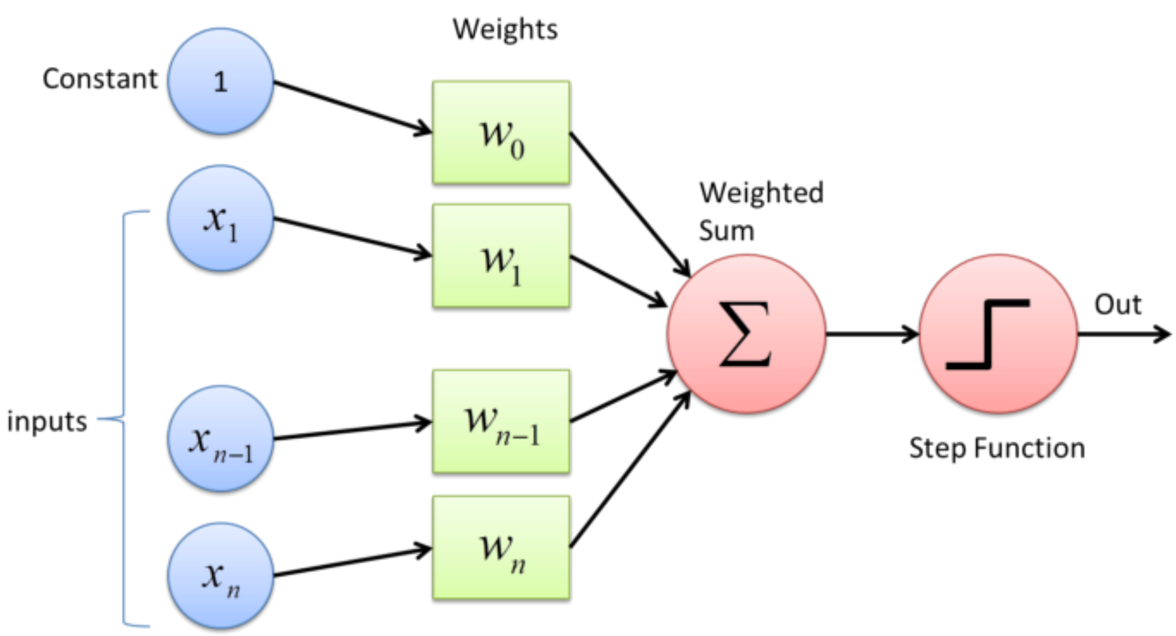
19290116
MOHAMMAD SHABIB

PART 1:

What is Perceptron?:

Perceptron (artificial neuron) is considered one of the early version of modern neural networks
Its consists of a single-layer neural network and is used as a linear(binary) classifier in supervised classification. Perceptron is also considered an iterative algorithm.

- It consists of a vector of inputs(including the constant 1-for bias-) X.
- It consists of a vector of weights(including the bias) W.
  - Bias value allows us to move the activation function curve up or down.
- Weighted sum.
- It consists of an activation function to map the input to the desired output (binary output) e.g: (0,1).



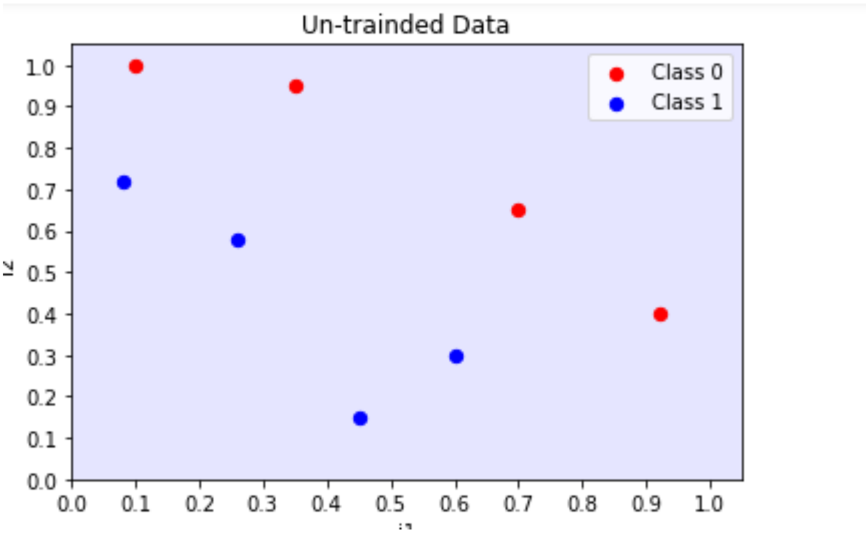
Applying the Algorithm by hand to some dataset:

