Activity 14
Machine Learning : Logistic Regression

Marc Jerrone Castro

Binary Classification

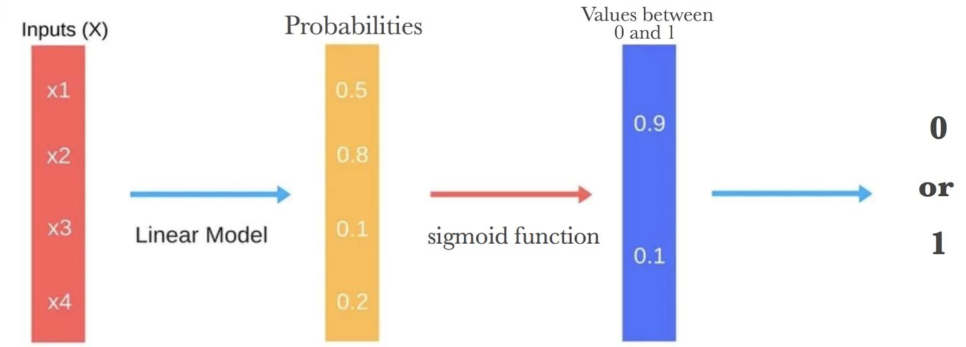


Figure 1. Simplified diagram on how a binary classifier using a sigmoid function works.

There has been an increasing amount of interest in the field of computer vision which deals with simple classification techniques using Machine Learning. In this study, we utilize a simple but a bit more complex activation function to classify if a mango is ripe(1) or not(0).

IMAGE SOURCE: https://towardsdatascience.com/real-world-implementation-of-logistic-regression-5136cefb8125

Sigmoid Function

$$f(x) = \frac{1}{1+e^{-x}}$$

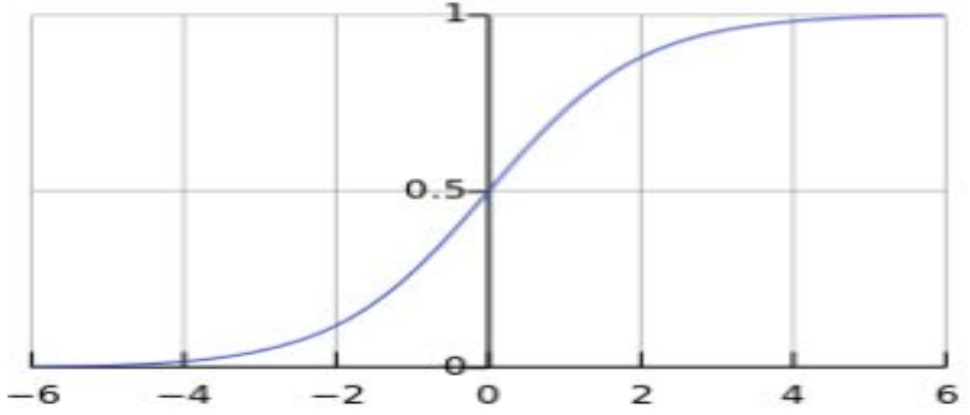
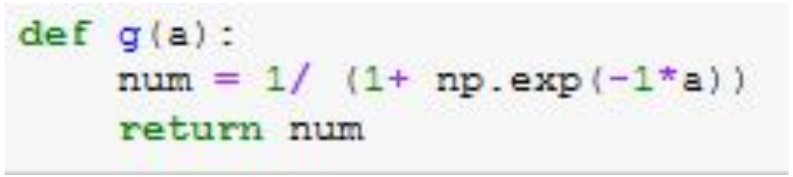


Figure 2. Logistic or Sigmoid Function.



```
def g(a):
    num = 1/ (1+ np.exp(-1*a))
    return num
```

The sigmoid function highly resembles a step function whereas it differs in the transition between low and high states such that a sigmoid function resembles a smooth and continuous slope. The resulting values for sigmoid functions in binary classifier via machine learning applications generally describes the features of an input using a scoring system ranging from 0 to 1. In this experiment we aim to train an artificial neuron to score the ripeness of a mango on the basis of its color using the sigmoid function as its activation function.

IMAGE SOURCE: https://en.wikipedia.org/wiki/Sigmoid_function

So how does it differ from simple perceptrons?

