In [1]:
```
#Fall 2021
#ML 633: Machine Learning
#Homework: 1
#Date: 22-09-2021
#Author: Adrita Anika
```

Question 1

Part a: Data Exploration

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [3]:
```
#import libraries and train data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv('/content/drive/MyDrive/ML HW/data_train.csv', header=None)
```

In [4]:
```
#view data
data.head()
```

Out[4]:

	0	1	2	3
0	38	66	0	1
1	38	66	11	1
2	38	60	1	1
3	38	67	5	1
4	39	66	0	2

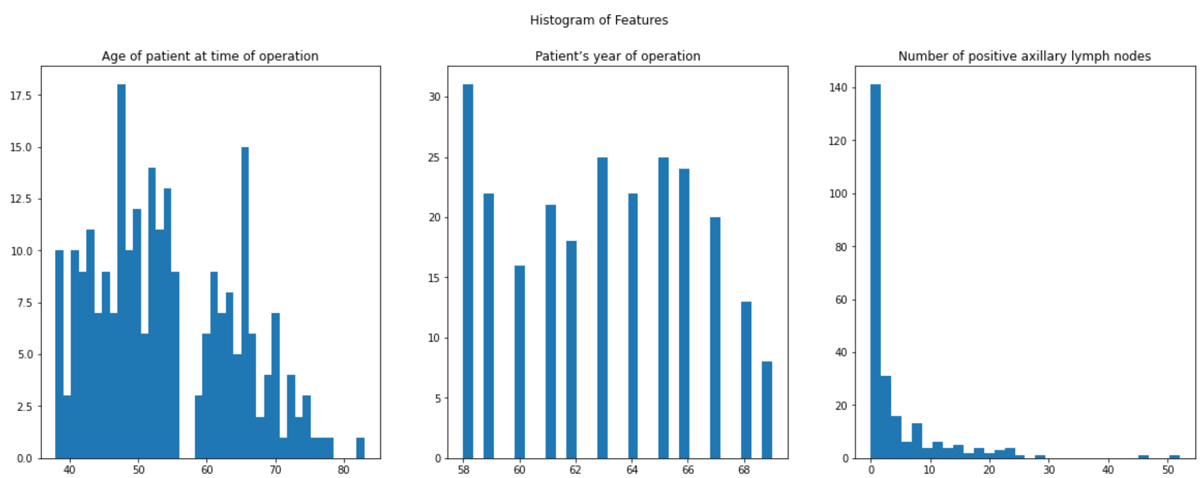
In [5]:
```
#a.i : compute the number of samples belonging to each class
no_of_samples = data.loc[:,3].value_counts()

print(f"The number of samples for class- 1 is {no_of_samples[1]} ")
print(f"The number of samples for class- 2 is {no_of_samples[2]}")
```

```
The number of samples for class- 1 is 173
The number of samples for class- 2 is 72
```

Answer to a.i : The classes are not equally distributed

In [6]:
```
#a.ii : plot the histogram of eachfeature (i.e.,  3 total histograms).

fig, axs = plt.subplots(1,3, figsize =(20, 7))
fig.suptitle('Histogram of Features')

feature_1 = axs[0].hist(data.loc[:,0], bins= 40)
axs[0].set_title("Age of patient at time of operation")
feature_2 = axs[1].hist(data.loc[:,1], bins= 30)
axs[1].set_title('Patient’s year of operation')
feature_3 = axs[2].hist(data.loc[:,2], bins= 30)
axs[2].set_title('Number of positive axillary lymph nodes')
```

