Activity 13
Machine Learning : Perceptrons

Marc Jerrone Castro

Perceptrons

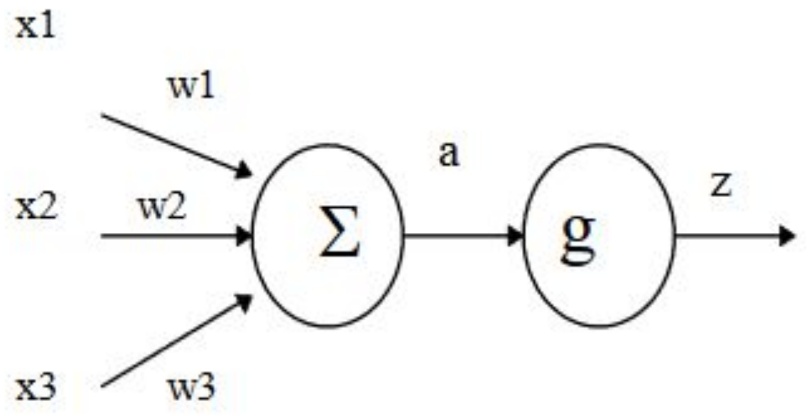


Figure 1. Artificial neurons that aim to simulate a human neuron through mathematical methods.

Perceptrons are derived from an earlier version of mathematical model for mapping the synapses between human neurons. The McCulloch-Pitts Neuron (MCP) aims to simulate these neuron-neuron interactions in the form shown in Figure 1. It derives a synaptic strength or weight w_i. The neuron sums these weighted inputs ($x_i w_i$) and lets the sum, a , act on an activation function, g. The result, z, is then fed into a neighboring neuron as its new input.

How does a perceptron work?

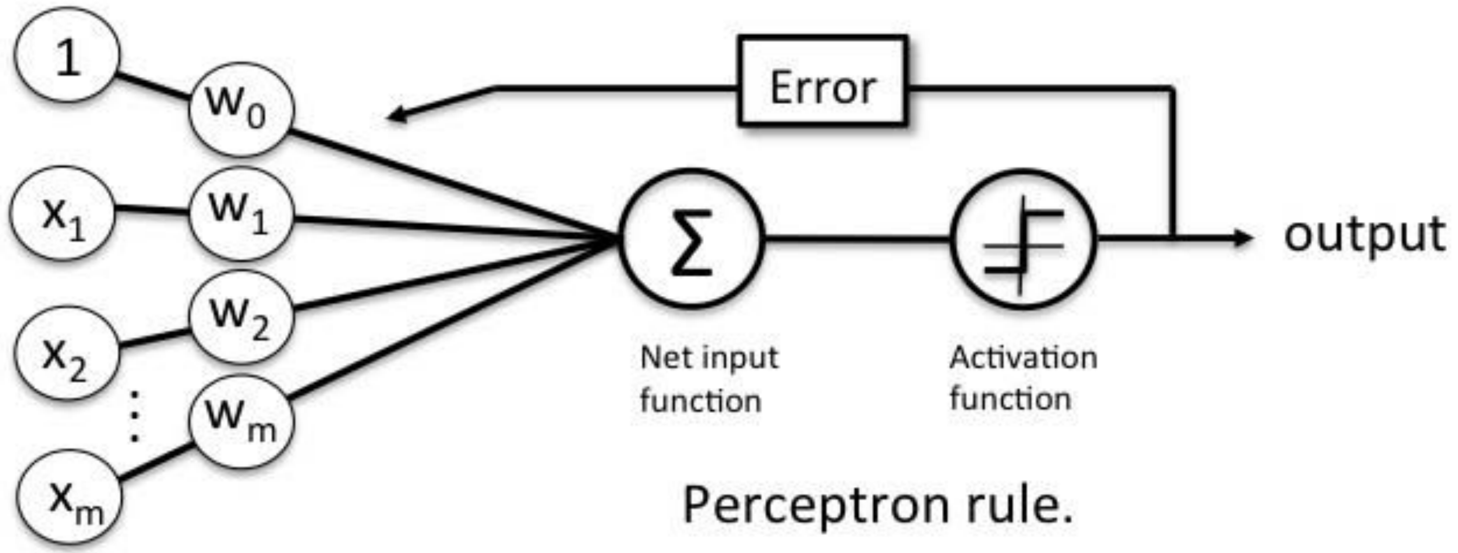


Figure 2. Diagram explaining how a single-layer perceptron works.