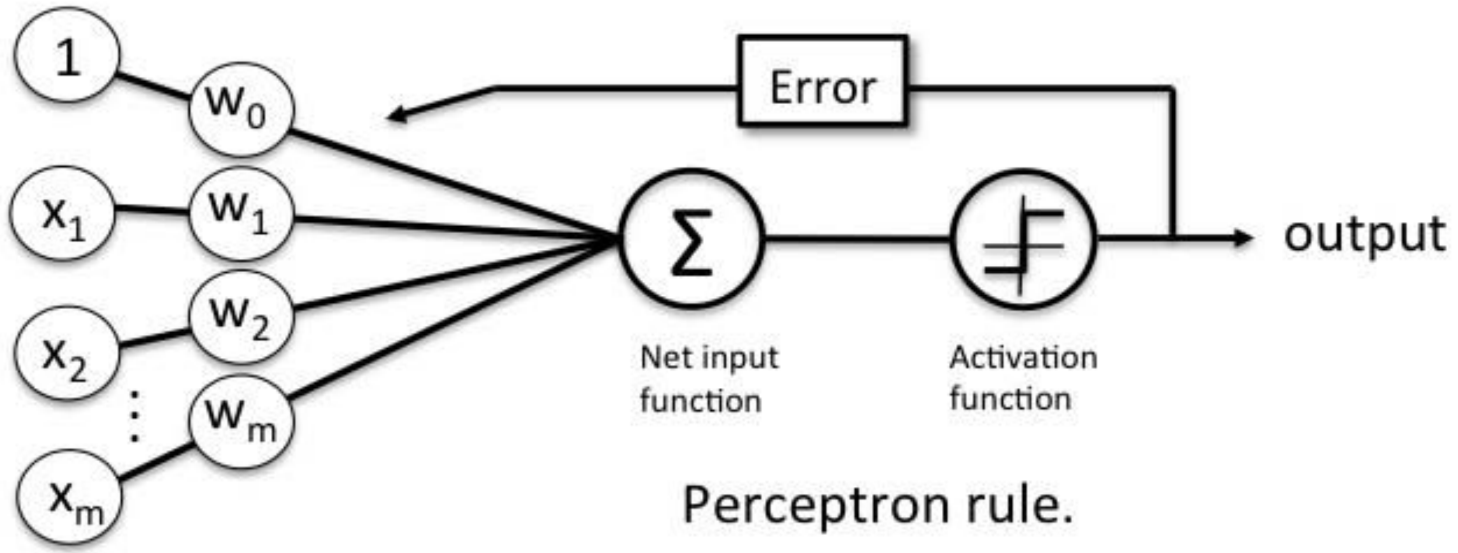


Figure 3a. Diagram explaining how a single-layer perceptron works.

Logistic Regression

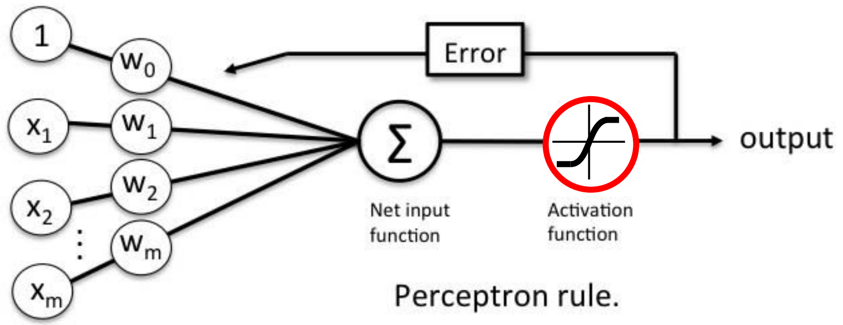
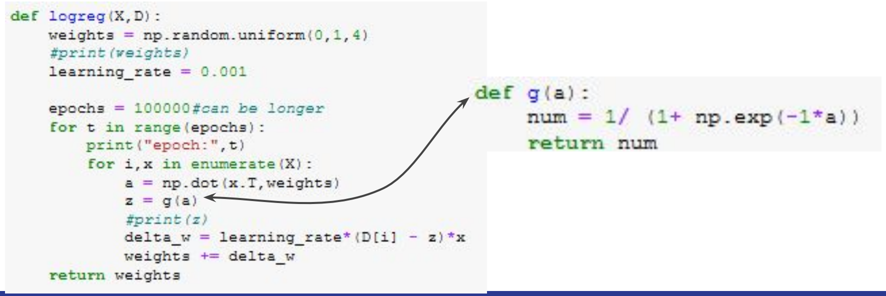


Figure 3b. Diagram explaining how a single-layer perceptron works using a sigmoid activation function.



```
def logreg(X,D):
    weights = np.random.uniform(0,1,4)
    #print(weights)
    learning_rate = 0.001

    epochs = 100000#can be longer
    for t in range(epochs):
        print("epoch:",t)
        for i,x in enumerate(X):
            a = np.dot(x.T,weights)
            z = g(a)
            #print(z)
            delta_w = learning_rate*(D[i] - z)*x
            weights += delta_w
    return weights
```

```
def g(a):
    num = 1/ (1+ np.exp(-1*a))
    return num
```

The process is **quite similar to our previous activity** on single-layer perceptrons whereas it takes in a set of features *X* and a predetermined bias of 1 and initially assign them random weights. These features are then multiplied to their corresponding weights and then summed. The sum is then fed into an activation function - the activation function used for this experiment is the **sigmoid function which determines the probability** of an input belonging to a certain class. This probability is then used to adjust the weights over a number of iterations as a means to develop a linear model of our binary classifier.

Objectives of Study

This study aims to train an artificial neuron to learn a linear model for classifying ripe and unripe oranges using sigmoid function as its activation function. We aim to train the neuron to adjust initially-random weights using our basic knowledge of a single-layer perceptron. Afterwards, we aim to validate the learned model using a test dataset which are images that were not used for training the neuron.

Data Used

The data used for this study is less than ideal, as the general image dimensions are not uniform - however pre-processing steps such as cropping via contour detection limit these factors in our results. The images used were collected from various sources listed in Google Images. A particular characteristic of each image that I have pre-selected is that it should be an image imposed on a white background as it is generally easier to parse out the white background in contrast to noisy backgrounds - although I have failed to remove watermarks superimposed on the image. As I have no control over the general resolution of the images, I had opted to assume this as a plausible source of error.

Pre-processing through cropping

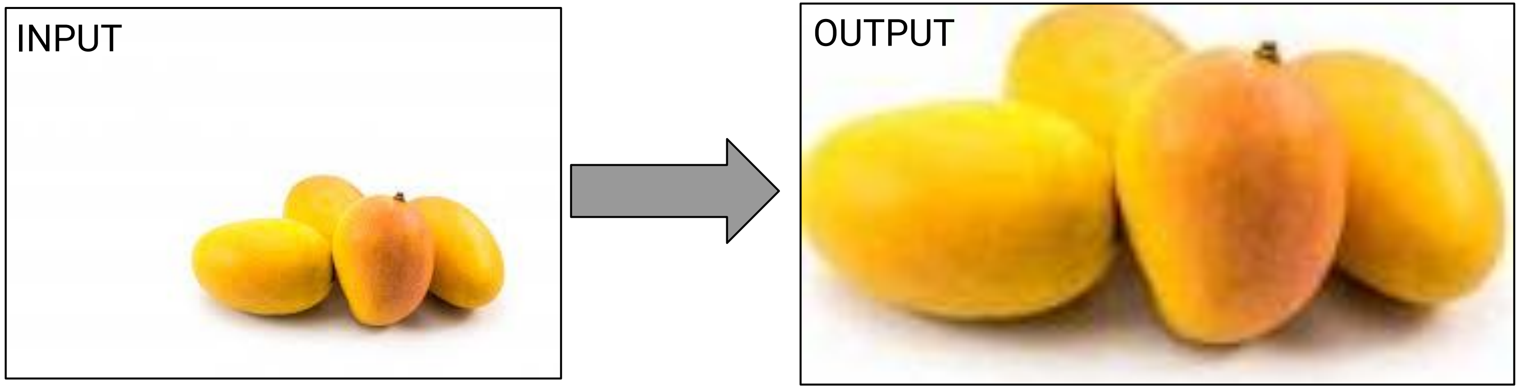
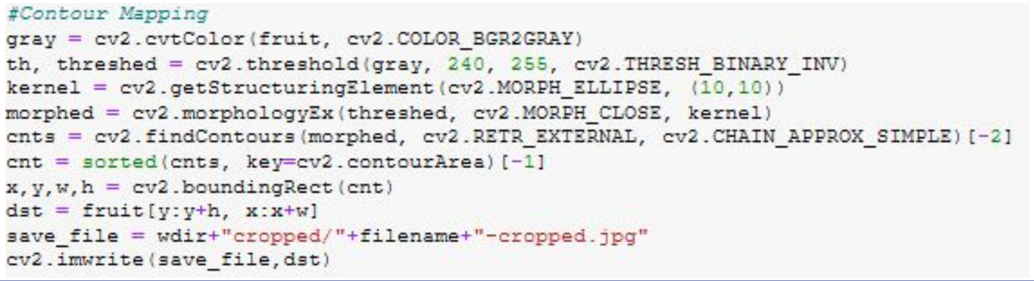


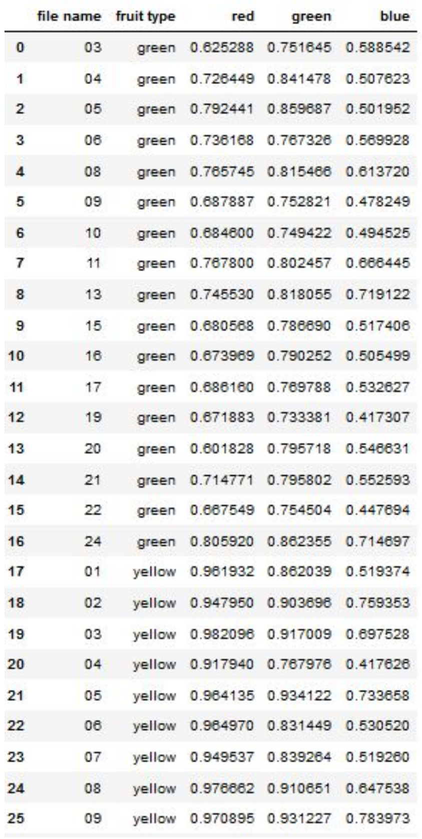
Figure 4.0 Comparison of input images after cropping via contour detection.

As for my pre-processing step, I had performed cropping via contour detection before any analysis as to limit the amount of white spaces found within the background. This would allow for the mean R,G, and B values of the image to have minimal dependency on the white pixels found within the image.



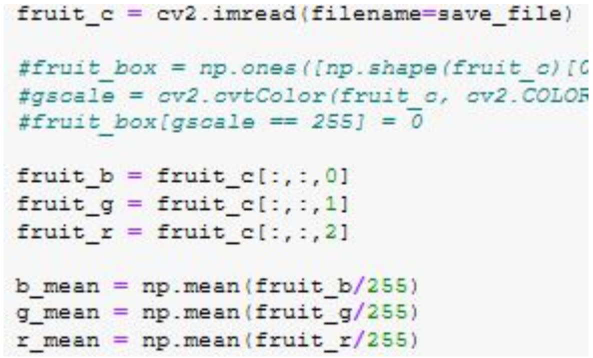
```
#Contour Mapping
gray = cv2.cvtColor(fruit, cv2.COLOR_BGR2GRAY)
th, threshed = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY_INV)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10,10))
morphed = cv2.morphologyEx(threshed, cv2.MORPH_CLOSE, kernel)
cnts = cv2.findContours(morphed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
cnt = sorted(cnts, key=cv2.contourArea)[-1]
x,y,w,h = cv2.boundingRect(cnt)
dst = fruit[y:y+h, x:x+w]
save_file = wdir+"cropped/"+filename+"-cropped.jpg"
cv2.imwrite(save_file,dst)
```

Features Extracted



	file name	fruit type	red	green	blue
0	03	green	0.625288	0.751645	0.588542
1	04	green	0.726449	0.841478	0.507623
2	05	green	0.792441	0.859687	0.501952
3	06	green	0.736168	0.767326	0.569928
4	08	green	0.765745	0.815466	0.613720
5	09	green	0.687887	0.752821	0.478249
6	10	green	0.684600	0.749422	0.494525
7	11	green	0.767800	0.802457	0.666445
8	13	green	0.745530	0.818055	0.719122
9	15	green	0.680568	0.786690	0.517406
10	16	green	0.673969	0.790252	0.505499
11	17	green	0.686160	0.769788	0.532627
12	19	green	0.671883	0.733381	0.417307
13	20	green	0.601828	0.795718	0.546631
14	21	green	0.714771	0.795802	0.552593
15	22	green	0.667549	0.754504	0.447694
16	24	green	0.805920	0.862355	0.714697
17	01	yellow	0.961932	0.862039	0.519374
18	02	yellow	0.947950	0.903696	0.759353
19	03	yellow	0.982096	0.917009	0.697528
20	04	yellow	0.917940	0.767976	0.417626
21	05	yellow	0.964135	0.934122	0.733658
22	06	yellow	0.964970	0.831449	0.530520
23	07	yellow	0.949537	0.839264	0.519260
24	08	yellow	0.976662	0.910651	0.647538
25	09	yellow	0.970895	0.931227	0.783973

Using the code below, we extracted the mean value for each of the RGB channels of the image. A reasonable difference was observed for green and red channels, however no distinct clustering was observed for the blue channels. We then import these features into a pandas dataframe. The image on the right shows the first 25 entries of our dataset along with its corresponding features. To visualize distinct differences between ripe and unripe mangoes we plot each relevant data pair of the dataframe using a scatter plot.



```
fruit_c = cv2.imread(filename=save_file)

#fruit_box = np.ones([np.shape(fruit_c)[C
#gscale = cv2.cvtColor(fruit_c, cv2.COLOR
#fruit_box[gscale == 255] = 0

fruit_b = fruit_c[:,:,0]
fruit_g = fruit_c[:,:,1]
fruit_r = fruit_c[:,:,2]

b_mean = np.mean(fruit_b/255)
g_mean = np.mean(fruit_g/255)
r_mean = np.mean(fruit_r/255)
```

Features Extracted

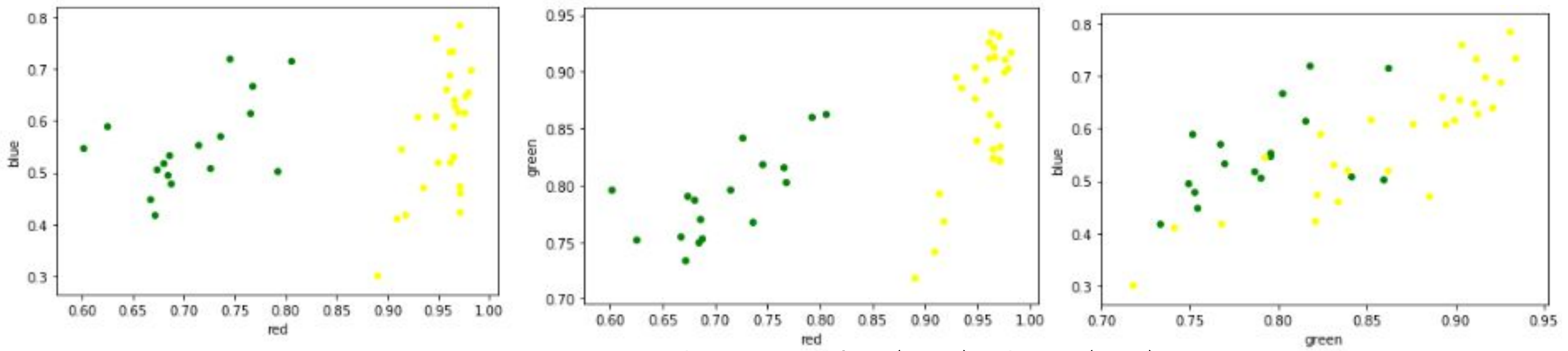


Figure 5. Scatter plots average color channel values of ripe (yellow) and unripe (green) mangoes.

Figure 5 visualizes a distinct clustering with respect to the ripeness of the fruit. Since we have verified that is possible to separate these two classes via clustering, we can now use a single-layer perceptron to train a neuron to learn a model which would distinguish between the ripeness of a fruit.

Learned Model

Wo :
```
[0.38010204 0.56654373 0.10510913 0.83643244]
```

Xo **R _mean** **G _mean** **B _mean**

Wn :
```
[-19.00492967,  28.90528774,  -4.1066128 ,  -3.39429941]
```

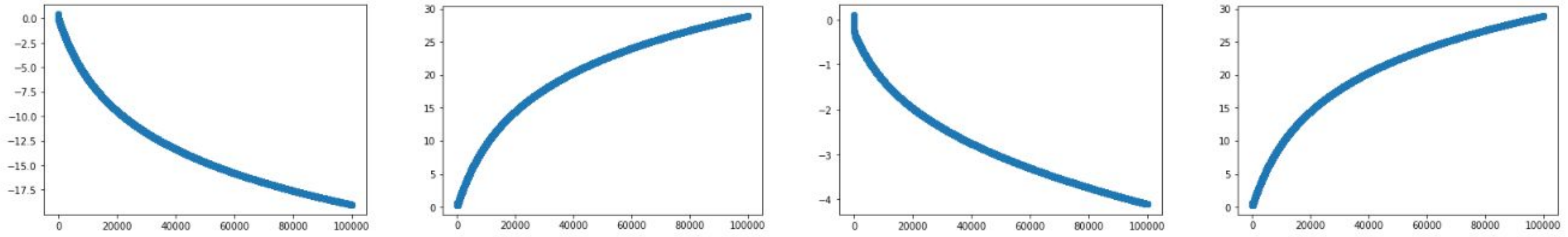


Figure 6. Gradual weight adjustment over 100000 iterations.

Our model gradually adjusted the initial random weights ***Wo*** into ***Wn*** after 100,000 epochs. By using weight adjustment used in Activity 13, the neuron determines the ideal weights for our binary classifier and steadily adjusts the model. Although there was a significant change in weights, this however, has yet to reach the maximum change it can go as the plots above have yet to plateau even after 100,000 epochs. Further increasing the number of epochs to reach this plateau would significantly benefit our model,however due to hardware limitations, we are unable to increase the number of epochs.

Learned Model

Wn :

```
[-19.00492967,  28.90528774,  -4.1066128 ,  -3.39429941]
```

Thus our model would have the following approximate linear equation:

a = (-19.005) + (28.905 * **x1**) + (-4.107 * **x2**) + (-3.394 * **x3**)

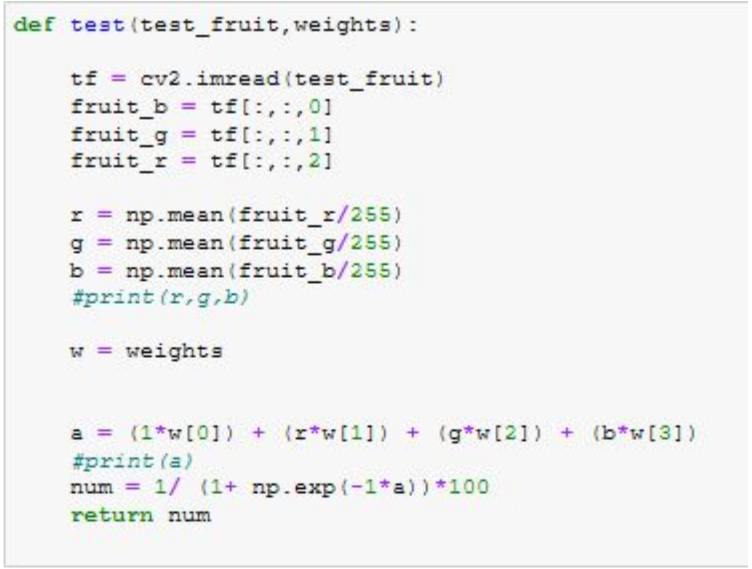
Where x1 is the average mean of the red channel, x2 is the average mean of the green channel, and x3 is the average mean of the blue channel. And the probability that the mango is ripe is given by the sigmoid function,

$$f(x) = \frac{1}{1+e^{-x}}$$

Where we treat ***a*** as the input of the sigmoid function. The resulting value would dictate the probability that the mango is ripe. To validate our implementation, we test the model on unlearned images of mangoes and check the accuracy of each prediction.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Learned Model



```
def test(test_fruit,weights):

    tf = cv2.imread(test_fruit)
    fruit_b = tf[:,:,0]
    fruit_g = tf[:,:,1]
    fruit_r = tf[:,:,2]

    r = np.mean(fruit_r/255)
    g = np.mean(fruit_g/255)
    b = np.mean(fruit_b/255)
    #print(r,g,b)

    w = weights


    a = (1*w[0]) + (r*w[1]) + (g*w[2]) + (b*w[3])
    #print(a)
    num = 1/ (1+ np.exp(-1*a))*100
    return num
```

To test the accuracy of our model, we utilize the following snippet of code to determine the percentage of ripeness an image has. The method extracts the mean pixel values of each of the RGB channels of an input image and then determine the sum of the synaptic learned-weights. Afterwards, we input the sum into the sigmoid function to determine the ripeness score of an image.

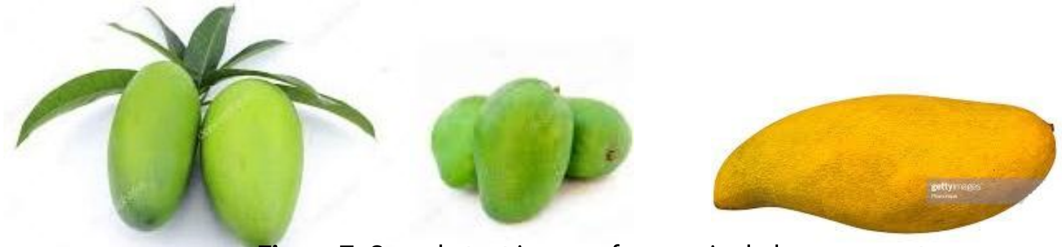
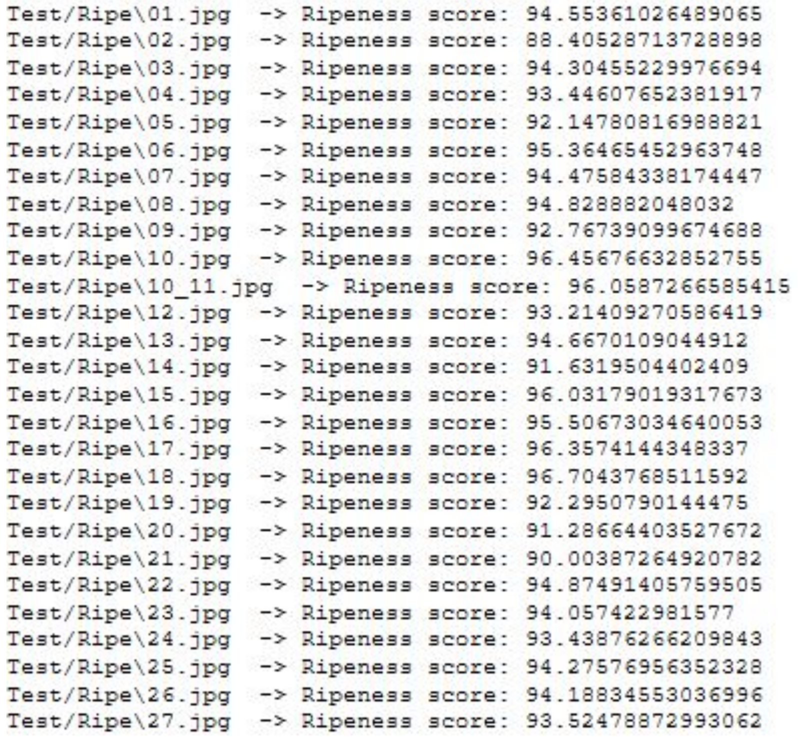
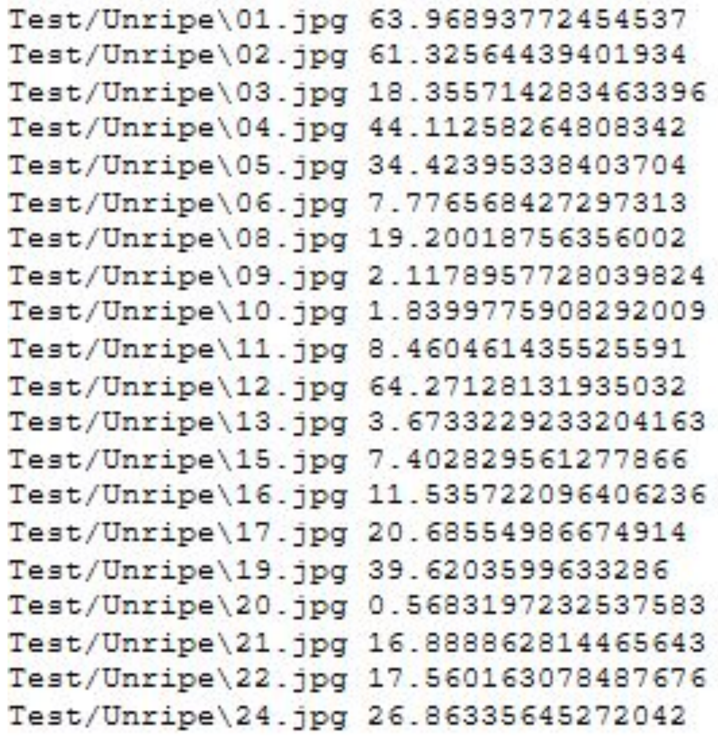


Figure 7. Sample test images for our single-layer perceptron.

Results



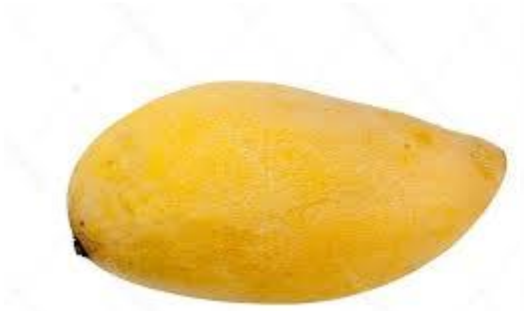
```
Test/Ripe\01.jpg  -> Ripeness score: 94.55361026489065
Test/Ripe\02.jpg  -> Ripeness score: 88.40528713728898
Test/Ripe\03.jpg  -> Ripeness score: 94.30455229976694
Test/Ripe\04.jpg  -> Ripeness score: 93.44607652381917
Test/Ripe\05.jpg  -> Ripeness score: 92.14780816988821
Test/Ripe\06.jpg  -> Ripeness score: 95.36465452963748
Test/Ripe\07.jpg  -> Ripeness score: 94.47584338174447
Test/Ripe\08.jpg  -> Ripeness score: 94.828882048032
Test/Ripe\09.jpg  -> Ripeness score: 92.76739099674688
Test/Ripe\10.jpg  -> Ripeness score: 96.45676632852755
Test/Ripe\10_11.jpg  -> Ripeness score: 96.0587266585415
Test/Ripe\12.jpg  -> Ripeness score: 93.21409270586419
Test/Ripe\13.jpg  -> Ripeness score: 94.6670109044912
Test/Ripe\14.jpg  -> Ripeness score: 91.6319504402409
Test/Ripe\15.jpg  -> Ripeness score: 96.03179019317673
Test/Ripe\16.jpg  -> Ripeness score: 95.50673034640053
Test/Ripe\17.jpg  -> Ripeness score: 96.3574144348337
Test/Ripe\18.jpg  -> Ripeness score: 96.7043768511592
Test/Ripe\19.jpg  -> Ripeness score: 92.2950790144475
Test/Ripe\20.jpg  -> Ripeness score: 91.28664403527672
Test/Ripe\21.jpg  -> Ripeness score: 90.00387264920782
Test/Ripe\22.jpg  -> Ripeness score: 94.87491405759505
Test/Ripe\23.jpg  -> Ripeness score: 94.057422981577
Test/Ripe\24.jpg  -> Ripeness score: 93.43876266209843
Test/Ripe\25.jpg  -> Ripeness score: 94.27576956352328
Test/Ripe\26.jpg  -> Ripeness score: 94.18834553036996
Test/Ripe\27.jpg  -> Ripeness score: 93.52478872993062
```



```
Test/Unripe\01.jpg 63.96893772454537
Test/Unripe\02.jpg 61.32564439401934
Test/Unripe\03.jpg 18.355714283463396
Test/Unripe\04.jpg 44.11258264808342
Test/Unripe\05.jpg 34.42395338403704
Test/Unripe\06.jpg 7.776568427297313
Test/Unripe\08.jpg 19.20018756356002
Test/Unripe\09.jpg 2.1178957728039824
Test/Unripe\10.jpg 1.8399775908292009
Test/Unripe\11.jpg 8.460461435525591
Test/Unripe\12.jpg 64.27128131935032
Test/Unripe\13.jpg 3.6733229233204163
Test/Unripe\15.jpg 7.402829561277866
Test/Unripe\16.jpg 11.535722096406236
Test/Unripe\17.jpg 20.68554986674914
Test/Unripe\19.jpg 39.6203599633286
Test/Unripe\20.jpg 0.5683197232537583
Test/Unripe\21.jpg 16.888862814465643
Test/Unripe\22.jpg 17.560163078487676
Test/Unripe\24.jpg 26.86335645272042
```

Results

Ripeness: 93.21%


Ripeness: 93.45%

Ripeness: 97.70%

Ripeness: 93.52%

Our model was able to accurately determine the ripeness of a mango as observed from the ripeness score observed above. As one may observe, the model determines the following images as above 90% probability of being ripe which is fairly accurate.

Results

Ripeness: 11.54%

Ripeness: 26.86%

Ripeness: 18.36%

Ripeness: 11.54%

Similarly, low probability values were determined for unripe mangoes which suggests that the model was able to distinguish between the two classes.

SCORE

QoP : 5/5

TC : 5/5

Initiative: 1