# Computer vision 2021 Assignment 3: image classification

This assignment contains 3 questions split across 2 notebooks. The first two questions, below, require you to complete and extend a simple 2D classifier.

# Question 1: Perceptron training algorithm (30%)

Below is the usual code to import libraries we need. You can modify it as needed.

In [1]:

```
import numpy as np # This is for mathematical operations

# this is used in plotting
import matplotlib.pyplot as plt
import time
import pylab as pl
from IPython import display

%matplotlib inline

from perceptron import *

%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

1. Starting from the boiler plate code provided in `perceptron.py`, implement the perceptron training algorithm as seen in the lecture. This perceptron takes a 2D point as input and classifies it as belonging to either class 0 or 1. By changing the `number_of_samples` parameter, different sample sizes can be generated for both classes.

You need the address the section of the code marked with **#TODO**

In this question, we implemented a single layer perceptron to do the binary classification. The algorithm used is from the lecture "Neuron Networks: Perceptron" in slides 37 shown below:

**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

Two classes has ground truth labels 0 and 1. The training samples are ramdomly selected from X. The program initially with a set of weights, and if a class 1 sample outputs a number less than 0 (a misclassification), the weight of this path will be corrected by increase an amount of x value (increment the weight vector, pushing it towards making a positive classification rather than a negative classification). When an 0 class sample outputs a positive value, the corresponding weights will be corrected by substracting x value. Through several epochs, the weight vector will converge, and the learning process stopped. The algorithm impelmentation in detail is shown below:

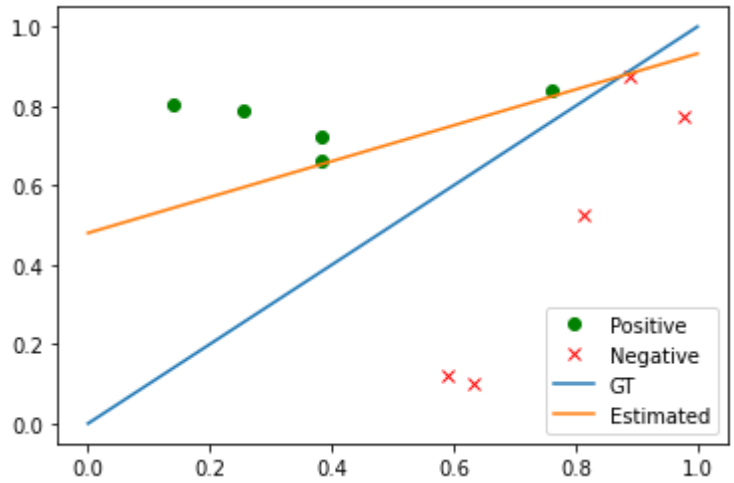
```
def train(self,X,Y, max_epochs = 100):

      # we clear history each time we start training
      self.history = []
      converged = False
      epochs = 0

      while not converged and epochs < max_epochs :
            converged = True
            learning_rate = 0.001
            count = 0
            for i in range(len(X)):
                  wxi = np.matmul(X[i],self.w)
                  if wxi >= 0:
                        wxi = 1.0
                  else:
                        wxi = 0.0
                  if wxi != Y[i]:
                        count +=1
                        converged = False
                        self.w = self.w + (Y[i] - wxi)*learning_rate*(X[i].reshape(-1,1))

            self.compute_train_accuracy(X,Y)
            epochs +=1

      if epochs == max_epochs:
            print("Qutting: Reached max iterations")

      if converged:
            print("Qutting: Converged with epoch of :", epochs)

      self.plot_training_history()
```

The plots below are the learning behaviours for different sample sizes of 20, 80, 800 and 3000.

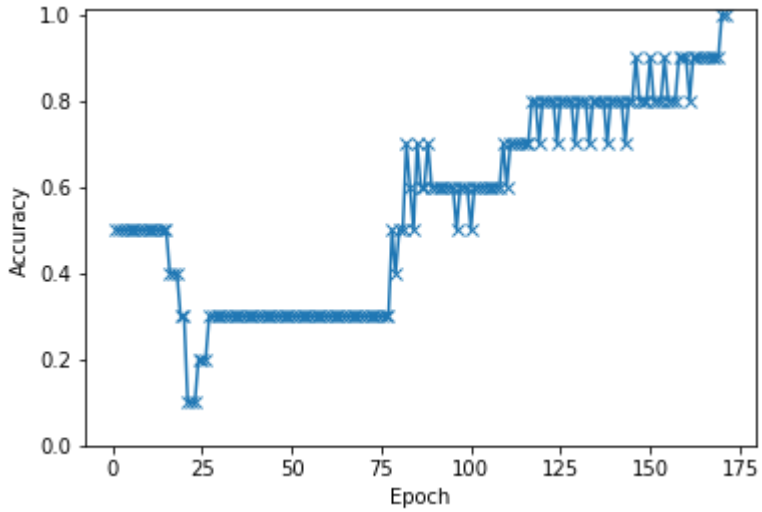
Number_of_samples: 10; max_number_of_epochs: 200; learning_rate: 0.01

In [191]:

