epochs):
      for x, y in zip(X, Y):
        y_pred = self.model(x)
        if y == 1 and y_pred == 0:
          self.w = self.w + lr * x
          self.b = self.b - lr * 1
        elif y == 0 and y_pred == 1:
          self.w = self.w - lr * x
          self.b = self.b + lr * 1


      accuracy[i] = accuracy_score(self.predict(X), Y)
      if (accuracy[i] > max_accuracy):
        max_accuracy = accuracy[i]
        chkptw = self.w
        chkptb = self.b

    self.w = chkptw
    self.b = chkptb

    print(max_accuracy)


    plt.plot(np.array(list(accuracy.values())).astype(float))
    plt.ylim([0, 1])
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.show()
```

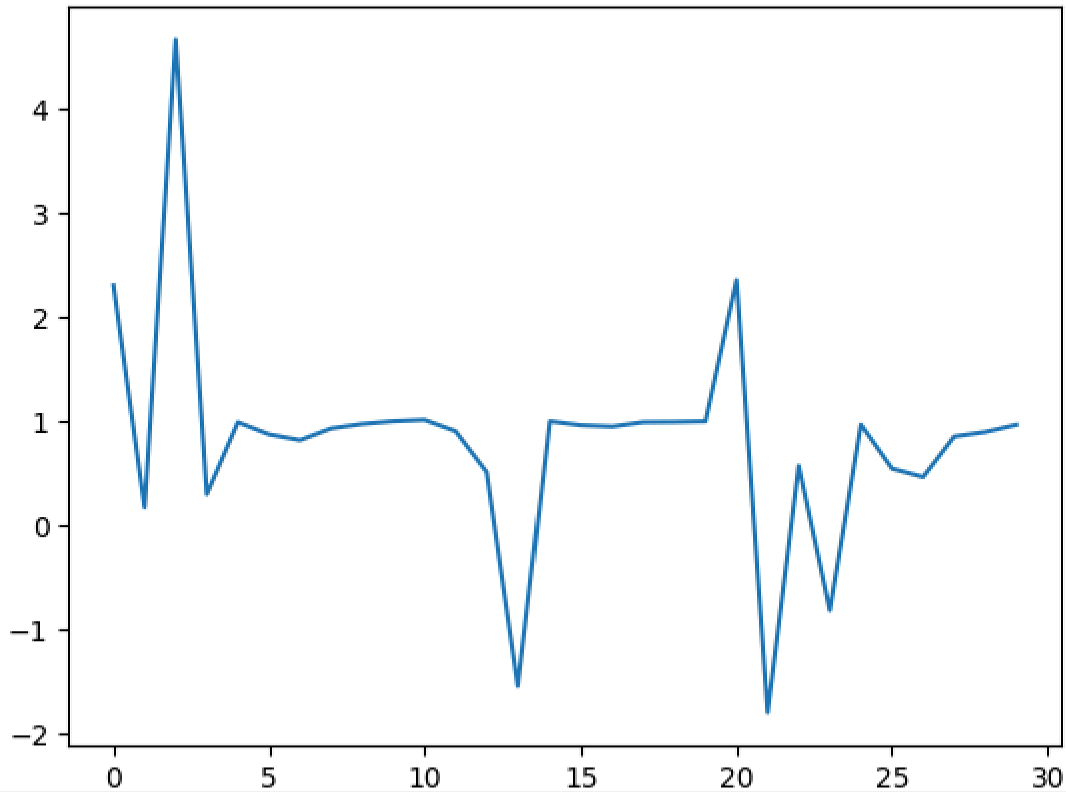
```
perceptron = Perceptron()
```