Out[6]:
```
Text(0.5, 1.0, 'Number of positive axillary lymph nodes')
```

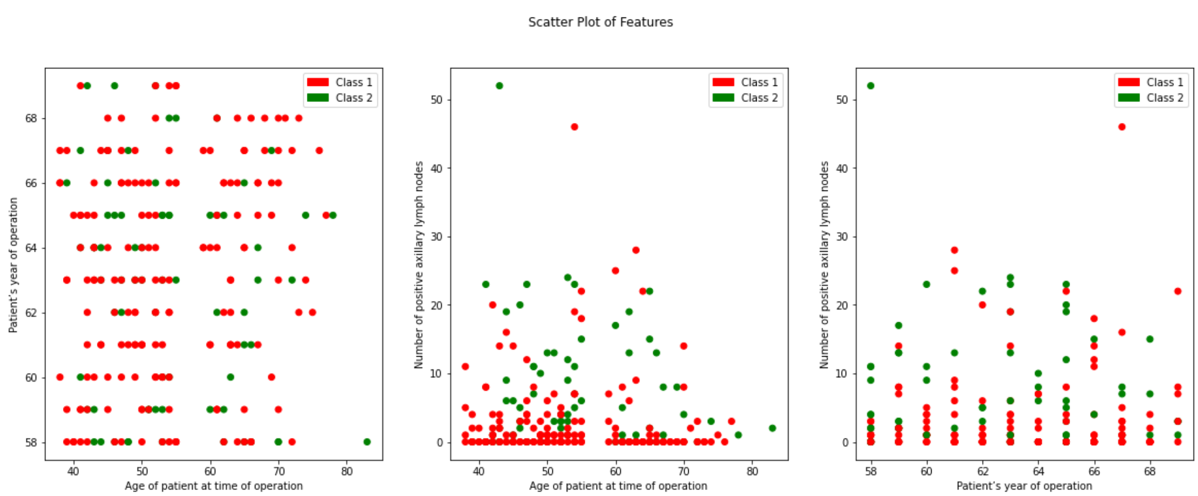


Answer to question a.ii.

1. Age of patient at time of operation has biimodal distrubution
2. Patient’s year of operation has uniform distribution
3. Number of positive axillary lymph nodes detected has unimodal distribution

In [7]:
```
#a.iii
'''
data_num = data.loc[:,3].to_numpy
idx_1 = np.where(data.loc[:,3]==1)
idx_2 = np.where(data.loc[:,3] ==2)
'''
import matplotlib.patches as mpatches
fig, ax = plt.subplots(1,3, figsize= (20,7))
fig.suptitle('Scatter Plot of Features')
colors = {1: 'red', 2: 'green'}

ax[0].scatter(data.loc[:,0], data.loc[:, 1], c = data.loc[:,3].map(colors))
pop_a = mpatches.Patch(color='red', label='Class 1')
pop_b = mpatches.Patch(color='green', label='Class 2')
ax[0].legend(handles=[pop_a,pop_b])
#ax[0].set_title('damped')
ax[0].set_xlabel('Age of patient at time of operation')
ax[0].set_ylabel('Patient’s year of operation')

ax[1].scatter(data.loc[:,0], data.loc[:, 2], c = data.loc[:,3].map(colors))
ax[1].legend(handles=[pop_a,pop_b])
ax[1].set_xlabel('Age of patient at time of operation')
ax[1].set_ylabel('Number of positive axillary lymph nodes')

ax[2].scatter(data.loc[:,1], data.loc[:, 2], c = data.loc[:,3].map(colors))
ax[2].legend(handles=[pop_a,pop_b])
ax[2].set_xlabel('Patient’s year of operation')
ax[2].set_ylabel('Number of positive axillary lymph nodes')
```

Out[7]: `Text(0, 0.5, 'Number of positive axillary lymph nodes')`



Answer to 1a.iii : From the above three graphs, it is observed that the two classes are not seperable. None of these feature combinations make the classes separable

