186

Mary Chris Go / 2014 11122

# Activity 13: Perceptron

## Goal

To be able to show that the three classes are separable through a decision line

## Perceptron Algorithm

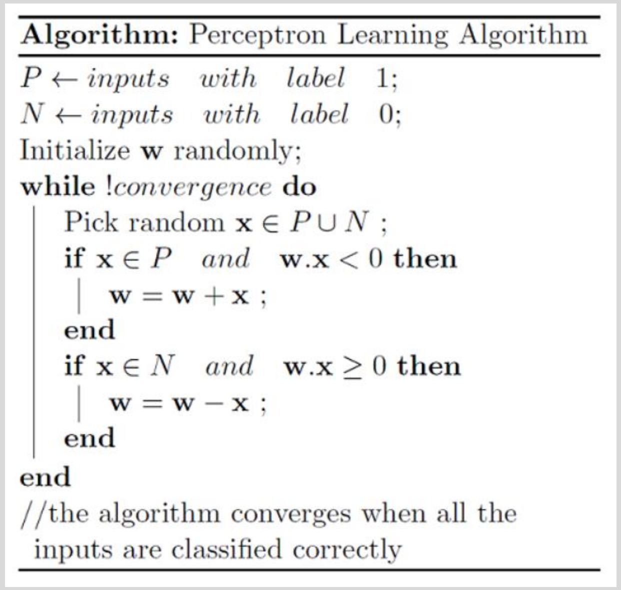
This algorithm is known to perform classification through supervised learning given that the classes are linearly separable in feature space. As I only made a 2D feature space, we expect the decision boundary to be a line.

## Process

We expect three combinations from the three classes. The goal is to find the vector that perfectly divides between these classes. The algorithm shown is the summary of how this works. Basically, we include a constant bias in all inputs. Then we initialize the weights to some random numbers. Therefore, x_0 has a corresponding weight w_0. Now, we calculate the perceptron with this thresholding function:

$$z = g(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

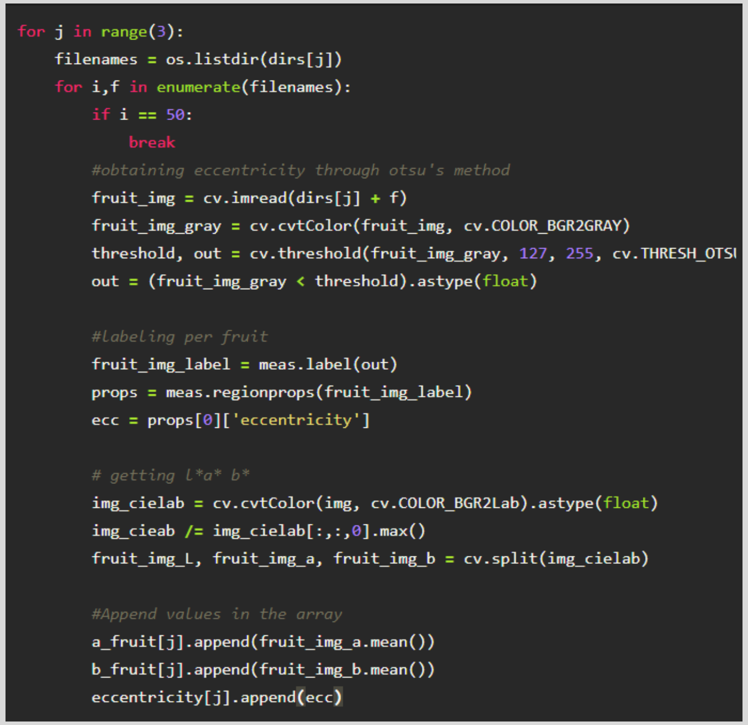
Now, we calculate and update the weight change. Lastly, we iterate through samples.