Input	Output
[0.08,0.72]	-> 1
[0.1, 1]	-> 0
[0.26,0.58]	-> 1
[0.35,0.95]	-> 0
[0.45,0.15]	-> 1
[0.6,0.30]	-> 1
[0.7, 0.65]	-> 0
[0.92, 0.4]	-> 0

0 -> Red Class
1 -> Blue Class

PS: We will add extra input for the input vector with value 1 (for bias)
Initial Weights:
[0, 0, 0]

Data before being classified, everything will be blue according to the initial weights and the activation function.
Activation function = 1 if weighted_sum>= 0 else 0



So we calculate the new weights for each input
Wij`= Wij + learning rate * (Actual Output - Predicted Output)*xij, i = which weight for input, J = which input.
Delta_Error

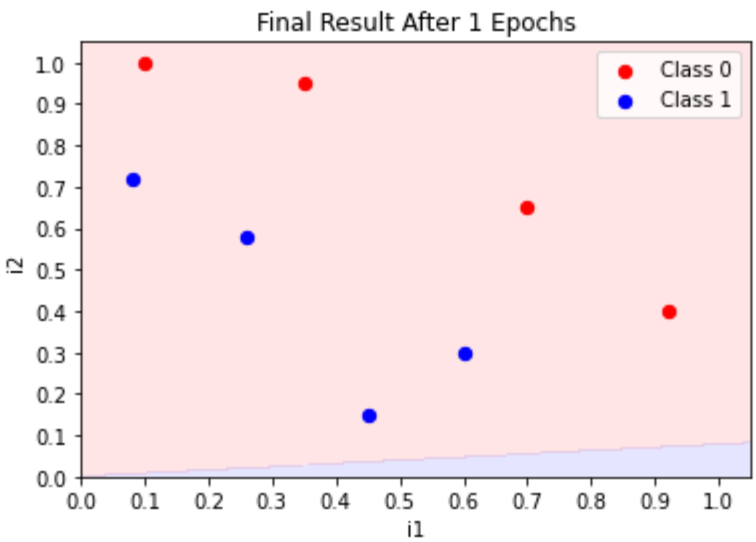
We will use learning rate = 0.2
Lower Lrate means need more epochs to train but will be more accurate than higher Lrate which needs fewer epochs but will give more incorrect results on the test

W11` = 0 + 0.2*(1-1)*1 = 0
W21` = 0+0.2*0*0.08 = 0
W21` = 0+0.2*0*0.72 = 0
And so on
We do this for every input every epoch and we reach 100% accuracy.
When we apply this for the 2th input ([0.35,0.95] -> 0) in the first epoch
W12` = 0+ 0.2*-1*1 = -0.2
W22’ = 0+ 0.2*-1*0.1 = -0.02
W32` = 0+ 0.2*-1*1 = -0.2

[0.   0.032 -0.084]
[0.   0.032 -0.084]
[0.   0.032 -0.084]
[0.2   0.152 -0.024]
[0.   0.012 -0.154]
[0.   0.012 -0.154]