How does a perceptron work?

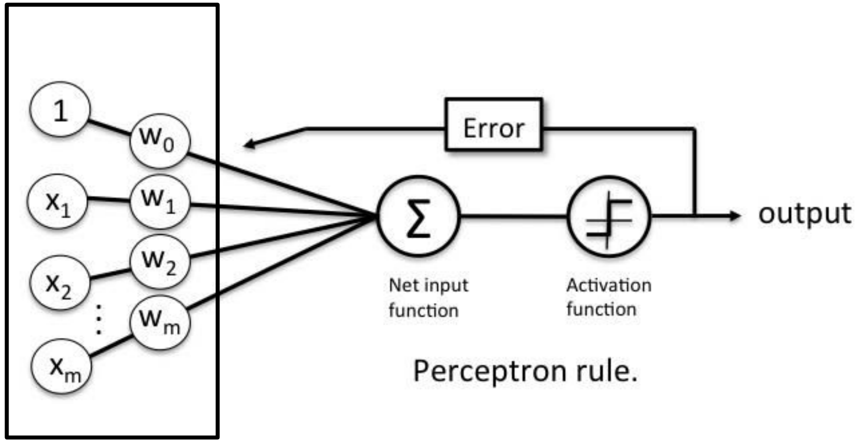
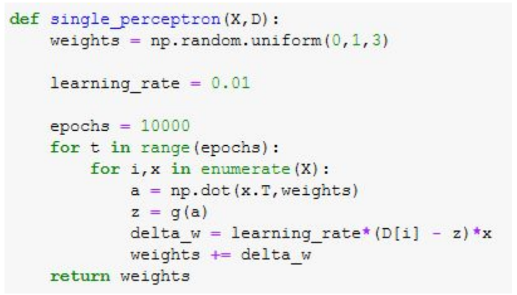


Figure 2. Diagram explaining how a single-layer perceptron works.



```python
def single_perceptron(X,D):
    weights = np.random.uniform(0,1,3)

    learning_rate = 0.01

    epochs = 10000
    for t in range(epochs):
        for i,x in enumerate(X):
            a = np.dot(x.T,weights)
            z = g(a)
            delta_w = learning_rate*(D[i] - z)*x
            weights += delta_w
    return weights
```

A perceptron works by taking 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 step function which returns a value of 1 if the function returns a number greater than zero, and outputs -1 ,otherwise

How does a perceptron work?

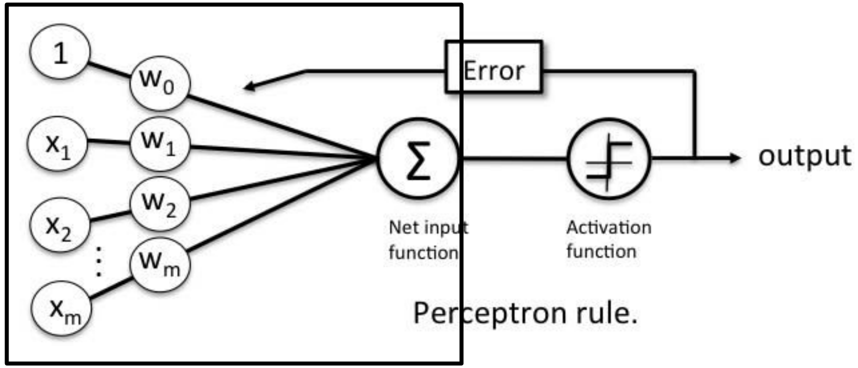
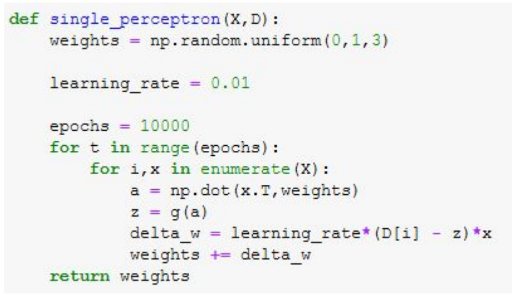