**Algorithm:** Perceptron Learning Algorithm

```
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ do
    Pick random $\mathbf{x} \in P \cup N$ ;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ then
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ then
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```

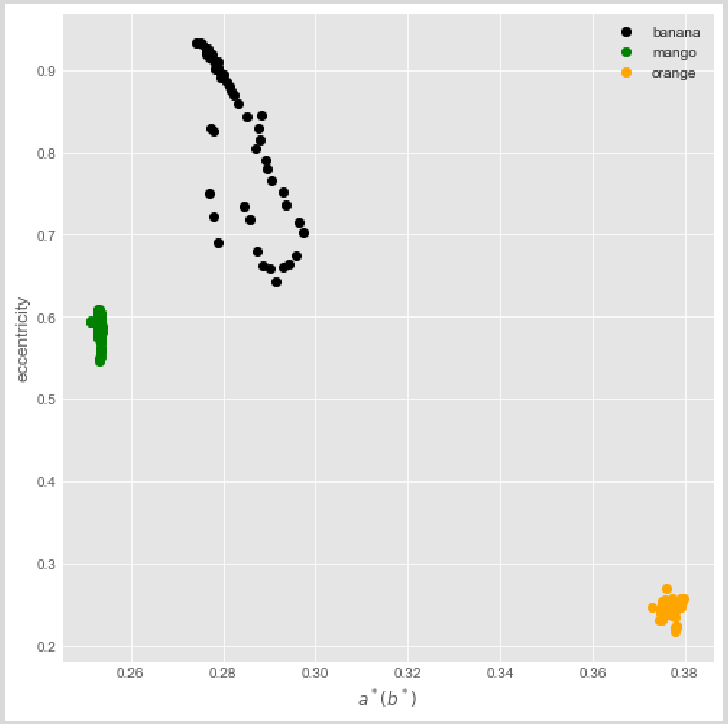
## Python Implementation



```python
for j in range(3):
    filenames = os.listdir(dirs[j])
    for i,f in enumerate(filenames):
        if i == 50:
            break
        #obtaining eccentricity through otsu's method
        fruit_img = cv.imread(dirs[j] + f)
        fruit_img_gray = cv.cvtColor(fruit_img, cv.COLOR_BGR2GRAY)
        threshold, out = cv.threshold(fruit_img_gray, 127, 255, cv.THRESH_OTS
        out = (fruit_img_gray < threshold).astype(float)

        #labeling per fruit
        fruit_img_label = meas.label(out)
        props = meas.regionprops(fruit_img_label)
        ecc = props[0]['eccentricity']

        # getting L*a* b*
        img_cielab = cv.cvtColor(img, cv.COLOR_BGR2Lab).astype(float)
        img_cieab /= img_cielab[:,:,0].max()
        fruit_img_L, fruit_img_a, fruit_img_b = cv.split(img_cielab)