Question 1

Part b: KNN classifier

Following is the KNN implementation for 1b.i

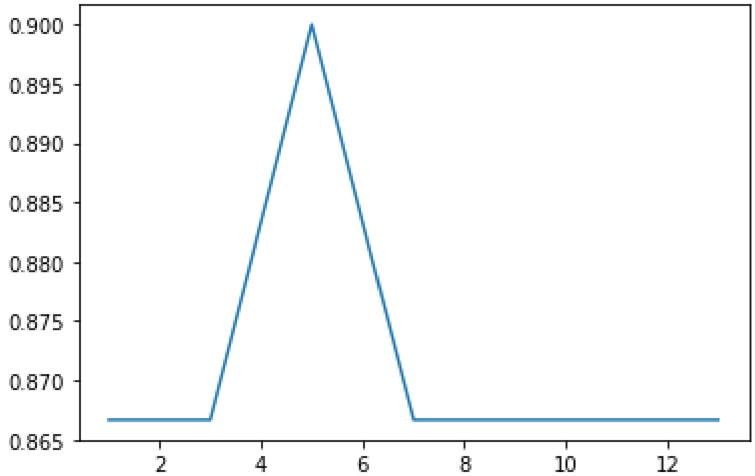
In [8]:
```
#1b.i
#Implementing KNN

class KNN_classifier:

  def __init__(self, K, distance = 'l2'):
    self.K = K
    self.distance = distance

  def fit(self, X_train, Y_train):
    self.X_train = X_train
    self.Y_train = Y_train

  def predict(self, test_sample):
    train_data_np = self.X_train.to_numpy()

    if self.distance == 'l1':
      distance_np = np.sum(abs(train_data_np - test_sample), axis = 1)
    else:
      distance_np = np.sum((train_data_np - test_sample)**2, axis = 1)
    #distance_np = eucledean_distance_from_sample(test_sample)

    train_label_np = self.Y_train.to_numpy()
    sort_index = np.argsort(distance_np)
    k_votes_index = sort_index[0:self.K]
    k_votes_labels = train_label_np[k_votes_index]
    labels_unique, labels_counts = np.unique(k_votes_labels, return_counts = T
rue)
    classified_label_index = np.argsort(labels_counts)
    return labels_unique[classified_label_index[-1]]
```

In [9]:
```
#1b.ii
# distinguish fetaures and labels from training data

data_train = data.loc[:,0:2]
label_train = data.loc[:,3]

#import dev data and seperate features and labels
data1= pd.read_csv('/content/drive/MyDrive/ML HW/data_dev.csv', header=None)
data_dev_ = data1.loc[:,0:2].to_numpy()
label_dev_ = data1.loc[:,3].to_numpy()

#count number of samples for each class
no_of_dev_data= data_dev_.shape[0]
no_of_dev_data_class1 = np.count_nonzero(label_dev_ == 1)
no_of_dev_data_class2 = np.count_nonzero(label_dev_==2)

#diffrent values of K
k_values = np.array([1,3,5,7,9,11,13])
Acc = np.empty(len(k_values))
BAcc = np.empty(len(k_values))

#hyperparameter tuning on dev data
for i, k in enumerate(k_values):
  my_classifier = KNN_classifier(k)
  my_classifier.fit(data_train, label_train)
  correct_classified = 0
  correct_classified_class1 = 0
  correct_classified_class_2 = 0

  for indeX, dataX in enumerate(data_dev_):
    res = my_classifier.predict(dataX)
    if res == label_dev_[indeX]:
      correct_classified +=1
      if res == 1:
        correct_classified_class1 += 1
      else:
        correct_classified_class_2 += 1

  Acc[i]  = correct_classified/no_of_dev_data
  BAcc[i] = 0.5 * (correct_classified_class1/ no_of_dev_data_class1) + 0.5 * (
correct_classified_class_2/ no_of_dev_data_class2)

  k_star1 = k_values[np.argmax(np.array(Acc))]
  k_star2 = k_values[np.argmax(np.array(BAcc))]

print(f"The value for K* that gives highest Acc is: {k_star1}")
print(f"The value for K** that gives highest BAcc is: {k_star2}")
```

```
The value for K* that gives highest Acc is: 5
The value for K** that gives highest BAcc is: 5
```

In [10]:
```
plt.plot(k_values, Acc)
```

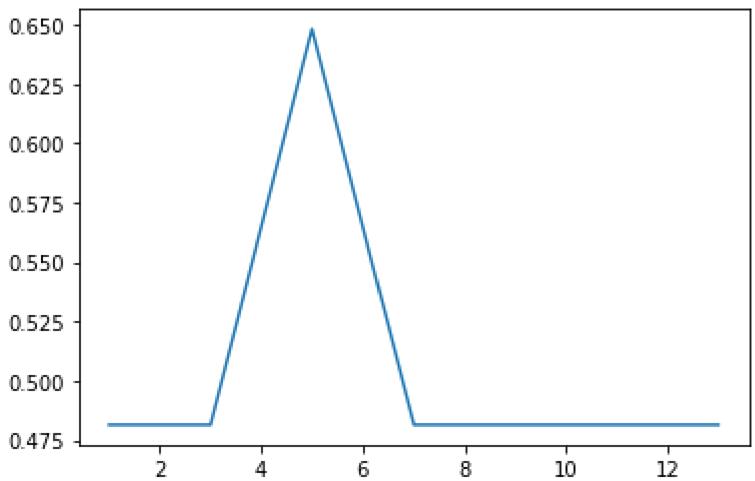
Out[10]: [<matplotlib.lines.Line2D at 0x7fa75fd53890>]