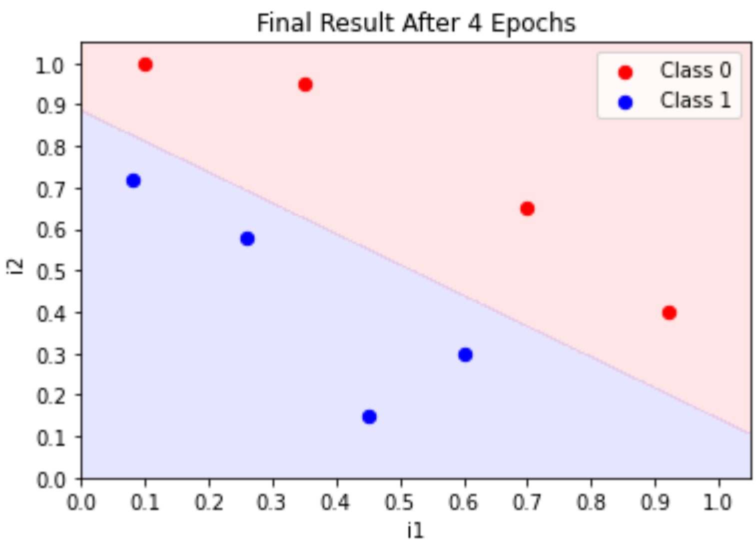
In the 2ndEpoch, the weights will be:
[0.   0.012 -0.154]
And the accuracy will be 50% so we keep training

In the 3rd epoch:
Weights: [0.   -0.06 -0.384], Accuracy: 50%
In the 4rd epoch:
Weights: [0.   -0.184 -0.37], Accuracy: 50%
In the 5rd epoch:
Weights: [0.2   -0.168 -0.226], Accuracy: 100%



And so on
When we reach the 5th epoch the accuracy will be 100% (so it took 4 epochs to train)
And the final weights will be:

Final Weights: [0.2   -0.168 -0.226]



<u>The code below represents the Perceptron algorithm for linear classification:</u>

jupyter Perceptron Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [8]:
```python
#19290116
#MOHAMMAD SHABIB
import numpy as np
from matplotlib import pyplot as plt
```

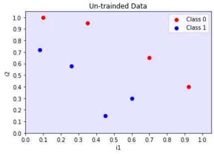
In [9]:
```python
#train the data by predicting the result and edit the weight each time
#until the accurecy is 100% or reachs the max. n_epochs
def Train_Data(inputs, outputs, weights, l_rate = 0.1, n_epoch=100):
    for epoch in range(1,n_epoch+1):
        if accuracy(inputs, outputs, weights) == 1.:
            break;
        for i in range(inputs.shape[0]):
            delta_error =  outputs[i] - predict(inputs[i], weights)
            for j in range(len(weights)):
                weights[j] = weights[j] + l_rate*delta_error*inputs[i][j]
    return epoch
```

In [10]:
```python
#predict the output based on the current weights
def predict(inputs, weights):
    weighted_sum = 0
    for i, w in zip(inputs,weights ):
        weighted_sum += i*w
    return 1 if weighted_sum>= 0 else 0
```

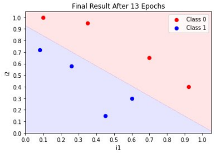
In [11]:
```python
#check the accuracy.
def accuracy(inputs, outputs, weights):
    n_correct = 0
    for i in range(inputs.shape[0]):
        if predict(inputs[i], weights) == outputs[i]:
            n_correct +=1

    return n_correct/len(outputs)
```