        #Append values in the array
        a_fruit[j].append(fruit_img_a.mean())
        b_fruit[j].append(fruit_img_b.mean())
        eccentricity[j].append(ecc)
```

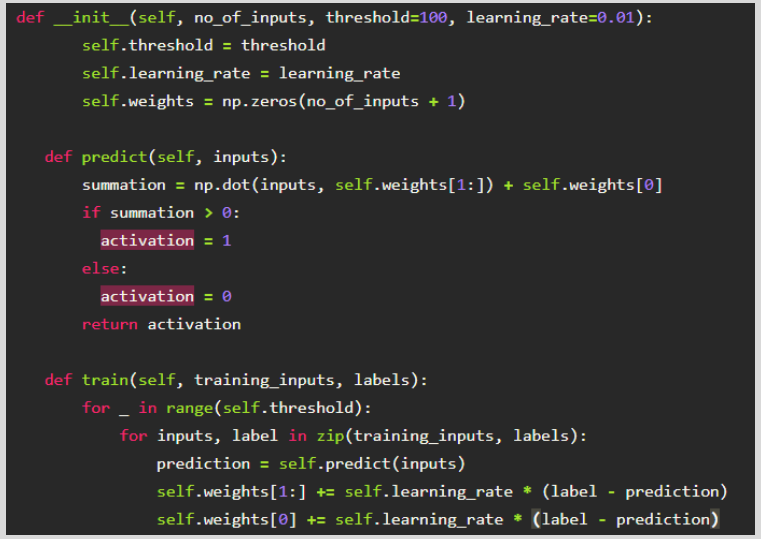
This code was used to determine the features of the images. This was already shown from the las activity, so I'll just attach this here for future references.

## Results from earlier activity

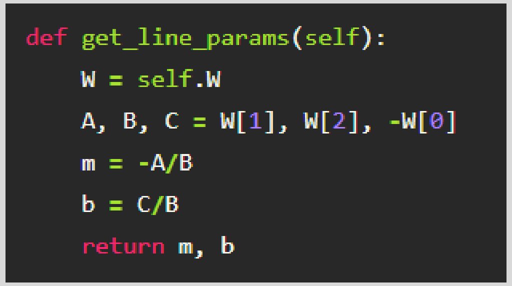


The graph shown is the result from three classes: banana, mango, and orange. We will try to separate them by taking two classes at a time.

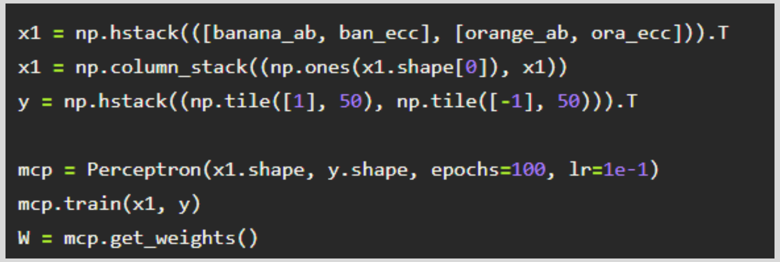
## Python Implementation



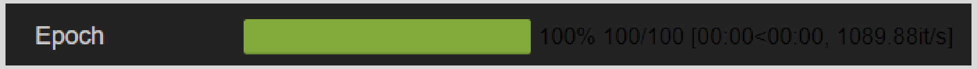
```
def __init__(self, no_of_inputs, threshold=100, learning_rate=0.01):
        self.threshold = threshold
        self.learning_rate = learning_rate
        self.weights = np.zeros(no_of_inputs + 1)

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        if summation > 0:
          activation = 1
        else:
          activation = 0
        return activation

    def train(self, training_inputs, labels):
        for _ in range(self.threshold):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction)
                self.weights[0] += self.learning_rate * (label - prediction)
```



```
def get_line_params(self):
    W = self.W
    A, B, C = W[1], W[2], -W[0]
    m = -A/B
    b = C/B
    return m, b
```



```
x1 = np.hstack(([banana_ab, ban_ecc], [orange_ab, ora_ecc])).T
x1 = np.column_stack((np.ones(x1.shape[0]), x1))
y = np.hstack((np.tile([1], 50), np.tile([-1], 50))).T