Answer to question 1b.ii : From the above graph,the value for K* that gives highest Acc is: 5

In [11]:
```
plt.plot(k_values, BAcc)
```

Out[11]: [<matplotlib.lines.Line2D at 0x7fa75fcea790>]



Answer to question 1b.ii : From the above graph,the value for K* that gives highest BAcc is: 5

In [12]:
```
#1b.iii
# distinguish fetaures and labels from training data

#import test data and seperate features and labels
data2= pd.read_csv('/content/drive/MyDrive/ML HW/data_test.csv', header=None)
data_test_ = data2.loc[:,0:2].to_numpy()
label_test_ = data2.loc[:,3].to_numpy()

no_of_test_data= data_test_.shape[0]
no_of_test_data_class1 = np.count_nonzero(label_test_ == 1)
no_of_test_data_class2 = np.count_nonzero(label_test_==2)


#different values of K
k_vals = np.array([k_star1, k_star2])
Acc_test = np.empty(len(k_vals))
BAcc_test = np.empty(len(k_vals))

#on test data with K* and K**
for i, k in enumerate(k_vals):
  my_classifier = KNN_classifier(k)
  my_classifier.fit(data_train, label_train)
  correct_classified_test = 0
  correct_classified_test_class1 = 0
  correct_classified_test_class_2 = 0

  for indexX, dataX in enumerate(data_test_):
    res = my_classifier.predict(dataX)
    if res == label_test_[indexX]:
      correct_classified_test +=1
      if res == 1:
        correct_classified_test_class1 += 1
      else:
        correct_classified_test_class_2 += 1

  Acc_test[i]  = correct_classified_test/no_of_test_data
  BAcc_test[i] = 0.5 * (correct_classified_test_class1/ no_of_test_data_class1
) + 0.5 * (correct_classified_test_class_2/ no_of_test_data_class2)
print(f"The value for Acc with K* and K** on the testing set are respectively:
{Acc_test}")
print(f"The value for BAcc with K* and K** on the testing set are respectivel
y: {BAcc_test}")
```

```
The value for Acc with K* and K** on the testing set are respectively: [0.838
70968 0.83870968]
The value for BAcc with K* and K** on the testing set are respectively: [0.64
666667 0.64666667]
```

Answer to question 1b.iii:

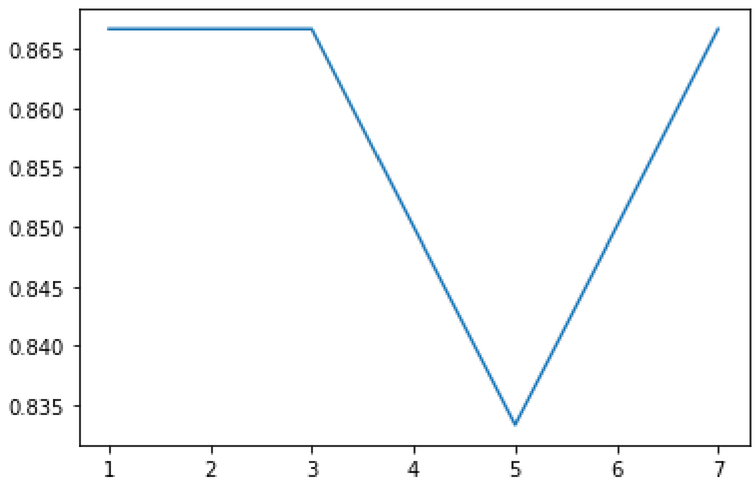
The value for Acc with K* on the testing set is 0.83870968

The value for BAcc with K** on the testing set is 0.64666667

In [13]:
```
#1b.iv
kk = [1,3,5,7]
Acc_dev = np.empty(len(kk))
BAcc_dev = np.empty(len(kk))

#test on dev data with l1 norm or Manhattan Distance
for i, k in enumerate(kk):
  my_classifier = KNN_classifier(k, distance='l1')
  my_classifier.fit(data_train, label_train)
  correct_classified_dev = 0
  correct_classified_dev_class1 = 0
  correct_classified_dev_class_2 = 0

  for indexX, dataX in enumerate(data_dev_):
    res = my_classifier.predict(dataX)
    if res == label_dev_[indexX]:
      correct_classified_dev +=1
      if res == 1:
        correct_classified_dev_class1 += 1
      else:
        correct_classified_dev_class_2 += 1
  print("distance" , my_classifier.distance)
  Acc_dev[i]  = correct_classified_dev/no_of_dev_data
  BAcc_dev[i] = 0.5 * (correct_classified_dev_class1/ no_of_dev_data_class1) +
0.5 * (correct_classified_dev_class_2/ no_of_dev_data_class2)

print(f"With l1 norm or Manhattan distance, the Acc are {Acc_dev} for K = {kk}
respectively.")
print(f"With l1 norm or Manhattan distance, the BAcc are {BAcc_dev} for K = {k
k} respectively.")
plt.plot(kk, Acc_dev)
```

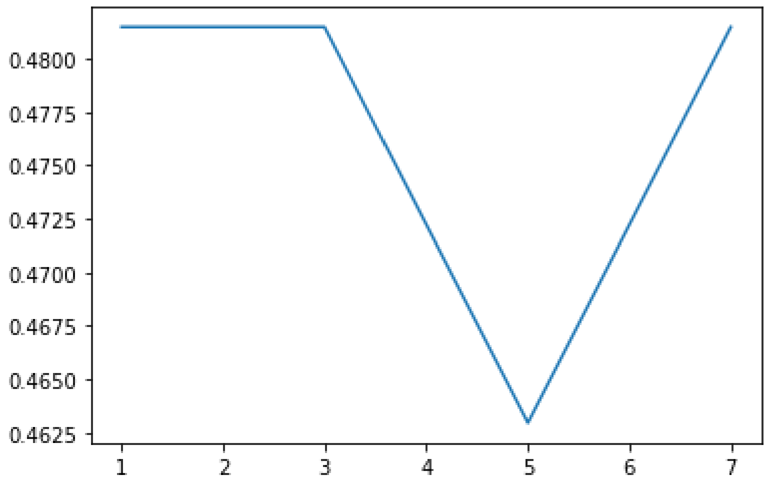
```
distance l1
distance l1
distance l1
distance l1
With l1 norm or Manhattan distance, the Acc are [0.86666667 0.86666667 0.8333
3333 0.86666667] for K = [1, 3, 5, 7] respectively.
With l1 norm or Manhattan distance, the BAcc are [0.48148148 0.48148148 0.462
96296 0.48148148] for K = [1, 3, 5, 7] respectively.
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x7fa75fc63310>]
```



In [14]:
```
plt.plot(kk, BAcc_dev)
k_star1 = kk[np.argmax(np.array(Acc_dev))]
k_star2 = kk[np.argmax(np.array(BAcc_dev))]

my_classifier = KNN_classifier(k_star1, distance='l1')
my_classifier.fit(data_train, label_train)
correct_classified_test = 0
correct_classified_test_class1 = 0
correct_classified_test_class_2 = 0

for indeX, dataX in enumerate(data_test_):
  res = my_classifier.predict(dataX)
  if res == label_test_[indeX]:
    correct_classified_test +=1
    if res == 1:
      correct_classified_test_class1 += 1
    else:
      correct_classified_test_class_2 += 1
print("distance" , my_classifier.distance)
Acc_test_l1= correct_classified_test/no_of_test_data

my_classifier = KNN_classifier(k_star2, distance='l1')
my_classifier.fit(data_train, label_train)
correct_classified_test = 0
correct_classified_test_class1 = 0
correct_classified_test_class_2 = 0

for indeX, dataX in enumerate(data_test_):
  res = my_classifier.predict(dataX)
  if res == label_test_[indeX]:
    correct_classified_test +=1
    if res == 1:
      correct_classified_test_class1 += 1
    else:
      correct_classified_test_class_2 += 1

print("distance", my_classifier.distance)
BAcc_test_l1 = 0.5 * (correct_classified_test_class1/ no_of_test_data_class1)
+ 0.5 * (correct_classified_test_class_2/ no_of_test_data_class2)

print(f"With l1 norm or Manhattan distance, the Acc are {Acc_test_l1} for K =
 {k_star1}.")
print(f"With l1 norm or Manhattan distance, the BAcc are {BAcc_test_l1} for K
 = {k_star2}.")
```

```
distance l1
distance l1
With l1 norm or Manhattan distance, the Acc are 0.9032258064516129 for K = 1.
With l1 norm or Manhattan distance, the BAcc are 0.75 for K = 1.
```



Answer to question 1b.iv

The best performance using Manhattan distance considering Acc and BAcc are found with K=1,3 and 7. If we choose, k=1, that will be computationally effiecient but the boundary will be comparatively noisy.