Figure 2. Diagram explaining how a single-layer perceptron works.



```
def single_perceptron(X,D):
    weights = np.random.uniform(0,1,3)

    learning_rate = 0.01

    epochs = 10000
    for t in range(epochs):
        for i,x in enumerate(X):
            a = np.dot(x.T,weights)
            z = g(a)
            delta_w = learning_rate*(D[i] - z)*x
            weights += delta_w
    return weights
```

A perceptron works by taking 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 step function which returns a value of 1 if the function returns a number greater than zero, and outputs -1 ,otherwise

How does a perceptron work?

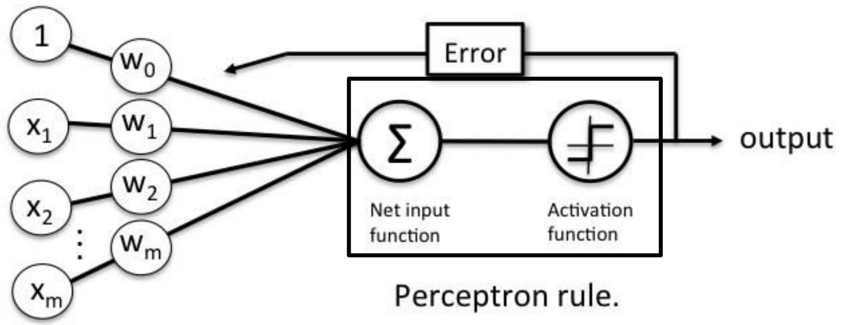
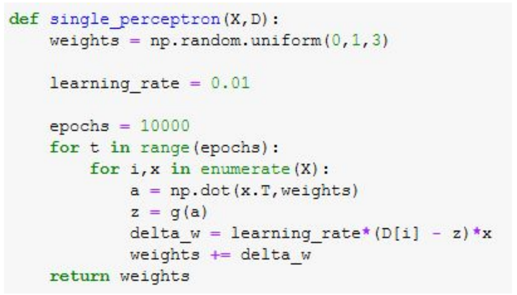


Figure 2. Diagram explaining how a single-layer perceptron works.



```python
def single_perceptron(X,D):
    weights = np.random.uniform(0,1,3)

    learning_rate = 0.01

    epochs = 10000
    for t in range(epochs):
        for i,x in enumerate(X):
            a = np.dot(x.T,weights)
            z = g(a)
            delta_w = learning_rate*(D[i] - z)*x
            weights += delta_w
    return weights
```

A perceptron works by taking 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 step function which returns a value of 1 if the function returns a number greater than zero, and outputs -1 ,otherwise.**

How does a perceptron work?

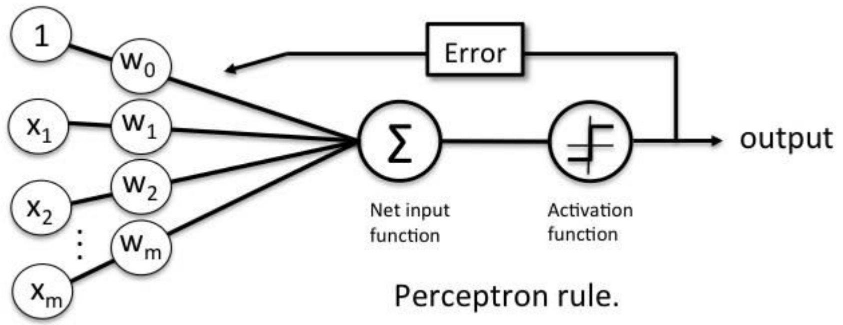
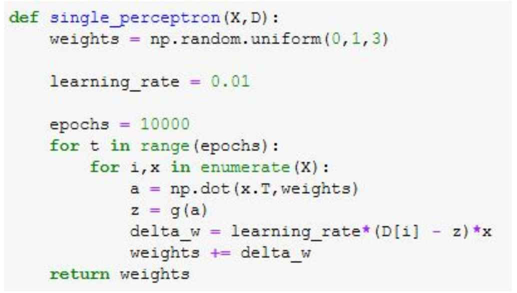