mcp = Perceptron(x1.shape, y.shape, epochs=100, lr=1e-1)
mcp.train(x1, y)
W = mcp.get_weights()
```



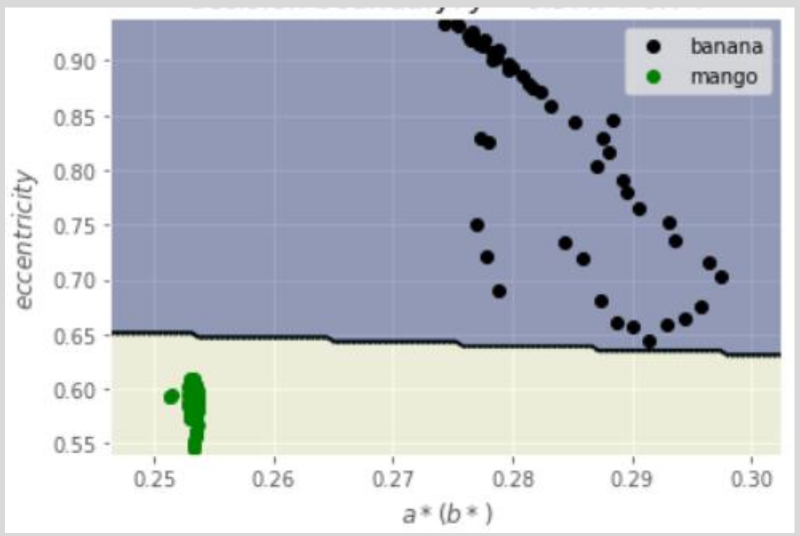
```
Epoch     100% 100/100 [00:00<00:00, 1089.88it/s]
```

The easiest way to attack this problem is to make and object-oriented programming with Python. Inside is train, predict, and obtaining weights and biases. We name this Perceptron. We will use this to train and get weights. In 129ms, 100 epoch was run and you will on the following slides that it is enough to train the algorithm. The epoch details can be seen above.

Through def get_line_params(self), we can get the decision boundary.

From these snippets, we can plot the graph of two classes with a decision line in the center. Three combinations of classes will be shown in the next slide.

## Banana and Mango

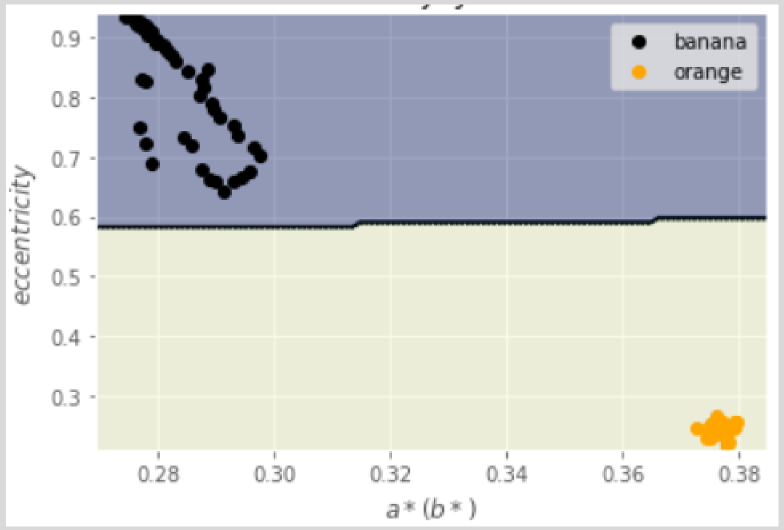


Decision line:

y = 0.37 x + 0.74

The following graph is a decision line we have established for the banana and mango class. The line was able to differentiate the two class.

## Banana and Orange

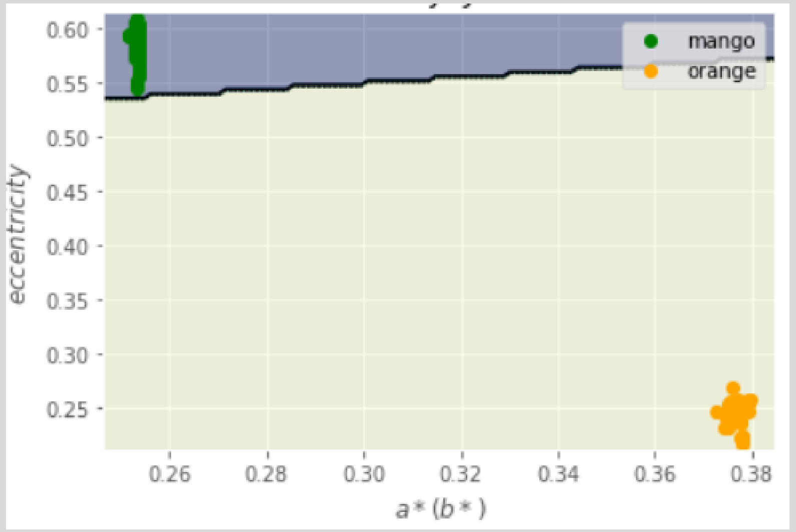


Decision line:

y = 0.15x +0.54

The following graph is a decision line we have established for the banana and orange class. The line was able to differentiate the two. But I was just wondering why is the line near the banana class.

## Mango and Orange



Decision line:

y = 0.28 x + 0.47

The following graph is a decision line we have established for the orange and mango class. There is a line between them. The code was able to differentiate them.

## Summary

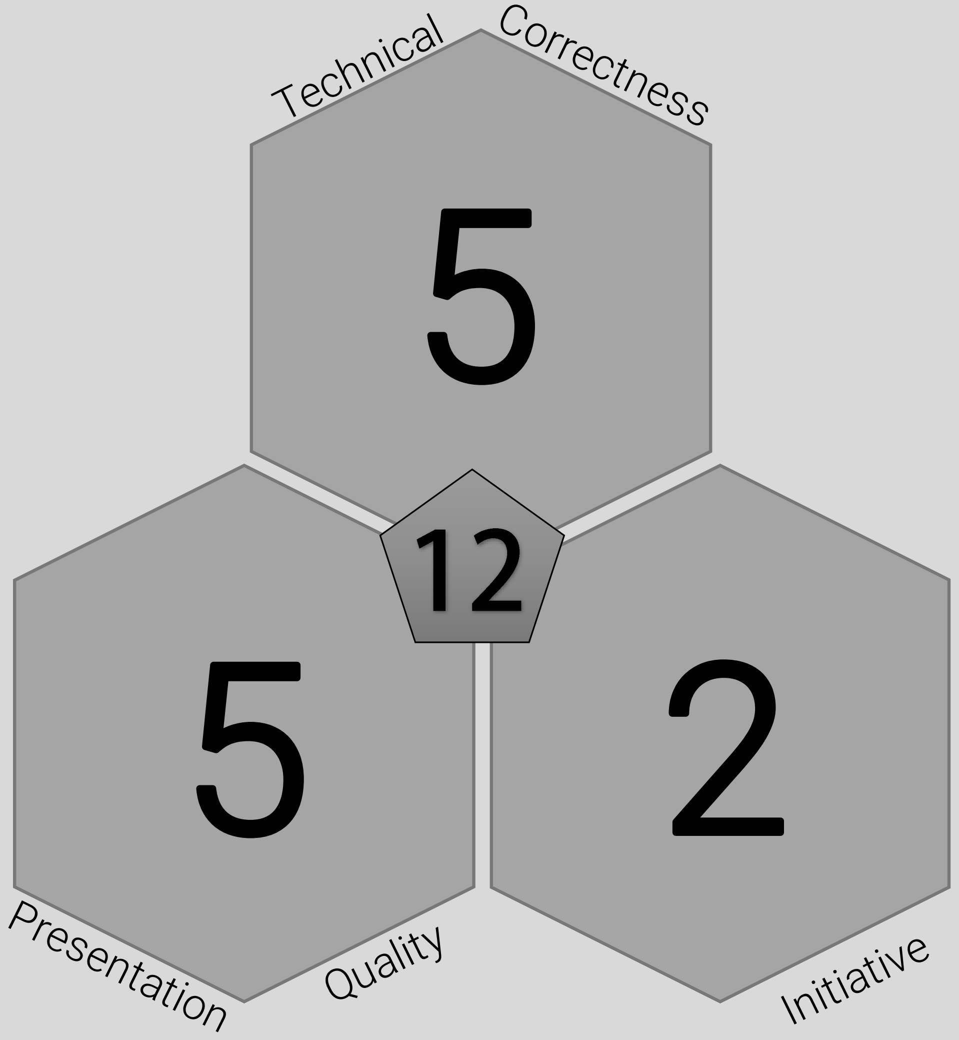
Decision lines per class were shown for different classes and the formula of the line was also obtained. The goal was achieved.

This was the hardest for me especially knowing that there are existing packages that can obtain the decision line. But we had to hardcode everything. The classes were well separated but I cannot still understand the distance between the two classes and what identifies it. I'm not 100% if I was able to execute the procedure right. I still have a lot of things to learn in this topic.

It was a bit frustrating doing this but you gotta love the challenges, right?

Thanks to all my classmates and github who helped me along the way. This was a tough activity.

## Self-Evaluation



## References

- Soriano, M., “Perceptron”. 2019
- Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42, 2018.
- https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975
- https://github.com/zmzhang/TEAM/blob/master/TEAM.py
- https://medium.com/@thomascountz/19-line-line-by-line-python-perceptron-b6f113b161f3