In [12]:
```python
#This code wasn't writter by me,
#(I just edited some bits of code to suit my code)
def plot(inputs, outputs,weights,title="Prediction Matrix"):
    fig,ax = plt.subplots()
    ax.set_title(title)
    ax.set_xlabel("i1")
    ax.set_ylabel("i2")


    map_min=0.0
    map_max=1.1
    y_res=0.001
    x_res=0.001
    ys=np.arange(map_min,map_max,y_res)
    xs=np.arange(map_min,map_max,x_res)
    zs=[]
    for cur_y in np.arange(map_min,map_max,y_res):
        for cur_x in np.arange(map_min,map_max,x_res):
            zs.append(predict([1.0,cur_x,cur_y],weights))
    xs,ys=np.meshgrid(xs,ys)
    zs=np.array(zs)
    zs = zs.reshape(xs.shape)
    cp=plt.contourf(xs,ys,zs,levels=[-1,-0.0001,0,1],colors=('b','r'),alpha=0.1)


    c1_data=[[],[]]
    c0_data=[[],[]]
    for i in range(len(outputs)):
        cur_i1 = inputs[i][1]
        cur_i2 = inputs[i][2]
        cur_y  = outputs[i]
        if cur_y==1:
            c1_data[0].append(cur_i1)
            c1_data[1].append(cur_i2)
        else:
            c0_data[0].append(cur_i1)
            c0_data[1].append(cur_i2)

    plt.xticks(np.arange(0.0,1.1,0.1))
    plt.yticks(np.arange(0.0,1.1,0.1))
    plt.xlim(0,1.05)
    plt.ylim(0,1.05)

    c0s = plt.scatter(c0_data[0],c0_data[1],s=40.0,c='r',label='Class 0')
    c1s = plt.scatter(c1_data[0],c1_data[1],s=40.0,c='b',label='Class 1')

    plt.legend(fontsize=10,loc=1)
    plt.show()
    return
```

In [13]:
```python
inputs = np.array([[0.08,0.72],
                   [0.1, 1],
                   [0.26,0.58],
                   [0.35,0.95],
                   [0.45,0.15],
                   [0.6,0.30],
                   [0.7, 0.65],
                   [0.92, 0.4]])
bias = np.ones((inputs.shape[0],1))
inputs = np.concatenate([bias,inputs ],axis=1) ## attaching the bias input

outputs = np.array([1, 0, 1, 0, 1, 1, 0, 0], dtype=float)

weights = np.random.rand(inputs.shape[1]) #I decided to make the weights random [0,1)
intial_weights = np.array(weights, copy=True)

plot(inputs, outputs,weights, "Un-trainded Data")

final_epoch = Train_Data(inputs, outputs, weights, 0.1,100)

print("Intial Weights : ",intial_weights, "\nFinal Weights:", weights)

plot(inputs, outputs,weights, "Final Result After %d Epochs"%(final_epoch-1))
```




```
Intial Weights :  [0.68622898 0.76750569 0.79523367]
Final Weights: [ 0.18622898 -0.17449431 -0.20076633]
```



In [ ]:

Applying the Algorithm By handwritting

we choose random at first (we can also start with fixed values)

bias	X_1	X_2	w_bias	w_1	w_2	net	pre-y	actual_y
1	0	0	-0.2	0.1	0.2	-0.2	0	0
1	1	0	-0.2	0.1	0.2	-0.1	0	1
1	0	1						1
1	1	1						1

change w to w'

we choose 0.2

$$w'_1 = w_i + \underline{\text{learing\_rate}} * (\text{Actual-y} - \text{pre-y}) * X_i$$

$$w'_{bias} = -0.2 + 0.2 * (1) * 1 = 0$$

$$w'_1 = 0.1 + 0.2 * 1 * 1 = 0.3$$

$$w'_2 = 0.2 + 0.2 * 1 * 0 = 0.2$$

bias	X_1	X_2	w_bias	w_1	w_2	net	pre-y	actual_y
1	0	0	-0.2	0.1	0.2	-0.2	0	0
1	1	1	-0.2	0.1	0.2	0	0	1
1	0	1	0	0.3	0.2	0.2	1	1
1	1	1	0	0.3	0.2	0.5	1	1

So weights
[0, 0.3, 0.2]

we check it again

bias	X_1	X_2	w_bias	w_1	w_2	net	pre-y	actual_y
1	0	0	0	0.3	0.2	0	0	0
1	1	0	0	0.3	0.2	0.3	1	1
1	0	1	0	0.3	0.2	0.2	1	1
1	1	1	0	0.3	0.2	0.5	1	1

So No change happened

Final weights

[0, 0.3, 0.2]

PART 2:

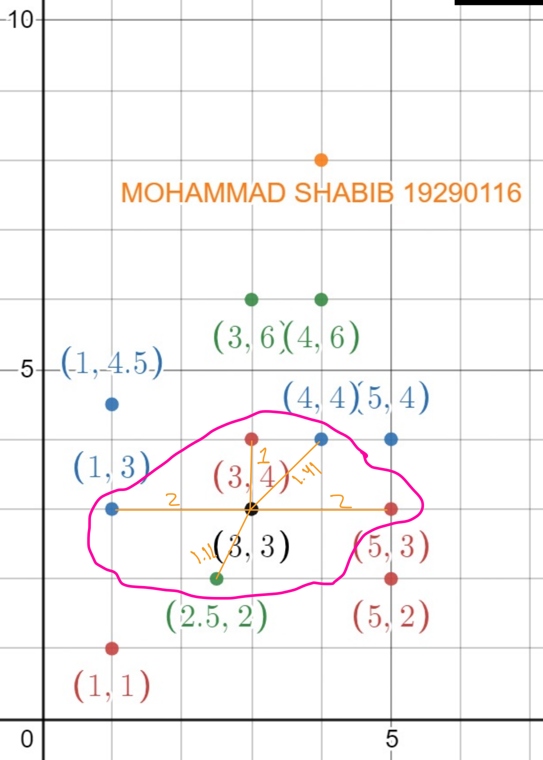
What is k-nearest neighbors(KNN)?:

It's an algorithm that can be considered as classification and regression,
the output depends on the most common class of the K nearest neighbors.
But for the weighted KNN algorithm distance plays a role, the more distance the neighbor has the less significance it has so (importance ~ 1/distance)

- Nearest Neighbor is considered a Lazy Algorithm because it doesn't learn a discriminative function from the training data but "memorizes" the training dataset.
- I implemented my code using the **weighted** k-nearest neighbor's algorithm.

Weighted KNN

$k = 5$



Point	distance	weight $(\frac{1}{d})$	
(3,4)	1	1	1.5
(5,3)	2	0.5	
(1,3)	2	0.5	1.2
(4,4)	1.41	0.7	
(2.5,2)	1.11	0.9	0.9

So the point (3,3) will be classified as red but if we used normal KNN we would have had an issue since 2 classes have the same popularity, then we would have needed to change k to 6 or something else jupyter Weighted-KNN-SmallExample Last Checkpoint: 9 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [99]:
```python
#19290116
#MOHAMMAD SHABIB
import numpy as np
import pandas as pd
from sklearn import datasets #this import is used for using the iris dataset
from sklearn.model_selection import train_test_split #this import is used to split the data into training and test
```

In [100]:
```python
#distance between two node
def euclidean_distance(x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))

#consider the weight of the node not just the popularity of the class
def weight_sum(k_weighted_labels):
    df = pd.DataFrame(k_weighted_labels, columns = ["label", "weight"])
    df = df.groupby(by=["label"]).sum()
    df.reset_index(inplace=True)
    df.sort_values(by=['weight'], ascending=False, inplace=True)
    return df["label"].iloc[0]
```