```
perceptron.fit(X_train, Y_train, 1000, 0.0001)
```

```
0.93359375
```

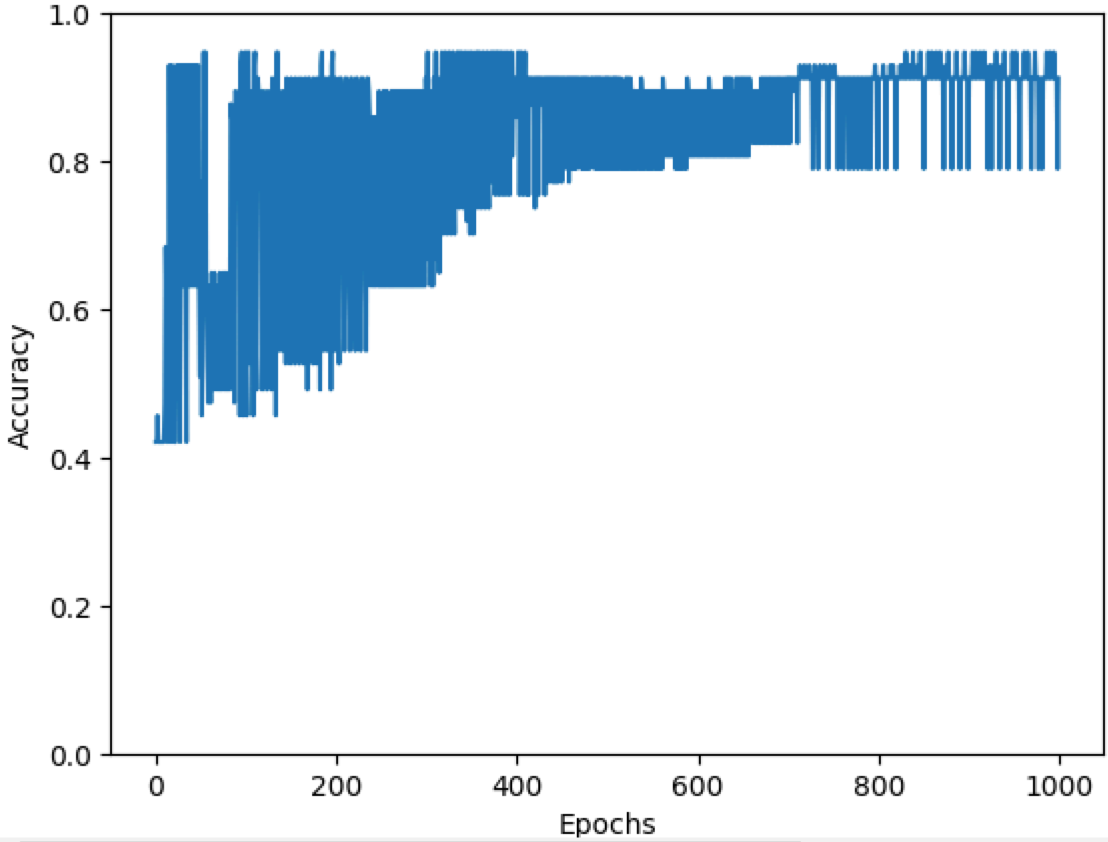



```
plt.plot(perceptron.w)
plt.show()
```




```
perceptron.fit(X_test, Y_test, 1000, 0.0001)
```

```
0.9473684210526315
```



Predicts the test set using the trained model and calculates the accuracy of these predictions compared to the true labels, providing a quantitative measure of model performance.

```
Y_pred_test = perceptron.predict(X_test)
print(accuracy_score(Y_pred_test, Y_test))
```

```
0.9473684210526315
```

Exercise 4: Implement Perceptron algorithm on the binary Iris dataset and explore its performance by adjusting learning rates and analyzing the weight changes during training

a. Understand the Iris Dataset and write a summary of features

b. Train/ Test Split

c. Implement the Perceptron Algorithm

d. Plot Train/Test Accuracy: Once the model is trained, evaluate the accuracy on both the training and testing datasets. Plot the accuracy for the training and testing data to visualize the model's performance over multiple epochs.

e. Experimenting with Learning Rates

f. Run the Perceptron algorithm with different learning rates.

▪ Observe how changing the learning rate impacts the model's ability to converge and its overall accuracy.

▪ Interpret the results: Does a higher learning rate lead to faster convergence or instability? Does a lower learning rate affect the speed or quality of the model's learning?

g. Visualizing the Weight Changes

▪ During training, the Perceptron's weights are updated in each epoch. To understand how the weights evolve:

o Create a weight matrix that stores the weight values for each epoch.

o After each epoch, append the current weights to the matrix.

o Plot the weights as they change across epochs. This will help visualize how the model adjusts its weights based on the data.

o Write the Interpretation: After plotting the weight changes, explain how the model's weights stabilize as it learns from the data. Do weights converge?

```
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load Iris dataset
iris = load_iris()
X = iris.data
Y = iris.target

# Select only the first two classes for binary classification (Setosa and Versicolor)
binary_indices = np.where(Y < 2)
X = X[binary_indices]
Y = Y[binary_indices]