```
#0.01
number_of_samples = 10
max_number_of_epochs = 200

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```



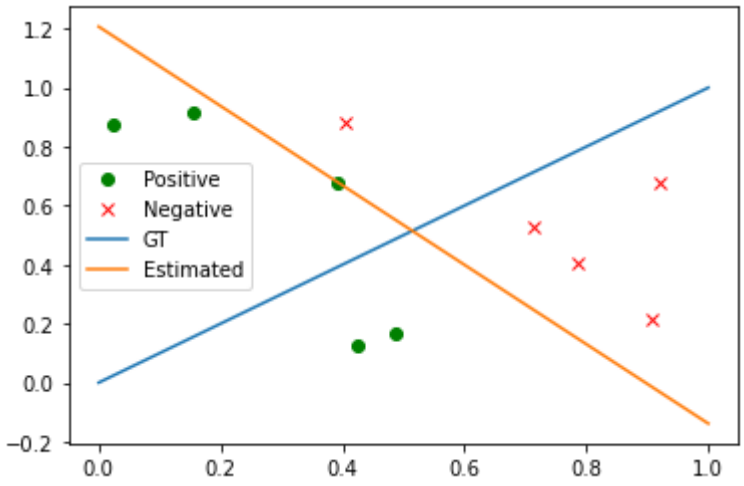

Qutting: Converged with epoch of : 171



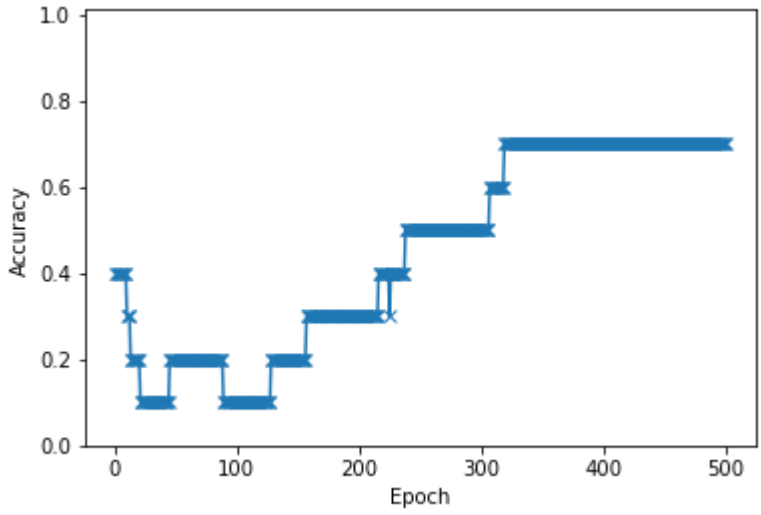

Number_of_samples: 10; max_number_of_epochs: 500; learning_rate: 0.001

In [200]:

```
#0.001
number_of_samples = 10
max_number_of_epochs = 500

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```



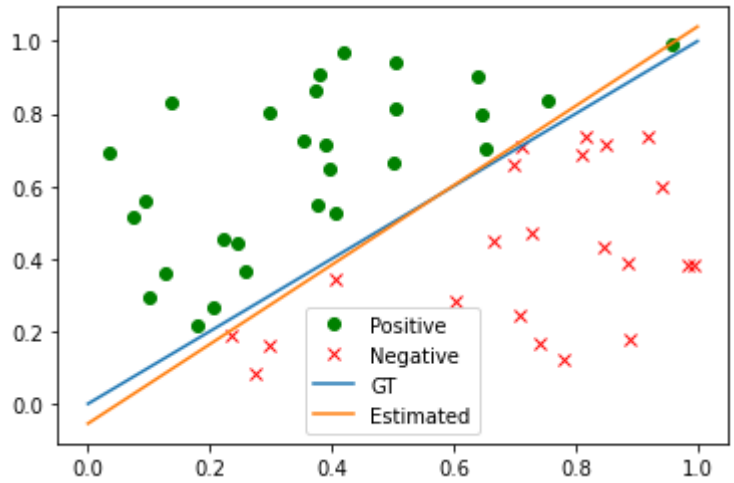

Qutting: Reached max iterations



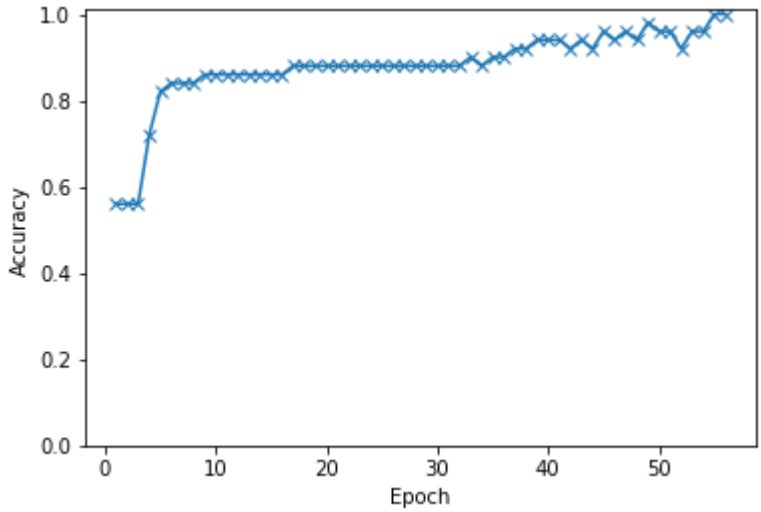

Number_of_samples: 50; max_number_of_epochs: 200; learning_rate: 0.01

In [194]:

```
#learning rate of 0.01
number_of_samples = 50
max_number_of_epochs = 200

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```



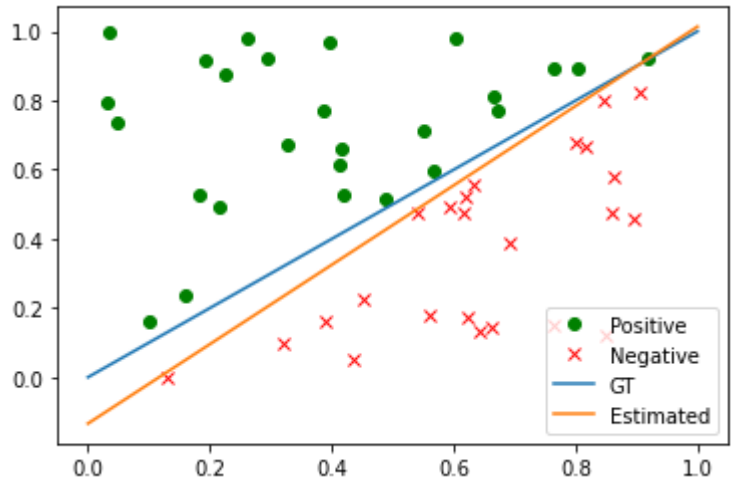

Qutting: Converged with epoch of : 56



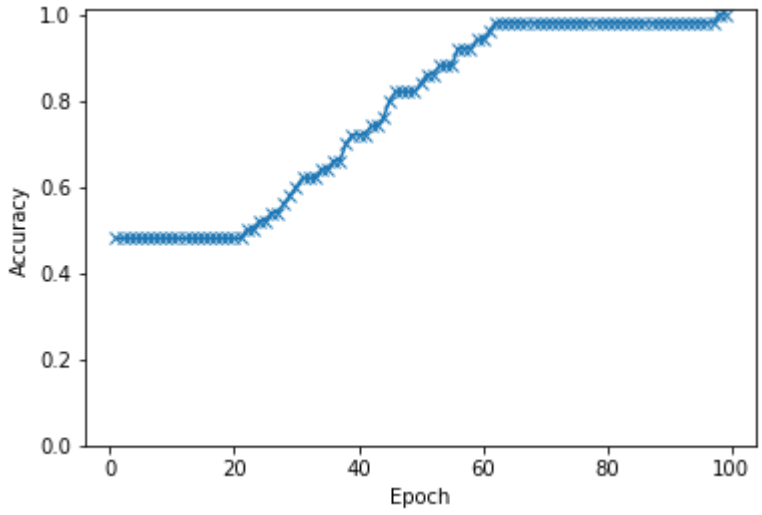

Number_of_samples: 50; max_number_of_epochs: 300; learning_rate: 0.001

In [201]:

```
#learning rate of 0.001
number_of_samples = 50
max_number_of_epochs = 300

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```



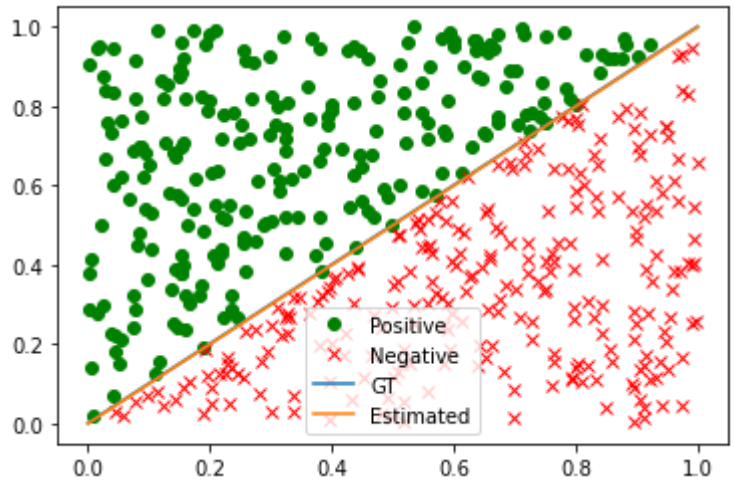

Qutting: Converged with epoch of : 99



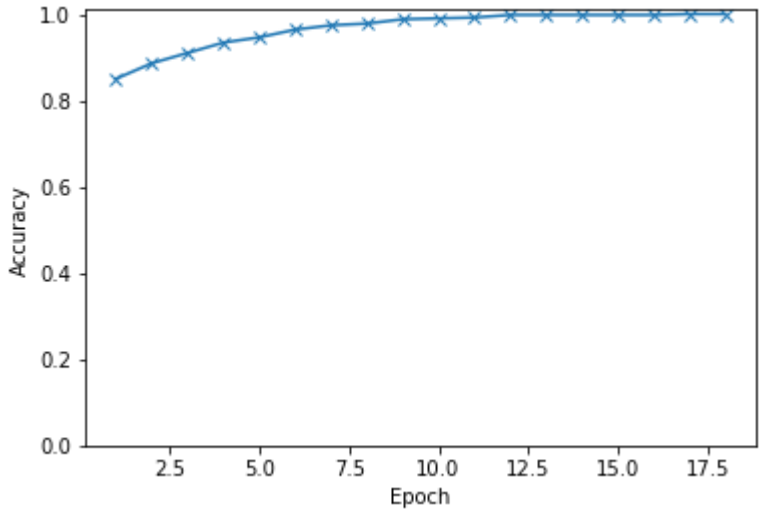

Number_of_samples: 500; max_number_of_epochs: 300; learning_rate: 0.01

In [195]:

```
#learning rate of 0.01
number_of_samples = 500
max_number_of_epochs = 300

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```



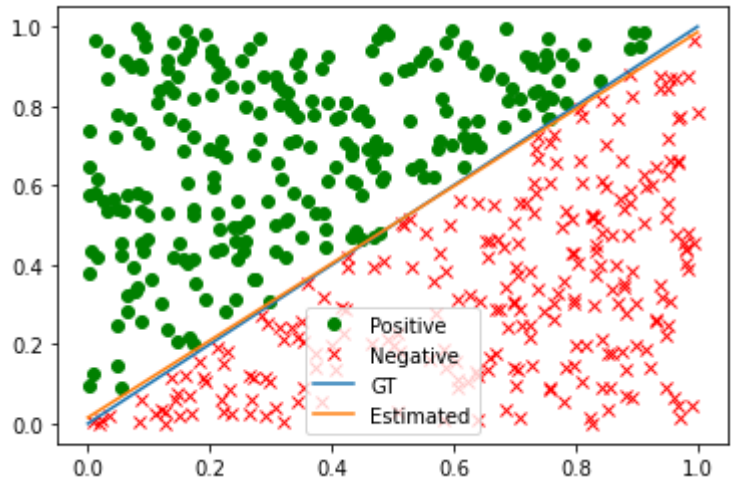

Qutting: Converged with epoch of : 18



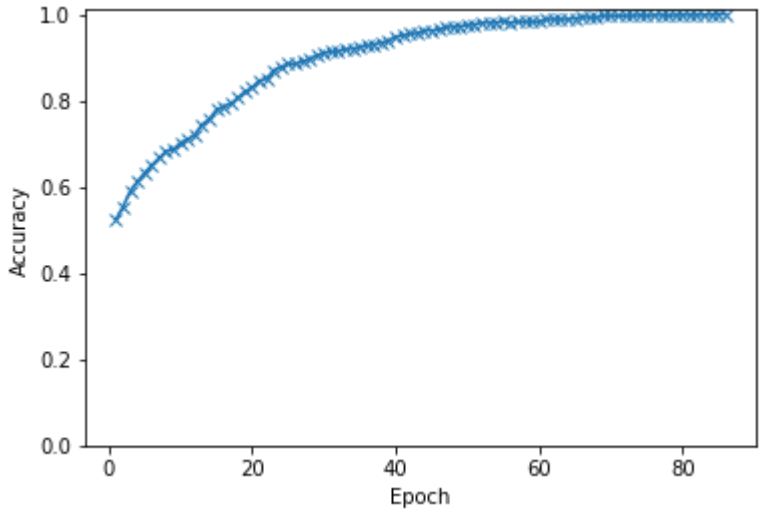

Number_of_samples: 500; max_number_of_epochs: 400; learning_rate: 0.001

In [205]:

```
#0.001
number_of_samples = 500
max_number_of_epochs = 400

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```




Qutting: Converged with epoch of : 86

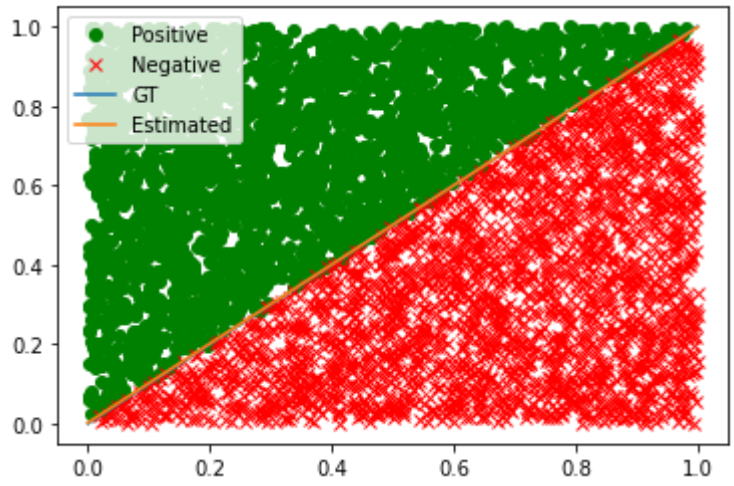



In [ ]:

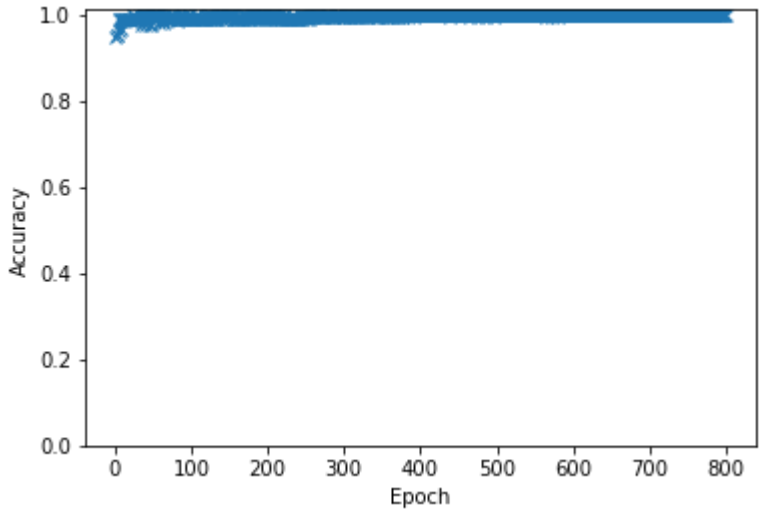
```
Number_of_samples: 5000; max_number_of_epochs: 800; learning_rate: 0.01
```

In [196]:

```
#learning rate of 0.01
number_of_samples = 5000
max_number_of_epochs = 800

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```



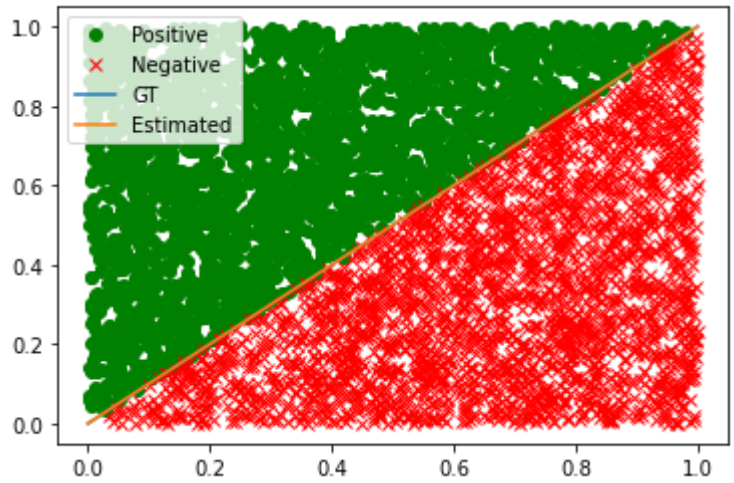

Qutting: Reached max iterations



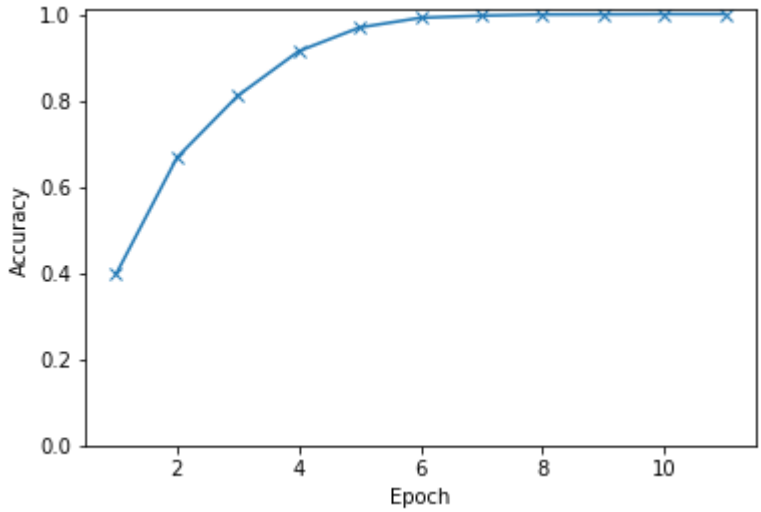

Number_of_samples: 5000; max_number_of_epochs: 800; learning_rate: 0.001

In [206]:

```
#learning rate of 0.001
number_of_samples = 5000
max_number_of_epochs = 800

X = np.random.rand(number_of_samples,2)
X = np.append(X, np.ones((X.shape[0],1)),axis = 1)

Y = X[:,1] > (X[:,0])
Y = np.float32(Y)
Y = Y.reshape((number_of_samples,1))

p = Perceptron(3)
p.train(X,Y,max_number_of_epochs)
```




Qutting: Converged with epoch of : 11




1. The code plots the ground-truth boundary between the two classes. We know this is the ground-truth because this is the boundary from which we generated data. However, even though the estimated line can correctly classify all the training samples, it might not agree with the ground-truth boundary.

- Explain why this happens.
- What is the potential disadvantage of such a decision boundary?
- How could you change the training algorithm to reduce this disadvantage? (Implementation not required)

The perceptrons behaviour for different sample sizes are illustrated above. It is shown that the accuracy increases as the sample size increases. The perceptron must be trained and callibrated for at least number_of_samples times (if there are more epoch, it will be trained through more iterations). More training and corrections leads to better classification performance. Another interesting observation is that even though the estimated line can correctly classify all the training samples, it might not agree with the ground-truth boundary. This phenomenon makes sense, because the perceptron is trained based on the generated data set. It is trained to seperated different data, and it did the job. The ground truth line can be interpreted as an output based on all these infinite number of points in this plane. The blank space for the ground truth is also a part its data, but for our perceptron, there are unseen data. This also explains the cases with large number_of_samples where has less empty space, so there are less space for weights vector to converge.

1. In some training sessions, you might observe the boundary oscillating between two solutions and not reaching 100% accuracy. Discuss why this can happen and modify the training algorithm to correct this behaviour. We will call this the modified algorithm. (Hint: learning rate)

The perceptron learning algorithm I used is gradient decend with a learning rate of 0.01. The learning rate is like the step size. The learning rate controls how much the weights change in each training iteration. A higher learning rate may increase training speed, but may take big steps oscillating between positive and negative until converge at a minimum. For sample size of 5000 case, a learning rate of 0.01 still cannot converge to 100% accuracy even after 800 epochs, while a reduced learning rate of 0.001 performs much better than the 0.01 case. The reason behind this is a large number of samples leads to a less space for convergence on the plane, so that the estimated line needs to be adjusted slightly and precisely. However, this cannot be achieved through a big learning rate of 0.01. 0.01 is a big step size for a proper adjustment, and each time the perceptron will oscillate between positive and negative, and hard to position itself to that precise line. However, a learning rate of 0.001 can easily solve this issue. And as shown in the plot above. A learning rate of 0.001 converged just through 11 epochs.

1. Random initialization causes the algorithm to converge in different number of epochs. Execute the training algorithm on sample sizes of 10, 50, 500 and 5000. Report in a table the mean number of epochs required to converge for both the original and the modified algorithm. Which algorithm performs better and why? Is there a clear winner?

| (number_of_samples, learning_rate) | 0.01 | 0.001 |
|---|---|---|
| 10 | 171 | Not converged |
| 50 | 56 | 99 |
| 500 | 18 | 86 |
| 5000 | Not converged | 11 |

The table above is the average epochs numbers for different number_of samples and learning_rate through many trails and observations. The table shows that, as the number_of_samples increases, there are less epochs needed for convergence for both algorithms. This is because as the number of samples increases, there will be more samples to be trained by the perceptron in each epochs, therefore, less epochs needed. Another interesting observation is that a bigger learning rate can converge for the cases with samll number_of_samples, and a small learning rate can converge with bigger number_of_samples, such as the first row and the last row. In the first row, 0.001 is such a small step for gradient decend of 10 samples. Each iterations the estimated line was just adjusted a little bit, and there are large blank space for making adjustments, but a learning rate of 0.001 is too small for making a proper adjustment. However, this can eventually converge, but it just need a very large number of epochs, and in this case, the max epochs is set to be 500, which still needs to be tuned up for congergence. For last row, 5000 samples cannot converge for a learning rate of 0.01. This is as discussed before, this is such a big step size for this setting. There are 5000 points in the plane, and very less space for performing a very precise convergence. A learning rate of 0.01 made the perceptron oscillating between positive and negative until an maximum number of epochs was reached.
Therefore, there is no clear winner, it's all depends on the other settings, such as number of samples and maximum epochs etc. But if I have to pick one, I will choose the learning rate of 0.001, even through it needs to take a large number of epochs in samll sample base problem, but it can converge with an correct answer eventually with many epochs.

# Question 2: Multiclass classifier (30%)

Pat yourselves on the back, you have successfully trained a binary classifier. Now, its time to move to 3 classes. The dataset we are going on work on is shown below and contains three classes represented by different colors.

In [2]:

```
import scipy.io as sio

data = sio.loadmat('training_data.mat')
X = np.array(data['X'])

X = np.append(X.T, np.ones((X.shape[1],1)),axis = 1)
Y = np.array(data['Y'])

print('Training data shape:',X.shape)
print('Labels shape:', Y.shape)
```

```
Training data shape: (15468, 3)
Labels shape: (15468, 1)
```

1. Extend the algorithm developed in Question 1 to distinguish between the three classes. Discuss the modification that were needed to extend the functionality to 3 classes. (Hint: One vs. All classification)

One-vs-rest (OvR for short, also referred to as One-vs-All or OvA) is a heuristic method for using binary classification algorithms for multi-class classification.It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident. This approach is that it requires one model to be created for each class, therefore, we have 3 base cases.

| | | |
|---|---|---|
| Base case 1 | Class 0: T | Class 1: F; Class2: F |
| Base case 2 | Class 1: T | Class 0: F; Class2: F |
| Base case 3 | Class 2: T | Class 0: F; Class1: F |

In the modified code, one weights vector is changed to 3 vectors for each three classes. In each epoch, 3 weights vectors are all trained based on the modified training labels Y. Training labels are modified in this task, because in each base case, the training labels are different. The modified training labels Y are shown below. These label sets basically state the 3 base cases in the tabel above.

```
Y0[Y0 == 1], Y0[Y0==0], Y0[Y0==2] = 0, 1, 0
Y1[Y1 == 0], Y1[Y1==2], Y1[Y1==1] = 0, 0, 1
Y2[Y2 == 1], Y2[Y2==0], Y2[Y2==2] = 0, 0, 1
```

Next, the 3 weight vectors for 3 base cases are trained based on its own base case training label. For example, the weight vector in base case 1 is trained for distinguishing the class 0 from class 1 and 2, and then a linear classification line will be drawn between the class 0 and other two classes after convergence. The training process for other base cases are the same process, but with different training aims (different Y). Thus, in the end,there will be 3 lines seperating its target class from the rest two classes, which form a multi-classes classification.

```
for i in range(len(X)):
      wxi = np.matmul(X[i],self.w0)
      if wxi >= 0:
            wxi = 1.0
      else:
            wxi = 0.0
      if wxi != Y0[i]:
            converged = False
            self.w0 = self.w0 + (Y0[i] - wxi)*learning_rate*X[i].reshape(-1,1)

      wxi = np.matmul(X[i],self.w1)
      if wxi >= 0:
            wxi = 1.0
      else:
            wxi = 0.0
      if wxi != Y1[i]:
            converged = False
            self.w1 = self.w1 + (Y1[i] - wxi)*learning_rate*(X[i].reshape(-1,1))

      wxi = np.matmul(X[i],self.w2)
      if wxi >= 0:
            wxi = 1.0
      else:
            wxi = 0.0
      if wxi != Y2[i]:
            converged = False
            self.w2 = self.w2 + (Y2[i] - wxi)*learning_rate*(X[i].reshape(-1,1))
```

1. By modifying the function `compute_train_accuracy`, plot the accuracy over time for the training data. Explain how you changed the accuracy measure for more than 2 classes.

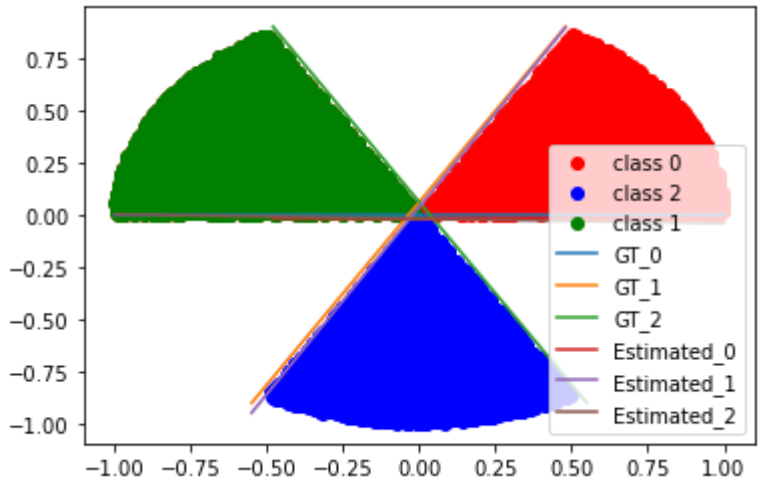
In a one vs all classification problem, the classification is accuracte only when the all classes are classified correctly, i.e. meet the above base case conditions. The whole system converges only when all three classes converges, i.e. all reached 100% accuracy. Therefore, most of accuracy calculation methods are kept as original, but the algorithmm is extended to 3 classes. The accuracy for each 3 class is claculated using original given method, then all the accuracy is added up and divided by the total number of classes. In another word, the average of the accuracies is the representation of overall accuracy for the whole system.The accuracy vs time plot is shown below.

1. Visualize the decision boundaries on the given dataset by implementing the draw() function for the modified algorithm and include it in your report. Based on your observation of the decision boundaries, discuss why a linear classifier is still the correct choice for this dataset.

from the plot below, each classes are seperated by straight lines i.e. linear classified. All three classes of data are located in different groups, which can be linearly seperated. Therefore, even we have to classify 3 classes, this is still a linear classification problem, which can be solved by a combination of linear classifiers.

In [4]:

```
#learning rate of 0.001
number_of_samples = 100
max_number_of_epochs = 60

p = Perceptron(3)
p.train_OvR(X,Y,max_number_of_epochs)
```




Qutting: Converged with epochs of : 30