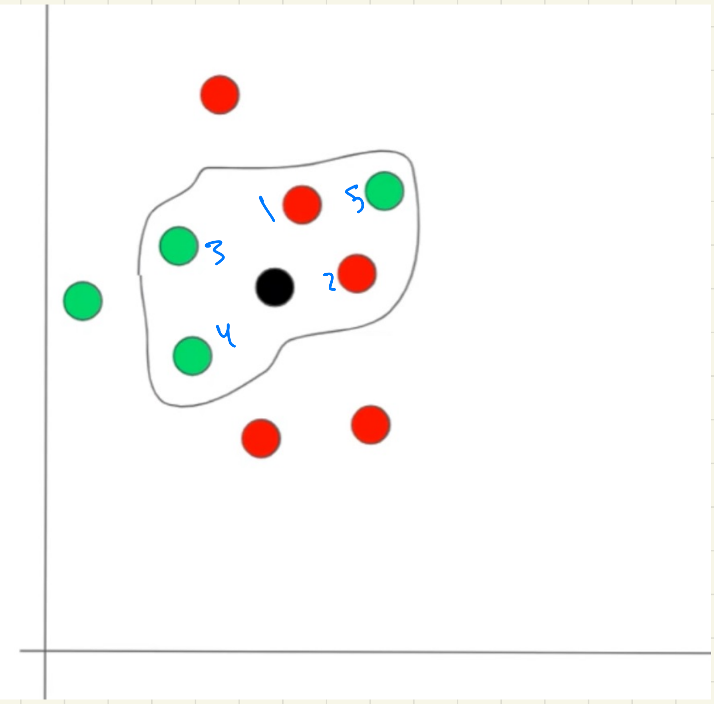
In [101]:
```python
def preditct_WKNN(x_test, x_train, y_train, k=3):
    y_predected = np.array([], dtype=int)
    for test_item in x_test:
        distances = [euclidean_distance(test_item, x_t)for x_t in x_train]
        sorted_indices = np.argsort(distances)[:k]
        k_weighted_labels = [(y_train[i], 1./distances[i]) for i in sorted_indices]
        y_predected = np.append(y_predected, weight_sum(k_weighted_labels))
    return y_predected
```

In [102]:
```python
x_test = np.array([[3,3],[1,4]])

x_train = np.array([
    [1,1],
    [5,3],
    [3,4],
    [5,2],
    [5,4],
    [1,4.5],
    [4,4],
    [1,3],
    [3,6],
    [4,6],
    [2.5,2]])

y_train = np.array([0,0,0,0,1,1,1,1,2,2,2])
#0 for red, 1 for blue, 2 for green
y_predicted = preditct_WKNN(x_test, x_train, y_train, k=5)
Colors = ["Red", "Blue", "Green"]
for i, result in enumerate(y_predicted):
    print("Point:(%0.1f,%0.1f)"%(x_test[i][0],x_test[i][1]), "Result: %s"%Colors[result])
```

```
Point:(3.0,3.0) Result: Red
Point:(1.0,4.0) Result: Blue
```

In [ ]:

In [ ]:



Weighted KNN:

k = 5

distance between the wanted point and

	distance	weight
point 1 :	0.2	5
point 2 :	0.5	2
point 3 :	0.7	1.4
point 4 :	1.2	0.8
point 5 :	1.5	0.6

we use function to give weights for each point (ex: $\frac{1}{\text{distance}}$)

weight of red points: 7

weigh of green points = 2.8

So the point to be classified will be red

But if it was normal KNN it would have been green (3 > 2) jupyter Weighted-KNN Last Checkpoint: 6 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```
#19290116
#MOHAMMAD SHABIB
import numpy as np
import pandas as pd
from sklearn import datasets #this import is used for using the iris dataset
from sklearn.model_selection import train_test_split #this import is used to split the data into training and test
```

In [2]:
```
#distance between two node
def euclidean_distance(x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))

#consider the weight of the node not just the popularity of the class
def weight_sum(k_weighted_labels):
    df = pd.DataFrame(k_weighted_labels, columns = ["label", "weight"])
    df = df.groupby(by=["label"]).sum()
    df.reset_index(inplace=True)
    df.sort_values(by=['weight'], ascending=False, inplace=True)
    return df["label"].iloc[0]
```

In [3]:
```
def preditct_WKNN(x_test, x_train, y_train, k=3):
    y_predected = np.array([], dtype=int)
    for test_item in x_test:
        distances = [euclidean_distance(test_item, x_t)for x_t in x_train]
        sorted_indices = np.argsort(distances)[:k]
        k_weighted_labels = [(y_train[i], 1/distances[i]) for i in sorted_indices]
        y_predected = np.append(y_predected, weight_sum(k_weighted_labels))
    return y_predected
```

In [4]:
```
def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy
```

In [5]:
```
iris = datasets.load_iris()
x, y = iris.data, iris.target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1234)

y_predicted = preditct_WKNN(x_test, x_train, y_train, k=3)
accuracy(y_test, y_predicted)
```

Out[5]: 1.0

In [ ]: