Our ultimate goal is to minimize the training error.

Another way to view this observation is that our ultimate goal, is to extract a hypothesis, which will return the minimum training error.

Perceptron Algorithm

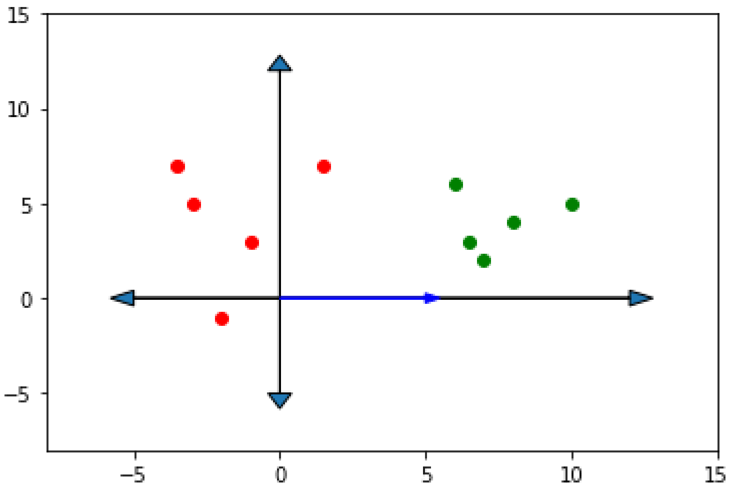
In this section, we are going to introduce the percepton algorithm. Perceptron algorithm, will generate a hypothesis function (linear classifier) for a given data set under consideration. From the explanation of the perceptron algorithm, we may receive a wealth of introductory needed concepts.

Introductory Example

Task: Given 2-d observation data points, return a hypothesis (using perceptron algorithm) that classifies them in the best possible way

In [3]:
```
# Package for scientific computing with Python
# https://numpy.org/
import numpy as np

# Matplotlib is a library for creating static, animated, and interactive visualizations in Python.
# https://matplotlib.org/
import matplotlib.pyplot as plt

# our horizontal and vertical graph axes
ax = plt.axes()

# x - axis positive:
# origin_vector_x, origin_vector_y, dx, dy
ax.arrow(0.0, 0.0, 12, 0.0, head_width=0.8, head_length=0.8)

# x - axis negative:
# origin_vector_x, origin_vector_y, dx, dy
ax.arrow(0.0, 0.0, -5.0, 0.0, head_width=0.8, head_length=0.8)

# y-axis positive
```

```
ax.arrow(0.0, 0.0, 0.0, 12.0, head_width=0.8, head_length=0.8)
#ax.text(-7, 4, "theta vector")

# y-axis negative
ax.arrow(0.0, 0.0, 0.0, -5.0, head_width=0.8, head_length=0.8)

x_no = np.array([-1, -2, -3, -3.5, 1.5])
y_no = np.array([3,  -1, 5, 7, 7])

x_yes = [7, 6, 6.5, 8, 10]
y_yes = [2, 6, 3, 4, 5]

# y axes of figure limits
plt.ylim(-8,15)

# x axes of figure limits
plt.xlim(-8,15)

plt.plot(x_no,y_no,'o', color='red')
plt.plot(x_yes,y_yes,'o', color='green')

ax.arrow(0.0,0.0, 5.0, 0.0, head_width=0.5, head_length=0.5, color="blue")

plt.show()
```



The horizontal axis (x) contains $x_1^{(i)}$ and the vertical axis (y) contains the $x_2^{(i)}$. With red color are depicted the data points labeled as -1, whereas with green color are depticted the data points labeled as +1.

Initialization

Based uppon the concepts of the previous introductory section, we are going to arbitrarily initialize a $\bar{\theta}$ vector as well as a θ_0 quantity as follows:

$$\bar{\theta} = (5.0, 0.0),$$
$$\theta_0 = 0.0$$

Additionally, we are going to initialize a constant value m, which will define our maximum number of iterations.

In [4]:
```
# maximum number of iterations (a.k.a. epochs)
m = 10
```

Data Set

In order for our algorithm to function, we have to pass a labeled data set. Therefore, we will give for each observation data point a label and then we are going to merge all data points into a general set.

In [5]:
```
# red points labeled as -1
x_no = np.array([-1, -2, -3, -3.5, 1.5])
y_no = np.array([3,  -1, 5, 7, 7])
label_no = np.array([-1, -1, -1, -1, -1])

# green points labeled as +1
x_yes = [7, 6, 6.5, 8, 10]
y_yes = [2, 6, 3, 4, 5]
label_yes = np.array([1, 1, 1, 1, 1])
```

In [6]:
```
# merge x-coordinates of all observations into 1 set
x = np.append(x_no, x_yes)
print(x)
```

```
[-1.  -2.  -3.  -3.5  1.5  7.   6.   6.5  8.   10. ]
```

In [7]:
```
# merge y-coordinates of all observations into 1 set
y = np.append(y_no, y_yes)
print(y)
```

```
[ 3 -1  5  7  7  2  6  3  4  5]
```

In [8]:
```
# merge all known facts (all labels) into 1 set
labels = np.append(label_no, label_yes)
print(labels)
```

```
[-1 -1 -1 -1 -1  1  1  1  1  1]
```

In [9]:
```
# formally initialize theta vector and theta_0 vector
theta = [5.0, 0.0]
theta_0 = 0.0
```

Perceptron Algorithm

What follows, is the perceptron algorithm (in pseudo code), the explanation and some insight analysis of its functionality as well as a python - code representation alongside the intermediate steps display of the functionality.

Pseudo - Code

// iterate for maximum number of defined iterations (epochs)

for t =1 through m

theta_change = False

// iterate through all observation data points

for i=1 through n

// if statement becomes true if we have made a false observation

if($y^{(i)} \cdot (\overline{\theta^T} \cdot \overline{x^{(i)}} + \theta_0) \leq 0$)

set $\overline{\theta}$ $+= y^{(i)} \cdot \overline{x^{(i)}}$

set θ_0 $+= y^{(i)}$

theta_change = True

if(theta_change == False)
break
return $\overline{\theta}, \theta_0$

Analysis

There are 2 primary points of interest worth noting in our perceptron algorithm:

1. The first is, when does our perceptron algorithm changes or is being modified.
   We can see that perceptron algorithm is being modified whenever this condition becomes true

$$y^{(i)} \cdot (\overline{\theta^T} \cdot \overline{x^{(i)}} + \theta_0) \leq 0$$

We remember that $y^{(i)}$ is our label (or established fact), and can receive either a positive number +1, or a negative number -1.
Furthemore, we remember that $\overline{\theta^T} \cdot \overline{x^{(i)}} + \theta_0$ is our prediction hypothesis for a given observation data point $\overline{x^{(i)}}$.

Given that our prediction follows the same logic as our labels, classifying either +1 or -1, the prementioned condition becomes true iff : our prediction and the established known fact do not match with each other.

1. The second point of interest worth noting, is how the perceptron is being modified when hypothesis result and label do not match with each other. We see that 2 main modifications take place:

$$\overline{\theta}_{new} = \overline{\theta}_{old} + (y^{(i)} \cdot \overline{x^{(i)}})$$

as well as

$$\theta_{0_{new}} = \theta_{0_{old}} + y^{(i)}$$

Let us review how the condition of (1) is being modified with the updated values $\overline{\theta}_{new}$ and $\theta_{0_{new}}$. The condition now becomes:

$$y^{(i)} \cdot (\overline{\theta_{new}^T} \cdot \overline{x^{(i)}} + \theta_{0_{new}}) \Rightarrow$$
$$y^{(i)} \cdot [[\overline{\theta}_{old} + (y^{(i)} \cdot \overline{x^{(i)}})]^T \cdot \overline{x^{(i)}} + (\theta_{0_{old}} + y^{(i)})] \Rightarrow$$
$$y^{(i)} \cdot [(\overline{\theta_{old}^T} \cdot \overline{x^{(i)}}) + (y^{(i)} \cdot \overline{x^{(i)T}} \cdot \overline{x^{(i)}}) + \theta_{0_{old}} + y^{(i)}] \Rightarrow$$
$$y^{(i)} \cdot \overline{\theta_{old}^T} \cdot \overline{x^{(i)}} + y^{(i)^2} \cdot \overline{x^{(i)^T}} \cdot \overline{x^{(i)}} + y^{(i)} \cdot \theta_{0_{old}} + y^{(i)^2} \Rightarrow$$
$$y^{(i)} \cdot (\overline{\theta_{old}^T} \cdot \overline{x^{(i)}} + \theta_{0_{old}}) + y^{(i)^2} \cdot (\overline{x^{(i)^T}} \cdot \overline{x^{(i)}} + 1) \Rightarrow$$
$$y^{(i)} \cdot hypothesis_{old}(\overline{x^{(i)}}) + 1 \cdot (length^2_{x^{(i)}} + 1) \Rightarrow$$
$$y^{(i)} \cdot hypothesis_{old}(\overline{x^{(i)}}) + (\|x^{(i)}\|^2 + 1)$$

What all the prementioned statements actually indicate, is that for every observation data point ($x^{(i)}$) that has been wrongfully classified ($y^{(i)} \cdot hypothesis_{old}(\overline{x^{(i)}}) \leq 0$), we add a strictly positive quantity based uppon $x^{(i)}$, to minimize the distance of $y^{(i)} \cdot hypothesis_{old}(\overline{x^{(i)}})$ from 0, untill it becomes a positive value $\forall \overline{x^{(i)}}$

Next we are going to present the execution of perceptron algorithm (in python) presented in the previously displayed data - set.

Python Code

First we are going to define a function, **plotPerceptronResult()**, which will generate the graphical representation of the θ_{vector}, the perpendicular to $\overline{\theta}$ vector ($hypothesis(\overline{v}) = 0$) alongside all of our observation data points.

In [10]:
```
def plotPerceptronResults(theta, x_i_1, x_i_2):
    # 1. Our horizontal and vertical graph axes
    ax = plt.axes()

    # 2. Cartesian axes vectors
        # x - axis positive:
        # origin_vector_x, origin_vector_y, dx, dy
    ax.arrow(0.0, 0.0, 12, 0.0, head_width=0.8, head_length=0.8)
        # x - axis negative:
    ax.arrow(0.0, 0.0, -5.0, 0.0, head_width=0.8, head_length=0.8)
        # y-axis positive
```

```
    ax.arrow(0.0, 0.0, 0.0, 12.0, head_width=0.8, head_length=0.8)
        # y-axis negative
    ax.arrow(0.0, 0.0, 0.0, -5.0, head_width=0.8, head_length=0.8)

    # 3. Data
        # -1 labeled red point observations
    x_no = np.array([-1, -2, -3, -3.5, 1.5])
    y_no = np.array([3,  -1, 5, 7, 7])
        # +1 labeled red point observations
    x_yes = [7, 6, 6.5, 8, 10]
    y_yes = [2, 6, 3, 4, 5]
        # data plot
    plt.plot(x_no,y_no,'o', color='red')
    plt.plot(x_yes,y_yes,'o', color='green')

    # 4. Figure limits
        # y axes limits of figure
    plt.ylim(-8,15)
        # x axes of figure limits
    plt.xlim(-8,15)

    # 5. Vectors
        # theta vector
    ax.arrow(0.0,0.0, theta[0], theta[1], head_width=0.5, head_length=0.5, color="blue")
        # perpendicular vector [x1, x2] with x1=5
        # for theta = [c1, c2]^T
        # c1*x1 + c2*x2 = 0 =>
        # c1*5 + c2*x2 = 0 =>
        # c2*x2 = -c1*5
        # x2 = (-c1/c2) * 5
        # plot from (0,0) to (5, (-c1/c2) * 5)
    ax.arrow(0.0,0.0, 5.0, (-theta[0]/theta[1])*5, head_width=0.5, head_length=0.5, color="red")
        #symmetric
        # plot from (0,0) to (-5, (c1/c2) * 5)
    ax.arrow(0.0,0.0, -5.0, (theta[0]/theta[1])*5, head_width=0.5, head_length=0.5, color="red")

    # 6. Point under consideration
    plt.plot(x_i_1, x_i_2, 'v', color="black")

    plt.show()
```

Next, we are defining a function, **dotProduct()**, which performs vector multiplication between a given $\overline{\theta}$ and a given $\overline{x^{(i)}}$, where $\overline{\theta}, \overline{x^{(i)}} \in R^2$

In [11]:

```python
# function that receives two vectors that can
# be multiplied and returns their dot product
def dotProduct(theta_vector, data_point_x, data_point_y):
    result = 0
    result += theta_vector[0]*data_point_x
    result += theta_vector[1]*data_point_y
    return result
```

Finally we present, the perceptron algorithm:

In [12]:

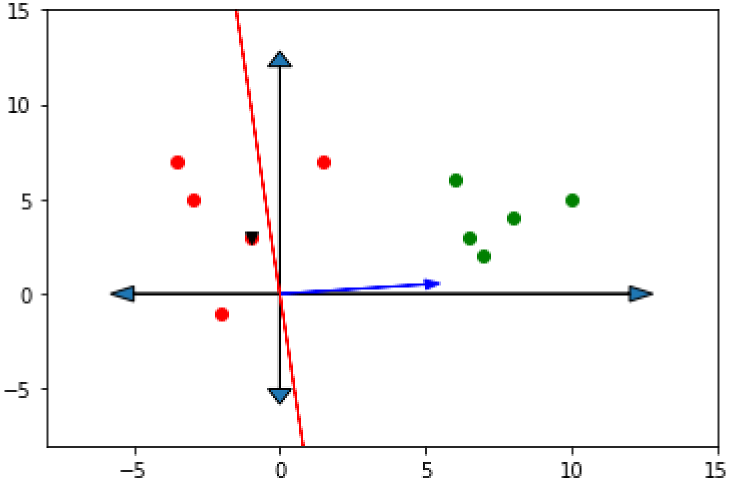
```python
# formally initialize theta vector and theta_0 vector
theta = [5.0, 0.5]
theta_0 = 0.0
```

In [13]:

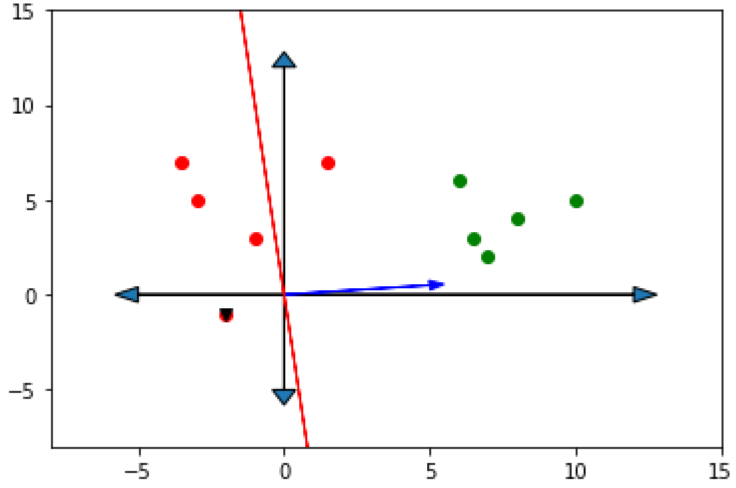
```python
# iterate from 1 to m
for t in range(1, m+1):

    print("Iteration:"+str(t))

    # have we observed a change in theta vector
    is_theta_changed = False

    #iterate through data points
    for i in range(len(x)):
        prediction = labels[i] * dotProduct(theta, x[i], y[i]) + labels[i] * theta_0

        print("Observation data point:["+str(x[i])+","+str(y[i])+"]")
        print("Label"+str(labels[i]))

        print("Theta:["+str(theta[0])+","+str(theta[1])+"]")
        print("Theta_0:"+str(theta_0))

        print("Prediction:"+str(prediction))

        plotPerceptronResults(theta, x[i], y[i])

        if(prediction<=0):
            #our hypothesis prediction is wrong and has to change
            print("Wrong prediction")
            theta[0] += labels[i]*x[i]
            theta[1] += labels[i]*y[i]
            theta_0 += labels[i]
```

```
            print("New Theta:["+str(theta[0])+","+str(theta[1])+"]")
            print("New theta_0:"+str(theta_0))

            is_theta_changed = True
        else:
            print("Right prediction")
        print("----------------\n")

    if(is_theta_changed==False):
        break
    print("============")
```

```
Iteration:1
Observation data point:[-1.0,3]
Label-1
Theta:[5.0,0.5]
Theta_0:0.0
Prediction:3.5
```

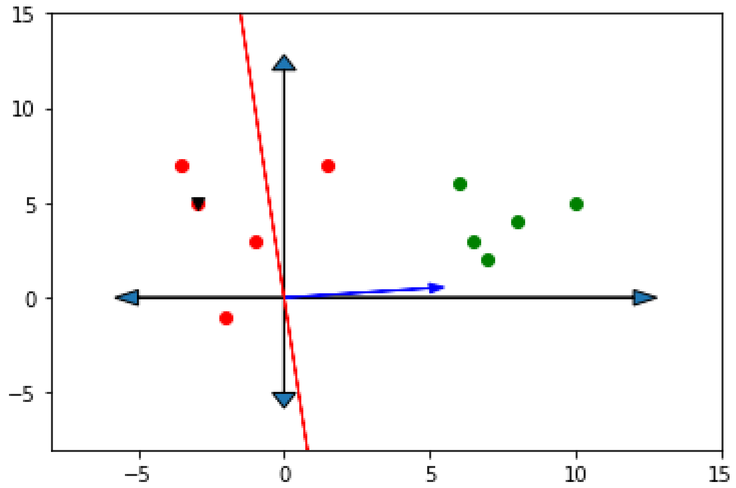



```
Right prediction
----------------

Observation data point:[-2.0,-1]
Label-1
Theta:[5.0,0.5]
Theta_0:0.0
Prediction:10.5
```

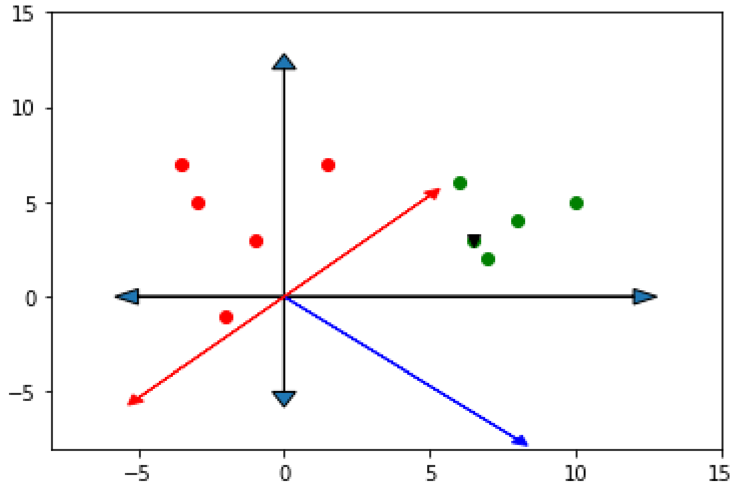


Right prediction

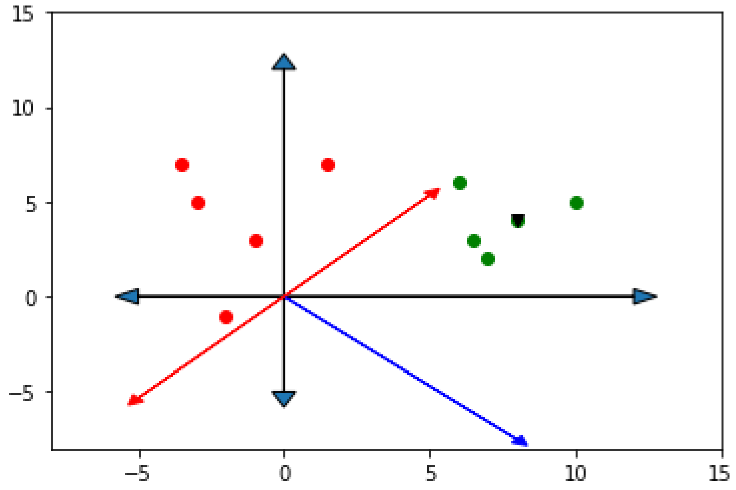
Observation data point:[-3.0,5]
Label-1
Theta:[5.0,0.5]
Theta_0:0.0
Prediction:12.5



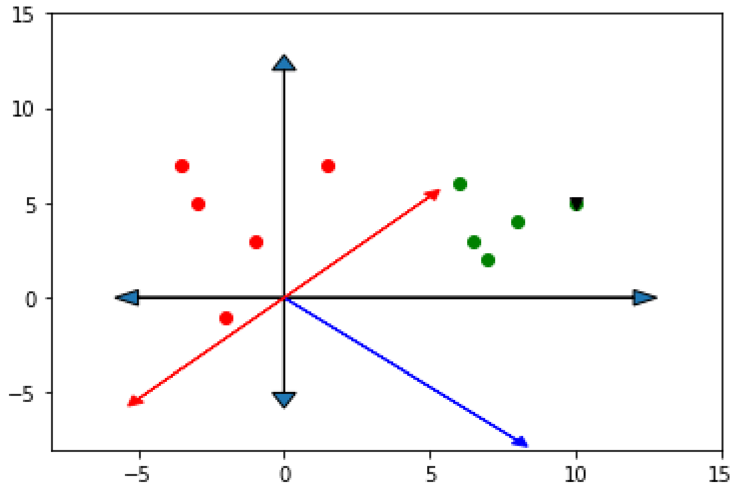
```
Right prediction
----------------

Observation data point:[6.5,3]
Label1
Theta:[8.0,-7.5]
Theta_0:-1.0
Prediction:28.5
```




```
Right prediction
----------------

Observation data point:[8.0,4]
Label1
Theta:[8.0,-7.5]
Theta_0:-1.0
Prediction:33.0
```




```
Right prediction
----------------

Observation data point:[10.0,5]
Label1
Theta:[8.0,-7.5]
Theta_0:-1.0
Prediction:41.5
```




```
Right prediction
----------------
```

It is easy to observe, that if our data set of observations is linearly classified through the origin, then the perceptron algorithm can be executed as it is, with no adjustments or modifications. Nevertheless, we can be certain that it is extremely likely to face an example of a data - set of observations, that is linearly classifiable, nevertheless it is not linearly classifiable through the origin. Let us review an alike example as well as the proper modifications that we have to make.

Basketball Example

Let us assume, that we are given as a task to generate a hypothesis based uppon the following:

Task:
We want to extract a prediction on whether or not, a high - scholl student of a given high - school, will be picked/drafted to take part in the High - School Basketball team for the current championship season, based uppon students height (in cms) and weight (in kg). We are given an observation data set, of (height, weight) vectors of students that were succesfully picked as well as not picked, during the last 3 years.

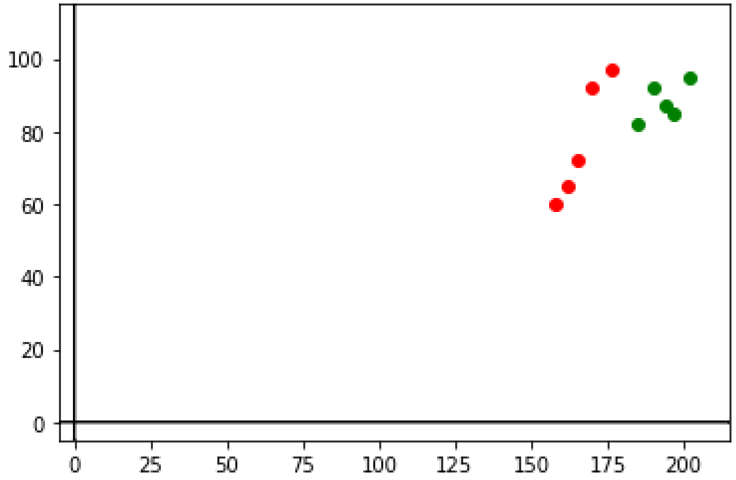
Data Set:

#Student Observation $x^{(i)}$	**Height (cm-s)**	**Weight (kg-s)**	**Chosen (label)**
$x^{(0)}$	158	60	-1
$x^{(1)}$	165	72	-1
$x^{(2)}$	170	92	-1
$x^{(3)}$	162	65	-1
$x^{(4)}$	176	97	-1
$x^{(5)}$	185	82	+1
$x^{(6)}$	194	87	+1
$x^{(7)}$	190	92	+1
$x^{(8)}$	197	85	+1
$x^{(9)}$	202	95	+1

Data Plot:

In [14]:

```
# 1. Our horizontal and vertical graph axes
ax = plt.axes()

# 2. Cartesian axes vectors
    # x - axis positive:
    # origin_vector_x, origin_vector_y, dx, dy
ax.arrow(0, 0, 215.0, 0.0, head_width=0.8, head_length=0.8)
    # x - axis negative:
ax.arrow(0.0, 0.0, -5.0, 0.0, head_width=0.8, head_length=0.8)
    # y-axis positive
ax.arrow(0.0, 0.0, 0.0, 115.0, head_width=0.8, head_length=0.8)
    # y-axis negative
ax.arrow(0.0, 0.0, 0.0, -5.0, head_width=0.8, head_length=0.8)

# 3. Data
    # -1 labeled red point observations
x_no = np.array([158, 165, 170, 162, 176])
y_no = np.array([60, 72, 92, 65, 97])
    # +1 labeled red point observations
x_yes = [185, 194, 190, 197, 202]
y_yes = [82, 87, 92, 85, 95]
    # data plot
plt.plot(x_no,y_no,'o', color='red')
plt.plot(x_yes,y_yes,'o', color='green')

# 4. Figure limits
    # y axes limits of figure
plt.ylim(-5,115)
    # x axes of figure limits
plt.xlim(-5,215)

plt.show()
```



We can see by the prementioned plot, that our data set is linearly classified, nevertheless is not linearly classified through the origin.

Linear classification through the origin

We can transform the existing data - set to be linearly classifiable through the origin, by performing a very simple dimensionanility transformation. More specifically, for:

$$\overline{\theta} = (\theta_1, \theta_2)$$

and

$$\overline{x^{(i)}} = (x_1^{(i)}, x_2^{(i)})$$

we can increase the dimensions by +1 as follows:

$$\overline{\theta} = (\theta_1, \theta_2, \theta_0)$$
$$\overline{x^{(i)}} = (x_1^{(i)}, x_2^{(i)}, 1)$$

In order for our human perception to understand more easily the way the algorithm functions via this transformation (also known as bias insertion) we can visualize the data points in R^2, where we have to expect that $\forall x^{(i)}$, the data points will be succesfully classified via the equation:

$$\theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)} + \theta_0 \cdot 1 = 0$$

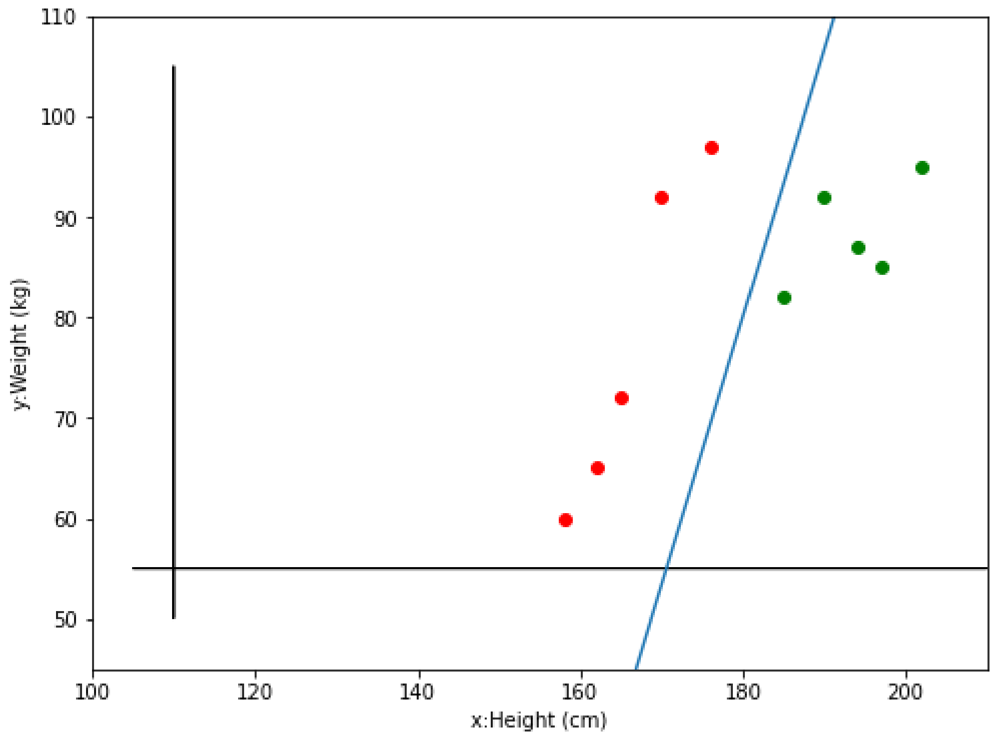
Let us review the given data set under this point of view, given a θ that correctly classifies the observation data set:

$$\bar{\theta} = (40, -15, -6000)$$

In [15]:
```
# Package for scientific computing with Python
# https://numpy.org/
import numpy as np

# Matplotlib is a library for creating static, animated, and interactive visualizations in Python.
# https://matplotlib.org/
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

In [16]:
```
# -1 labeled red point observations
x_no = np.array([158, 165, 170, 162, 176])
y_no = np.array([60,  72,  92,  65,  97])
z_no = np.array([1, 1, 1, 1, 1])
    # +1 labeled red point observations
x_yes = [185, 194, 190, 197, 202]
y_yes = [82, 87, 92, 85, 95]
z_yes = [1, 1, 1, 1, 1]

theta = [40, -15, -6000]

fig = plt.figure(figsize=(8,6))
#============================= PLOT XY ====================================
# 1. Add subplot (rows, columns, index)
ax = fig.add_subplot(111)

ax.set_xlim([100,210])
# height
ax.set_ylim([45,110])

# generate x axis
    #positive
ax.arrow(110, 55, 100, 0.0, color="black")
    #negative
ax.arrow(110, 55, -5, 0.0, color="black")
```

```
# generate y axis
    #positive
ax.arrow(110, 55, 0.0, 50, color="black")
    #negative
ax.arrow(110, 55, 0.0, -5, color="black")

# 2. Plot all the observation data points
ax.plot(x_no, y_no, 'o', color='red')
ax.plot(x_yes, y_yes,'o', color='green')

ax.set_xlabel("x:Height (cm)")
ax.set_ylabel("y:Weight (kg)")
#==========================================================================
#============== GENERATE 2 POINTS FOR THE CLASSIFIER LINE =================
# equation:
# x2 = -( x1*theta1 + 1*theta0 ) / theta2
# y = -( x1*theta[0] + 1*theta[2] ) / theta[1]
# for x= [120, 210]
x = [-120,210]
y = [0,0]
y[0] = -( x[0]*theta[0] + 1*theta[2] ) / theta[1]
y[1] = -( x[1]*theta[0] + 1*theta[2] ) / theta[1]
ax.plot(x,y)
#==========================================================================
plt.show()
```



It would be interesting to review the verification of the correct theta of our perceptron algorithm, by reviewing its 3D representation alongside.

$$\bar{\theta} = (40, -15, -6000)$$

In [17]:
```
def plotPerceptronResults(theta, x_i_x, x_i_y, x_i_z):
    # -1 labeled red point observations
    x_no = np.array([158, 165, 170, 162, 176])
    y_no = np.array([60,  72,  92,  65,  97])
    z_no = np.array([1, 1, 1, 1, 1])
        # +1 labeled red point observations
    x_yes = [185, 194, 190, 197, 202]
    y_yes = [82, 87, 92, 85, 95]
    z_yes = [1, 1, 1, 1, 1]

    fig = plt.figure(figsize=(20,8))
    #============================= PLOT XY ====================================
    # 1. Add subplot (rows, columns, index)
    ax = fig.add_subplot(121)
```

```
#ax.set_xlim([100,210])
# height
#ax.set_ylim([45,110])

# generate x axis
    #positive
ax.arrow(0, 0, 210, 0.0, color="black")
    #negative
ax.arrow(0, 0, -5, 0.0, color="black")

# generate y axis
    #positive
ax.arrow(0, 0, 0.0, 110, color="black")
    #negative
ax.arrow(0, 0, 0.0, -5, color="black")

# 2. Plot all the observation data points
ax.plot(x_no, y_no, 'o', color='red')
ax.plot(x_yes, y_yes,'o', color='green')
ax.plot(x_i_x, x_i_y, 'v', color='black')

ax.set_xlabel("x:Height (cm)")
ax.set_ylabel("y:Weight (kg)")
#===========================================================================
#============== GENERATE 2 POINTS FOR THE CLASSIFIER LINE =================
# equation:
# x2 = -( x1*theta1 + 1*theta0 ) / theta2
# y = -( x1*theta[0] + 1*theta[2] ) / theta[1]
# for x= [120, 210]
x = [-12,210]
y = [0,0]
y[0] = -( x[0]*theta[0] + 1*theta[2] ) / theta[1]
y[1] = -( x[1]*theta[0] + 1*theta[2] ) / theta[1]
ax.plot(x,y)
#===========================================================================
#============================ PLOT 3D ====================================
# add subplot (3d projection of rows, columns, index)
ax = fig.add_subplot(122, projection='3d')
#ax.set_xlim([100,210])
#ax.set_ylim([45,110])
ax.set_zlim([-0.5,1.2])
# generate x axis
    #positive
```

```
ax.quiver(0.0, 0.0, 0.0, 220.0, 0.0, 0.0, arrow_length_ratio=0.001, alpha=0.5, color="black")
    #negative
ax.quiver(0.0, 0.0, 0.0, -220.0, 0.0, 0.0, arrow_length_ratio=0.001, alpha=0.5, color="black")
# generate x plane
#x_x=[-250,250, 250, -250]
#x_y=[0, 0, 0, 0]
#x_z=[-5.0,-5.0, 5.0, 5.0]
#vertices = [list(zip(x_x,x_y,x_z))]
#poly = Poly3DCollection(vertices, alpha=0.1, color="black")
#ax.add_collection3d(poly)

#---------------------
# generate y axis
    #positive
ax.quiver(0.0, 0.0, 0, 0.0, 120.0, 0.0, arrow_length_ratio=0.001, alpha=0.5, color="black")
    #negative
ax.quiver(0.0, 0.0, 0, 0.0, -120.0, 0.0, arrow_length_ratio=0.001, alpha=0.5, color="black")
# generate y plane
y_x=[-250,250, 250, -250]
y_y=[-130, -130, 130, 130]
y_z=[0,0, 0, 0]
vertices = [list(zip(y_x,y_y,y_z))]
poly = Poly3DCollection(vertices, alpha=0.1, color="black")
ax.add_collection3d(poly)

#---------------------
# generate z axis
    #positive
ax.quiver(0.0, 0.0, 0, 0.0, 0.0, 5.0, arrow_length_ratio=0.1, alpha = 0.5, color="black")
    #negative
ax.quiver(0.0, 0.0, 0, 0.0, 0.0, -5.0, arrow_length_ratio=0.1, alpha = 0.5, color="black")
# generate z plane
#z_x=[0,0, 0, 0]
#z_y=[-130, 130, 130, -130]
#z_z=[-5.0,-5, 5.0, 5.0]
#vertices = [list(zip(z_x,z_y,z_z))]
#poly = Poly3DCollection(vertices, alpha=0.1, color="black")
#ax.add_collection3d(poly)

ax.scatter(x_no,y_no,z_no,'o', color='red')
ax.scatter(x_yes,y_yes,z_yes,'o', color='green')
ax.scatter(x_i_x, x_i_y, x_i_z,'v', color='black')

ax.set_xlabel("x:Height (cm)")
```

```
    ax.set_ylabel("y:Weight (kg)")
    ax.set_zlabel("z:Constant 1")

    ax.quiver(0.0, 0.0, 0.0, theta[0], theta[1], theta[2], arrow_length_ratio=0.001, color="blue")
    perp = [190, 90, 0]
    perp[2] = -(perp[0]*theta[0]+perp[1]*theta[1])/theta[2]
    ax.quiver(0.0, 0.0, 0.0, theta[0], theta[1], theta[2], arrow_length_ratio=0.001, color="blue")
    ax.quiver(0.0, 0.0, 0.0, perp[0], perp[1], perp[2], arrow_length_ratio=0.0001, color="red")
    ax.view_init(15,45)
    #===========================================================================

    plt.show()
```

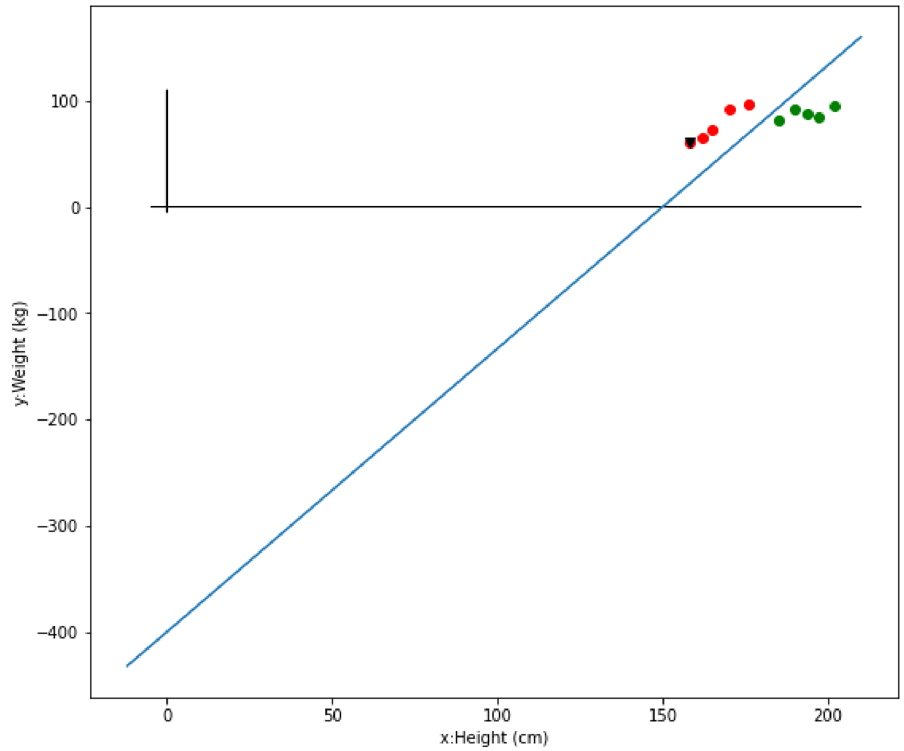
In [18]:
```
# function that receives two vectors that can
# be multiplied and returns their dot product
def dotProduct(theta_vector, data_point_x, data_point_y, data_point_z):
    result = 0
    result += theta_vector[0]*data_point_x
    result += theta_vector[1]*data_point_y
    result += theta_vector[2]*data_point_z
    return result
```

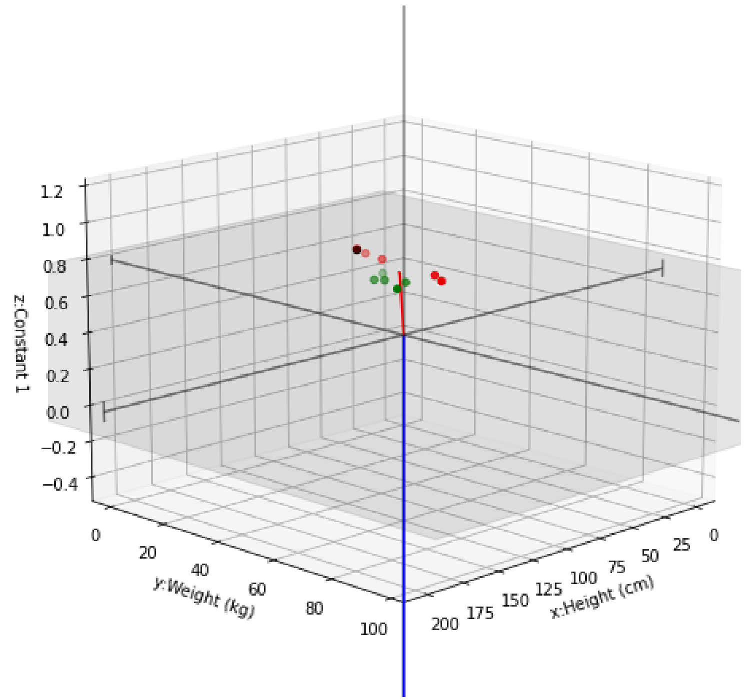
In [19]:
```
# formally initialize theta vector and theta_0 vector
theta = [40,-15,-6000]
m =1
```

In [20]:
```
# -1 labeled red point observations
x_no = np.array([158, 165, 170, 162, 176])
y_no = np.array([60, 72, 92, 65, 97])
z_no = np.array([1, 1, 1, 1, 1])
l_no = np.array([-1, -1, -1, -1, -1])
# +1 labeled red point observations
x_yes = [185, 194, 190, 197, 202]
y_yes = [82, 87, 92, 85, 95]
z_yes = [1, 1, 1, 1, 1]
l_yes = np.array([1, 1, 1, 1, 1])