# Convert labels to 0 and 1
Y = np.where(Y == 0, 0, 1)

print("Shape of X:", X.shape)
print("Shape of Y:", Y.shape)
```

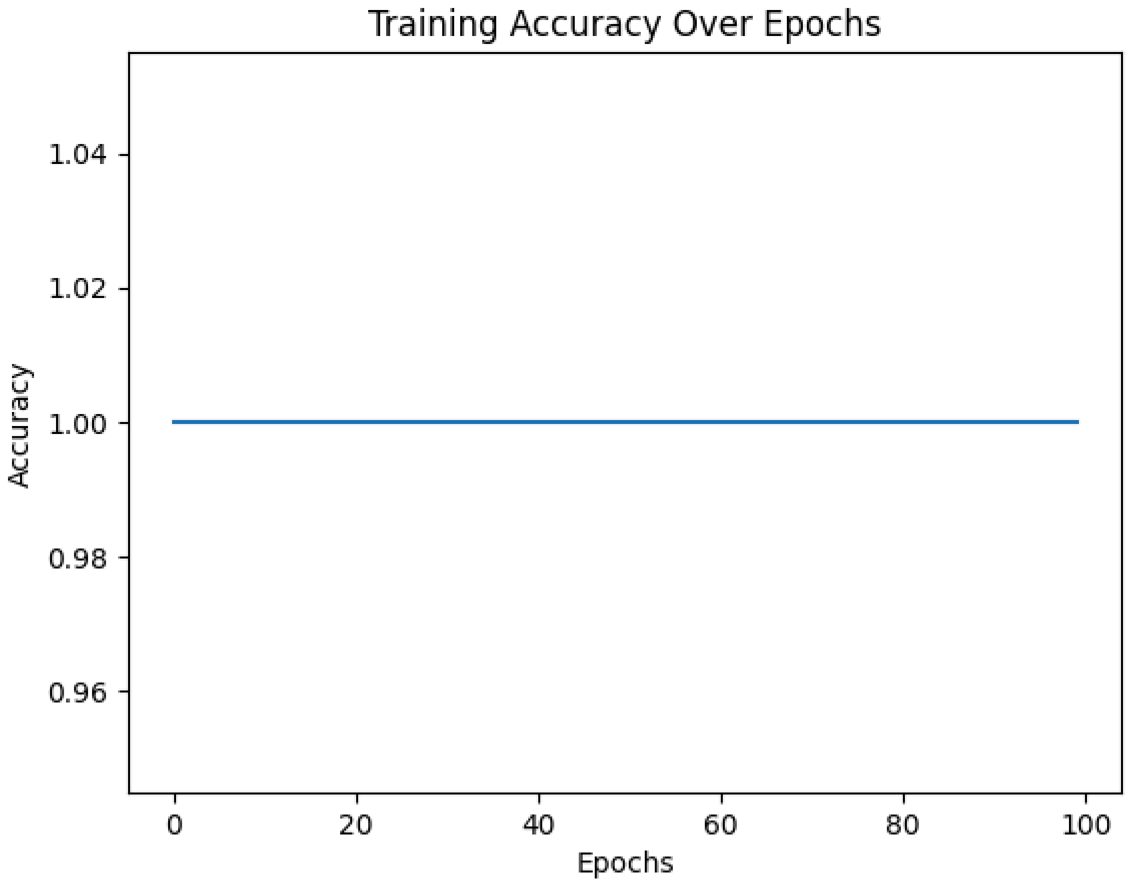
```
Shape of X: (100, 4)
Shape of Y: (100,)
```

```
# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

```
class Perceptron:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs
```

```
        self.w = None
        self.b = None
        self.accuracies = []

    def model(self, x):
        return 1 if (np.dot(self.w, x) >= self.b) else 0

    def fit(self, X, Y):
        self.w = np.zeros(X.shape[1])
        self.b = 0
        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                y_pred = self.model(X[i])
                if Y[i] == 1 and y_pred == 0:
                    self.w += self.learning_rate * X[i]
                    self.b -= self.learning_rate
                elif Y[i] == 0 and y_pred == 1:
                    self.w -= self.learning_rate * X[i]
                    self.b += self.learning_rate
            # Calculate accuracy for visualization
            accuracy = self.accuracy(X, Y)
            self.accuracies.append(accuracy)

    def predict(self, X):
        return [self.model(x) for x in X]

    def accuracy(self, X, Y):
        Y_pred = self.predict(X)
        return np.mean(Y_pred == Y)
```

```
# Train the Perceptron
perceptron = Perceptron(learning_rate=0.01, epochs=100)
perceptron.fit(X_train, Y_train)

# Plot accuracy over epochs
plt.plot(perceptron.accuracies)
plt.title("Training Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.show()

# Test set accuracy
test_accuracy = perceptron.accuracy(X_test, Y_test)
print("Test Set Accuracy:", test_accuracy)
```



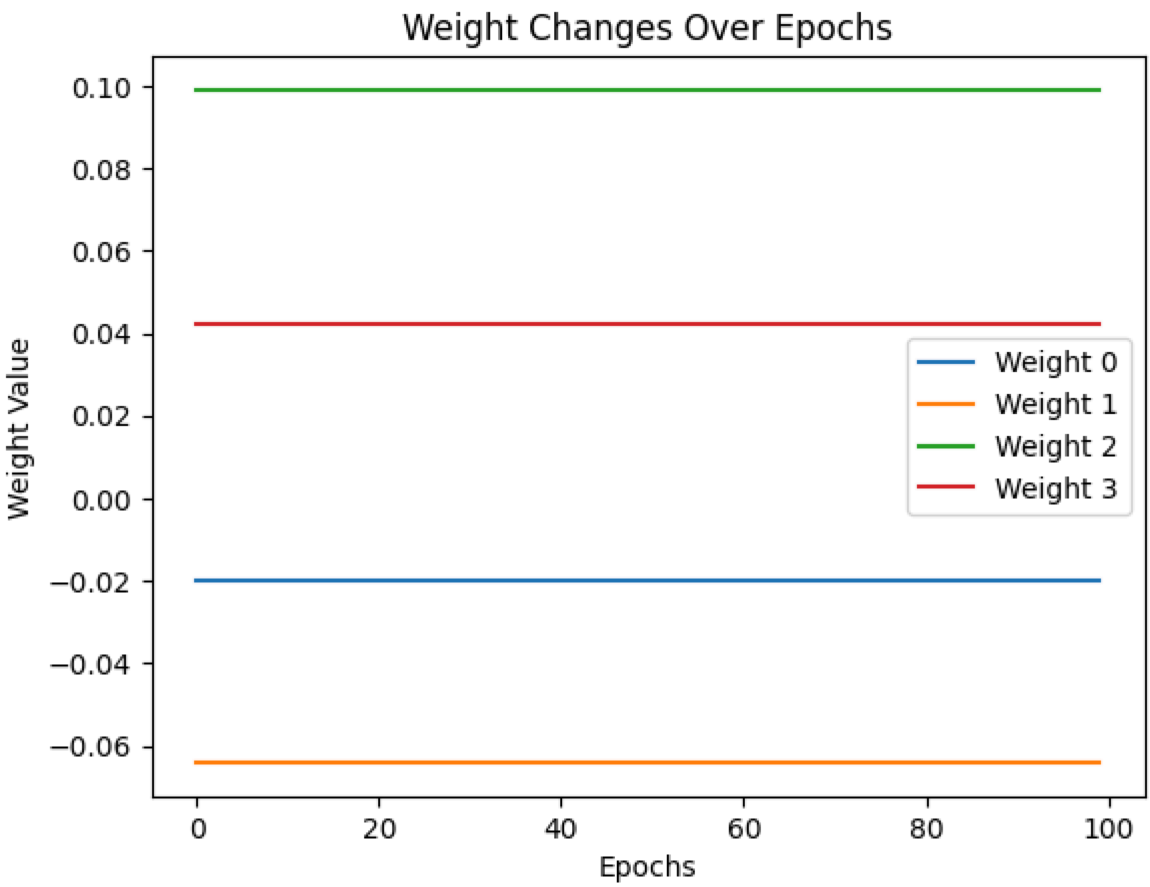

```
Test Set Accuracy: 1.0
```

```
learning_rates = [0.001, 0.01, 0.1]
for lr in learning_rates:
    perceptron = Perceptron(learning_rate=lr, epochs=100)
    perceptron.fit(X_train, Y_train)
    print(f"Learning Rate: {lr}, Final Training Accuracy: {perceptron.accuracies[-1]}")
```

```
Learning Rate: 0.001, Final Training Accuracy: 1.0
Learning Rate: 0.01, Final Training Accuracy: 1.0
Learning Rate: 0.1, Final Training Accuracy: 1.0
```

```
class PerceptronWithWeightTracking(Perceptron):
    def __init__(self, learning_rate=0.01, epochs=100):
        super().__init__(learning_rate, epochs)
        self.weight_history = []

    def fit(self, X, Y):
        self.w = np.zeros(X.shape[1])
        self.b = 0
        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                y_pred = self.model(X[i])
                if Y[i] == 1 and y_pred == 0:
                    self.w += self.learning_rate * X[i]
                    self.b -= self.learning_rate
                elif Y[i] == 0 and y_pred == 1:
                    self.w -= self.learning_rate * X[i]
```

```
                    self.b += self.learning_rate
            # Store weights after each epoch
            self.weight_history.append(self.w.copy())

# Track and plot weight changes
perceptron_wt = PerceptronWithWeightTracking(learning_rate=0.01, epochs=100)
perceptron_wt.fit(X_train, Y_train)

# Plot weight changes
weights = np.array(perceptron_wt.weight_history)
for i in range(weights.shape[1]):
    plt.plot(weights[:, i], label=f"Weight {i}")
plt.title("Weight Changes Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Weight Value")
plt.legend()
plt.show()
```




```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```