Figure 2. Diagram explaining how a single-layer perceptron works.



```python
def single_perceptron(X,D):
    weights = np.random.uniform(0,1,3)

    learning_rate = 0.01

    epochs = 10000
    for t in range(epochs):
        for i,x in enumerate(X):
            a = np.dot(x.T,weights)
            z = g(a)
            delta_w = learning_rate*(D[i] - z)*x
            weights += delta_w
    return weights
```

An error or difference of weight is then determined and then added into the initial randomized weight.This performed for a number of iterations commonly referred to as epochs. This weight adjustment is also dependent on a constant value called the learning rate of the machine - or the rate at which the machine adjusts the weight of the machine. Tendencies of the learning rate are that the smaller it is, the slower the weight adjustment will be and the more accurate the adjustment is. This is contrast to larger values of the learning rate, where robust changes occur in the weight.

Objectives of Study

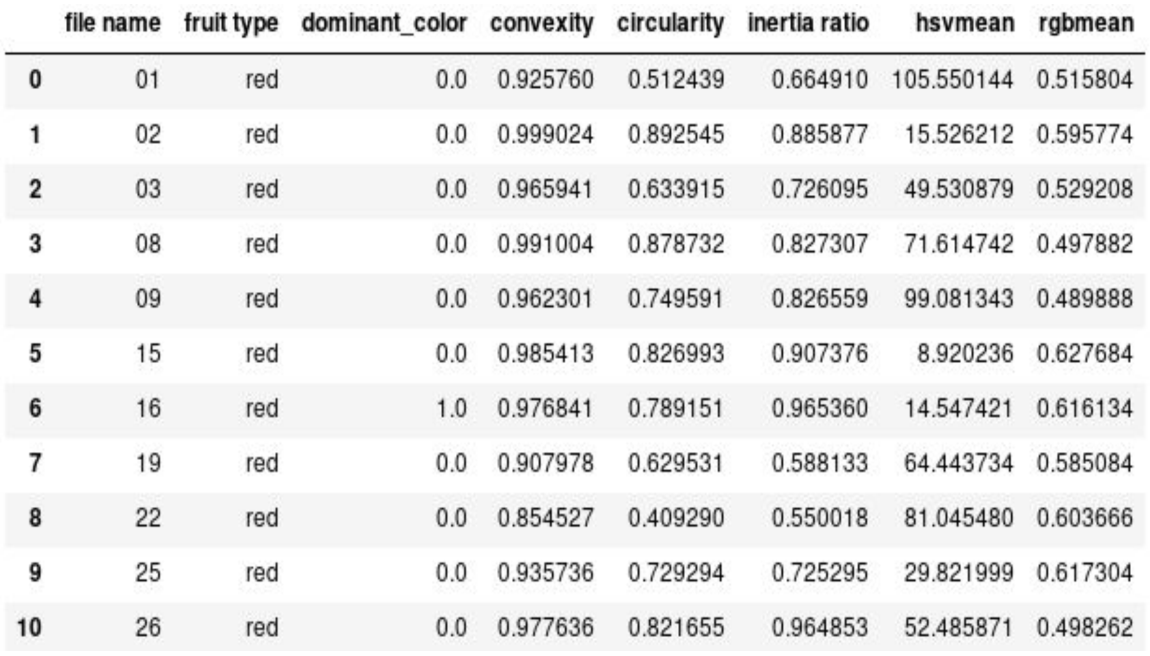
In this particular study, we aim to utilize the features we extracted from Activity 12 and use a single perceptron layer to create a decision line which would separate one class from another.

The equation of the line is described by the following equation,

$$y=\frac{C}{B}-\frac{A}{B}x$$

Where C/B is the y-intercept and A/B is the slope of the line. C is equal to the negative weight of our initial bias of 1 while A and B are respective weights derived from two predetermined features.

Features Extracted



	file name	fruit type	dominant_color	convexity	circularity	inertia ratio	hsvmean	rgbmean
0	01	red	0.0	0.925760	0.512439	0.664910	105.550144	0.515804
1	02	red	0.0	0.999024	0.892545	0.885877	15.526212	0.595774
2	03	red	0.0	0.965941	0.633915	0.726095	49.530879	0.529208
3	08	red	0.0	0.991004	0.878732	0.827307	71.614742	0.497882
4	09	red	0.0	0.962301	0.749591	0.826559	99.081343	0.489888
5	15	red	0.0	0.985413	0.826993	0.907376	8.920236	0.627684
6	16	red	1.0	0.976841	0.789151	0.965360	14.547421	0.616134
7	19	red	0.0	0.907978	0.629531	0.588133	64.443734	0.585084
8	22	red	0.0	0.854527	0.409290	0.550018	81.045480	0.603666
9	25	red	0.0	0.935736	0.729294	0.725295	29.821999	0.617304
10	26	red	0.0	0.977636	0.821655	0.964853	52.485871	0.498262

From Activity 12, we extracted the following features. However, we will only be focusing three primary features - convexity, circularity, and the mean RGB value as they have resulted to the best possible combinations for differentiating apples, oranges, and bananas.

Features Extracted

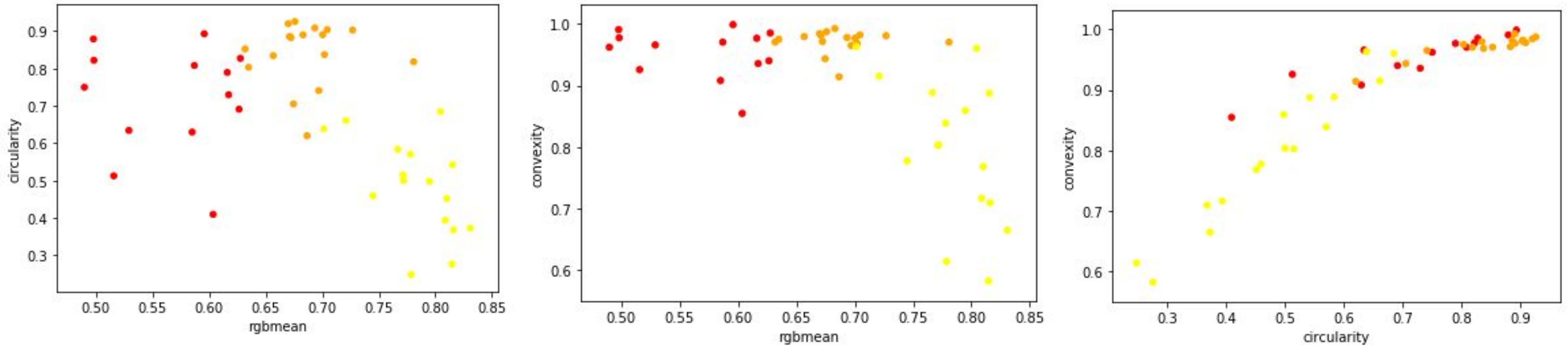


Figure 3. Scatter plots of average RGB value, convexity, and circularity of apples (red), oranges (orange), and bananas (yellow).

As mentioned earlier, the objective of this study is to determine a decision line which would partition the plots into two specific halves which would pertain to classifying the objects into distinct classes.

Circularity vs Mean RGB Value

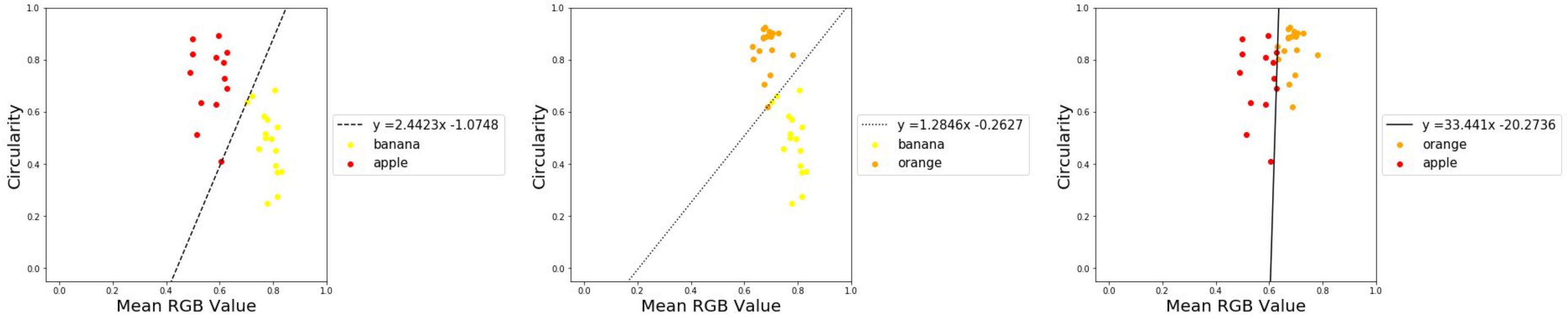


Figure 4. Scatter plots comparing the mean RGB value and circularity of each pair of fruit class. In addition, the perceptron-determined decision line in superimposed on the graph which simulates a division between the two classes. For this pair of features, a distinct separation between the two classes is observed. This would suggest that the features are linearly separable - and is thus comparatively easier to distinguish from one another. An observation was also made in terms of the equation of the line derived from the perceptron.

The slope of the line strongly narrates how steeper slopes tend to get classified with stronger bias towards a single feature. In retrospect, gentler slopes tend to suggest that both features are equally important for distinguishing between fruits. Although, flatter slopes would suggest that the separation is highly dependent on the y-axis feature. These findings suggests that between apples and oranges, the distinction between the two is highly reliant on the mean RGB values in contrast to their circularity. While both features are almost equally important for distinguishing between bananas and apples and between bananas and oranges.

Convexity vs Mean RGB Value

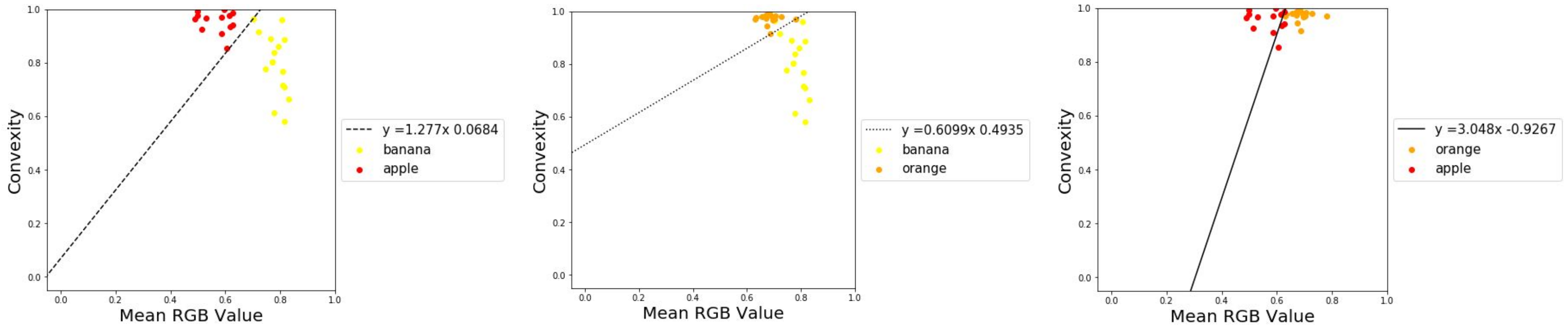
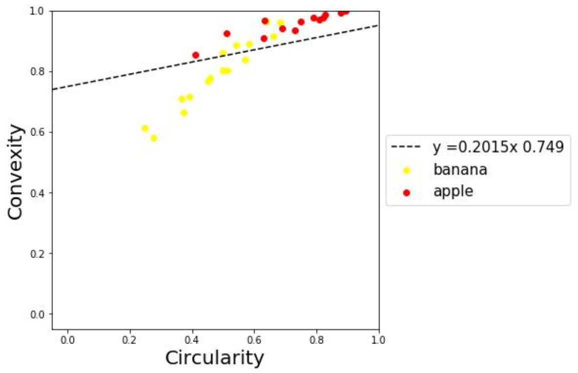
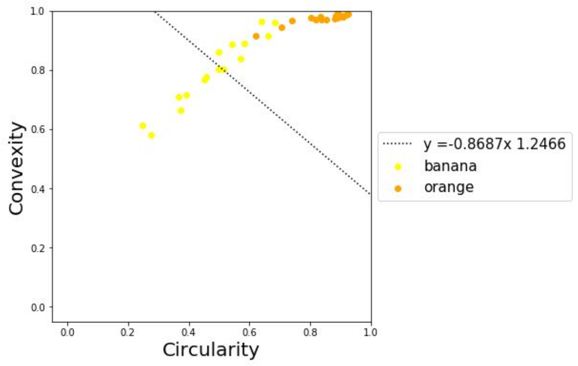


Figure 5. Scatter plots comparing the mean RGB value and convexity of each pair of fruit class. In addition, the perceptron-determined decision line in superimposed on the graph which simulates a division between the two classes. For this pair of features, a distinct separation between the two classes is observed. This would suggest that the features are linearly separable - and is thus comparatively easier to distinguish from one another. An observation was also made in terms of the equation of the line derived from the perceptron.

Using the same concepts mentioned in the previous slide, it can be observed that the mean RGB value and convexity of a banana and an apple are equally important for distinguishing between each other. The same can be said for oranges and apples - although some samples were misclassified which may be due to improper thresholding or cropping of the image during blob analysis. As for differentiation between bananas and oranges, it can be observed that the distinction between each class tends towards the convexity of each sample. Whereas, fruits which exhibited lower convexity (thus are deemed concave) are generally considered as samples of the banana class.

Convexity vs Circularity





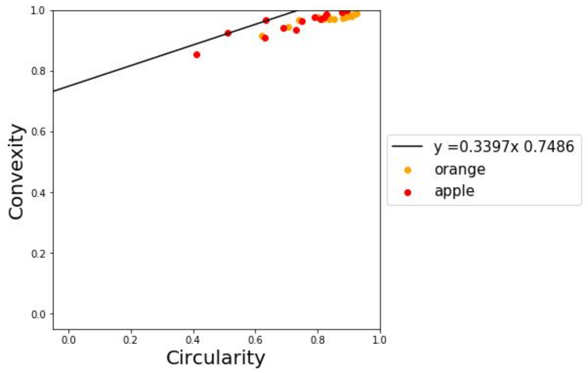


Figure 6. Scatter plots comparing the circularity and convexity of each pair of fruit class. In addition, the perceptron-determined decision line in superimposed on the graph which simulates a division between the two classes. For this pair of features, low accuracy was observed for the classification of samples.

Comparisons for this pair of features yielded a relatively low accuracy for distinguishing between fruits as significant overlaps were found - and thus no distinct approximation for a decision can be made. As this yielded at least one fruit in each pair, it would suggest that some samples are highly similar in terms of these two features. It is although expected for oranges and apples to have similar shapes. Although - upon inspecting the dataset used, it was found that some bananas came in bundles rather than singular bananas and thus would suggest that it would highly affect the circularity and convexity of such samples. This would explain strays found in the first two scatter plots.

SCORE

QoP : 5/5

TC : 5/5

Initiative: 0