# merge x-coordinates of all observations into 1 set
x = np.append(x_no, x_yes)
# merge y-coordinates of all observations into 1 set
y = np.append(y_no, y_yes)
# merge z-coordinates of all observations into 1 set
```

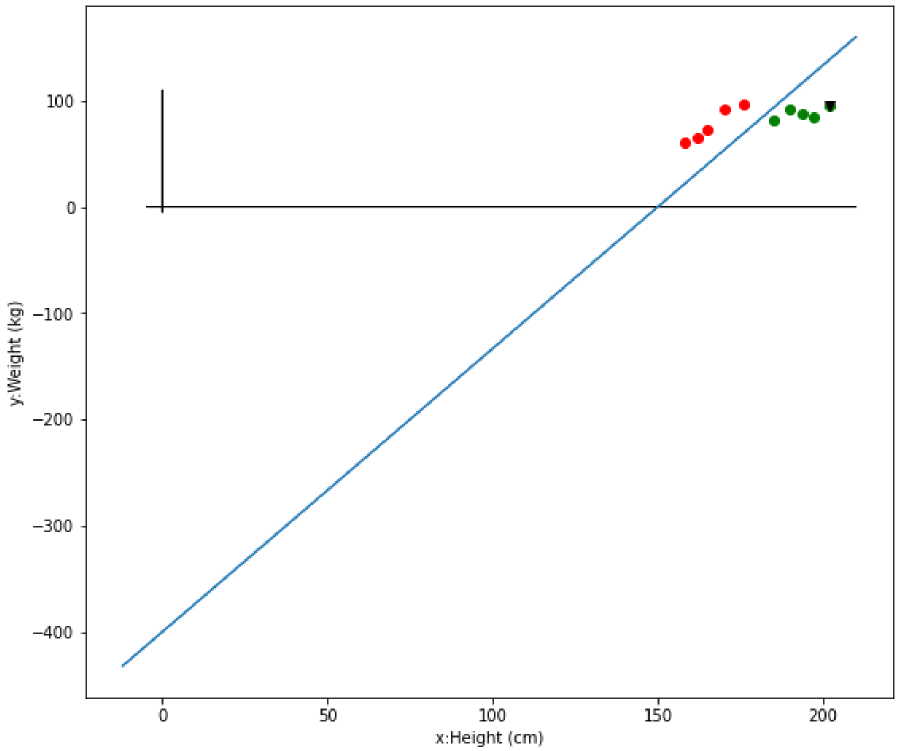
```python
z = np.append(z_no, z_yes)
# merge z-coordinates of all observations into 1 set
labels = np.append(l_no, l_yes)


# iterate from 1 to m
for t in range(1, m+1):
    # have we observed a change in theta vector
    is_theta_changed = False
    #iterate through data points
    for i in range(len(x)):
        print("\tObservation data point:["+str(x[i])+","+str(y[i])+","+str(z[i])+"]")
        print("\tLabel"+str(labels[i]))
        print("\tPrediction:"+str(prediction))
        print("\tFrom Theta:"+str(theta))
        plotPerceptronResults(theta, x[i], y[i], z[i])
        prediction = labels[i] * dotProduct(theta, x[i], y[i], z[i])

        if(prediction<=0):
            print("\tWrong Prediction")
            print("\tFrom Theta:"+str(theta))
            #our hypothesis prediction is wrong and has to change
            theta[0] += labels[i]*x[i]
            theta[1] += labels[i]*y[i]
            theta[2] += labels[i]*z[i]
            is_theta_changed = True
            print("\tTo Theta:"+str(theta))
            print("--------------------------------------")
    if(is_theta_changed==False):
        print("Finished at:"+str(t))
        print("Theta:"+str(theta))
        plotPerceptronResults(theta, 0, 0, 0)
        break
    #print("==============================================================")
```

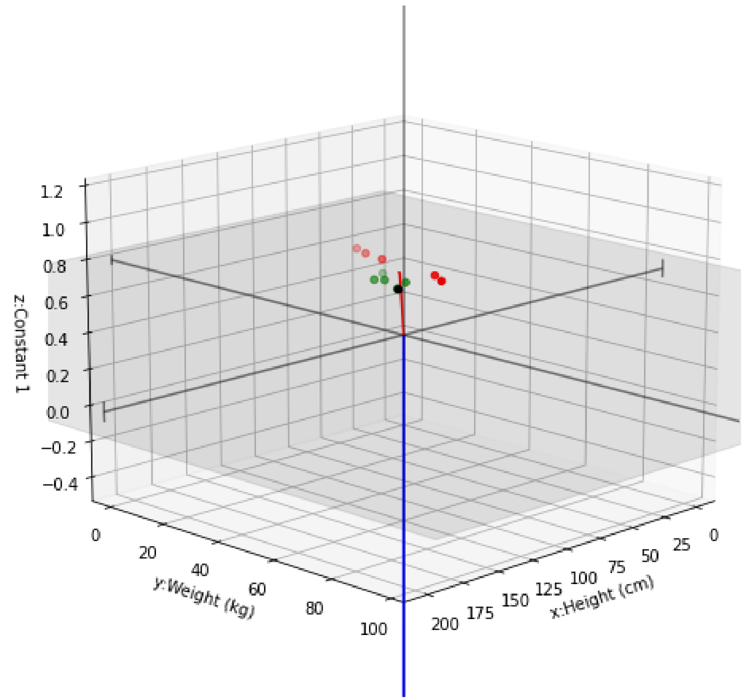
```
	Observation data point:[158,60,1]
	Label-1
	Prediction:41.5
	From Theta:[40, -15, -6000]
```



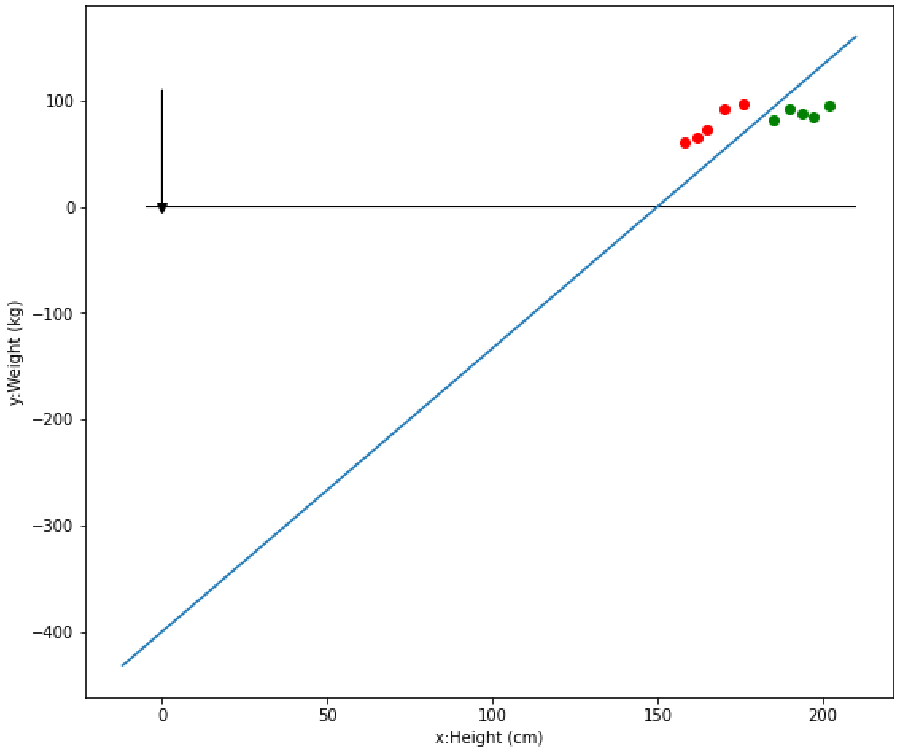


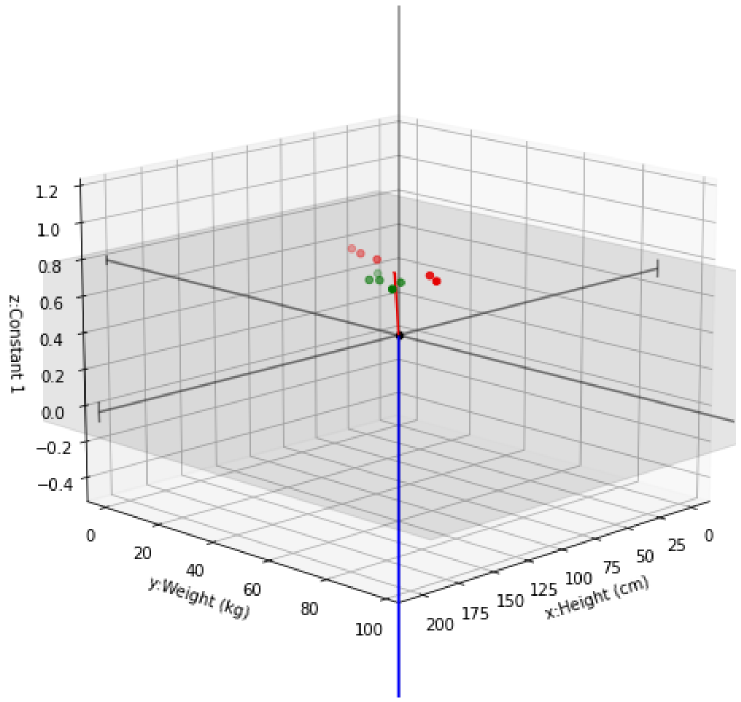

```
Observation data point:[165,72,1]
Label-1
Prediction:580
From Theta:[40, -15, -6000]
```






```
Finished at:1
Theta:[40, -15, -6000]
```





As we can see, for the same equation:

$$\theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)} + \theta_0 \cdot x_3^{(i)} = 0$$

in the left hand size, we have the classification representation of the hypothesis that every $x_2^{(i)}$ (or y or $f(x_1^{(i)})$) must satisfy the function:

$$hypothesis: f(x_1^{(i)}) = -\frac{(\theta_1 \cdot x_1^{(i)} + x_3 \cdot \theta_0)}{\theta_2}$$

whereas in the right hand side, we have the classification representation of the hypothesis, that must be perpendicular to θ, meaning that for every given x, y:

$$hypothesis = \overline{h} = (x, y, -\frac{(\theta_1 \cdot x + \theta_2 \cdot y)}{x_3 \cdot \theta_0})$$

where $\overline{h}$ passes through the origin.

Given that our perceptron algorithm is ensured to work whenever the data set is linearly classified and the classifier passes through the origin, the hypothesis vector solution in R^3 is exactly equivelant to the satisfaction of the hypothesis function in R^2.

An additional natural interpretation that we can attribute to the hypothesis function of R^2, is that of the weighted sum, where of a given classification problem, we attribute an importance to each parameter. Namely, we can view $\overline{\theta}$ as $\overline{w}$ vector, where:

$$\theta_1 = w_1 = importanceOf(x_1)$$
$$\theta_1 = w_1 = importanceOf(x_2)$$
$$\theta_0 = w_1 = bias$$