Question 1c: ML Deployment

The dataset is highly imabalanced as there are much less samples in class 2 compared to class 1. And that is reflected with Acc and BAcc metric as well. When the model considers correctly classified samples for class 2, the accuracy drops to 64% from 84%. So my concern is class 2 will have less correctly classified cases than class 1. My thought is to collect more samples for class 2 or choose other algorithms to handle this issue.

In [17]:
```
%cd /content/drive/MyDrive/Colab Notebooks
```

```
/content/drive/MyDrive/Colab Notebooks
```

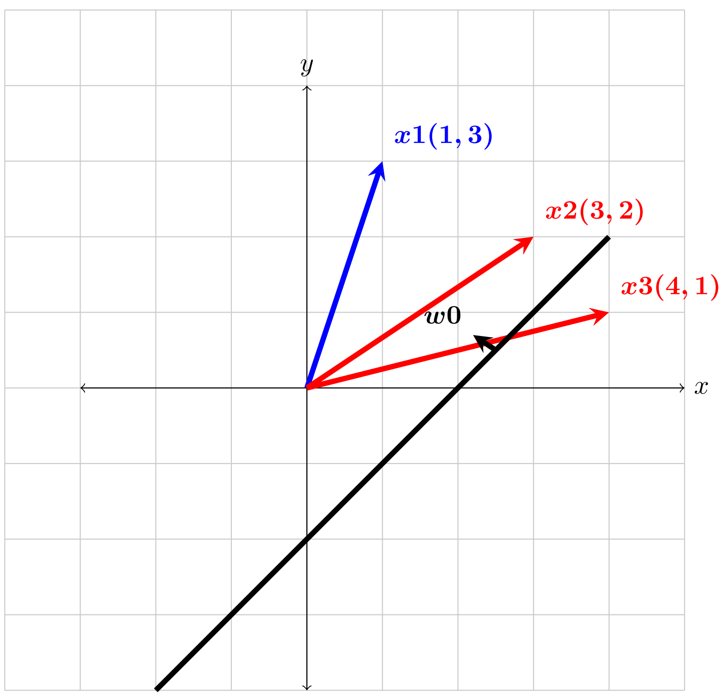
In [ ]:
```
!jupyter nbconvert --to html ML_Fall_HW1.ipynb
```

EEE 633

September 27, 2021

Question 2

Question 2.i



In the above graph, the blue line corresponds to the sample belonging to class 1, the red lines correspond to the samples belonging to class 2 and the black line corresponds to weight, w[0].

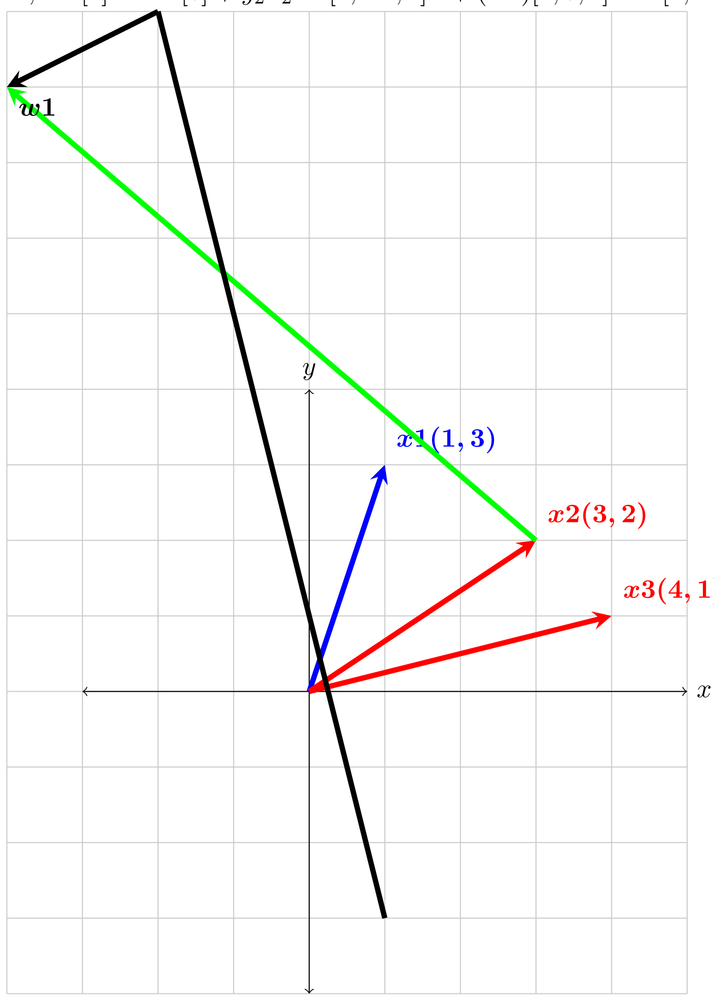
Question 2.ii

$w^T[0] = [2, -1, 1]$.
$w^T[0]x_1 = 4 > 0$. It is correctly classified to class 1
$w^T[0]x_2 = 1 > 0$. It is incorrectly classified to class 1

$w^T[0]x_3 = -1 > 0$. It is correctly classified to class 2

Question 2.iii

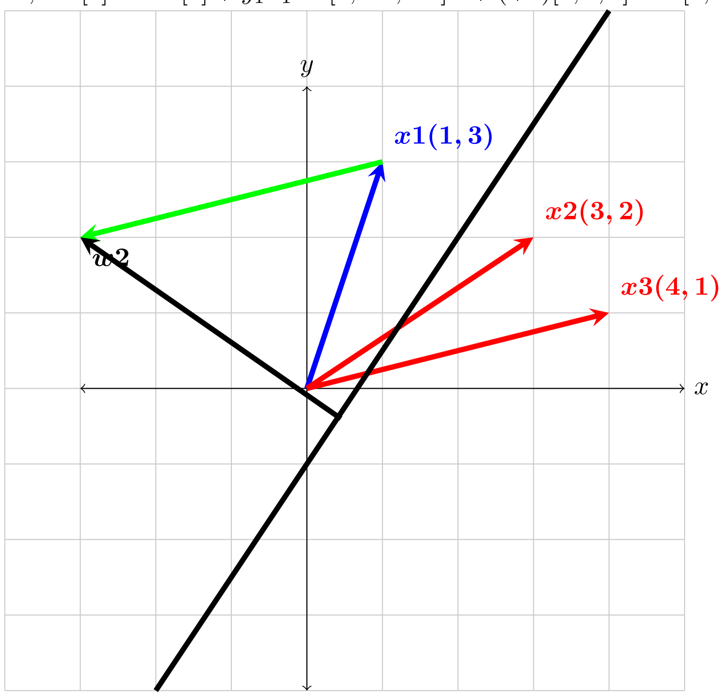
As sample x2 is misclassified,
so, $w^T[1] = w^T[0] + y_2 x_2 = [2, -1, 1]^T + (-1)[1, 3, 2]^T = [1, -4, -1]^T$.



In the above graph, the black line corresponds to the updated weight, w[1] and the green line corresponds to the previous weight direction, w[0].
$w^T[1]x_1 < 0$. It is incorrectly classified to class 2
$w^T[1]x_2 < 0$. It is correctly classified to class 2
$w^T[1]x_3 < 0$. It is correctly classified to class 2

Question 2.iv

As sample x1 is misclassified,
so, $w^T[2] = w^T[1] + y_1 x_1 = [1, -4, -1]^T + (+1)[1, 1, 3]^T = [2, -3, 2]^T$.



In the above graph, the black line corresponds to the updated weight, w[2] and the green line corresponds to the previous weight direction, w[1].
$w^T[2]x_1 < 0$. It is correctly classified to class 1
$w^T[2]x_2 > 0$. It is correctly classified to class 2
$w^T[2]x_3 > 0$. It is correctly classified to class 2