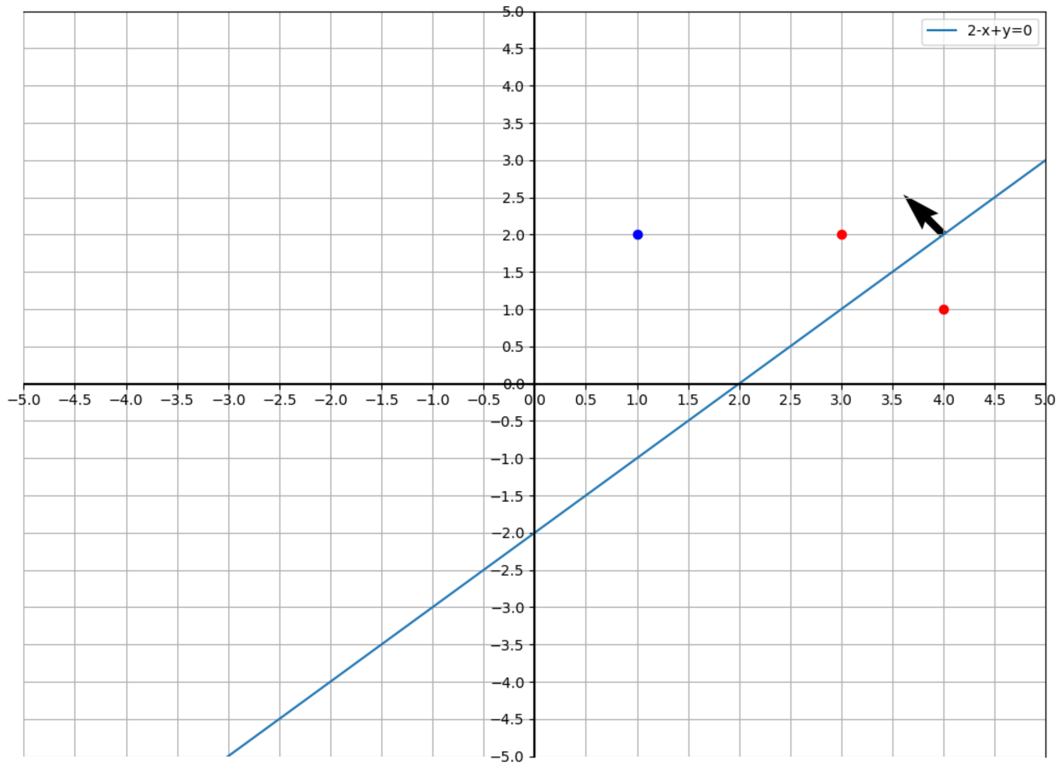
CSCE 633 - Machine Learning - Home work 1

Guru Sarath Thangamani
UIN: 829009551

February 10, 2020

1 Question 1

(i)



(ii)

$2 - x + y = 0$
$2 - 1(1) + (3) = 4 \implies y1 = 1$ ✓
$2 - 1(3) + (2) = 1 \implies y2 = 1$ ✖
$2 - 1(4) + (1) = -1 \implies y3 = -1$ ✓
Sample 2 was not calssified correctly.

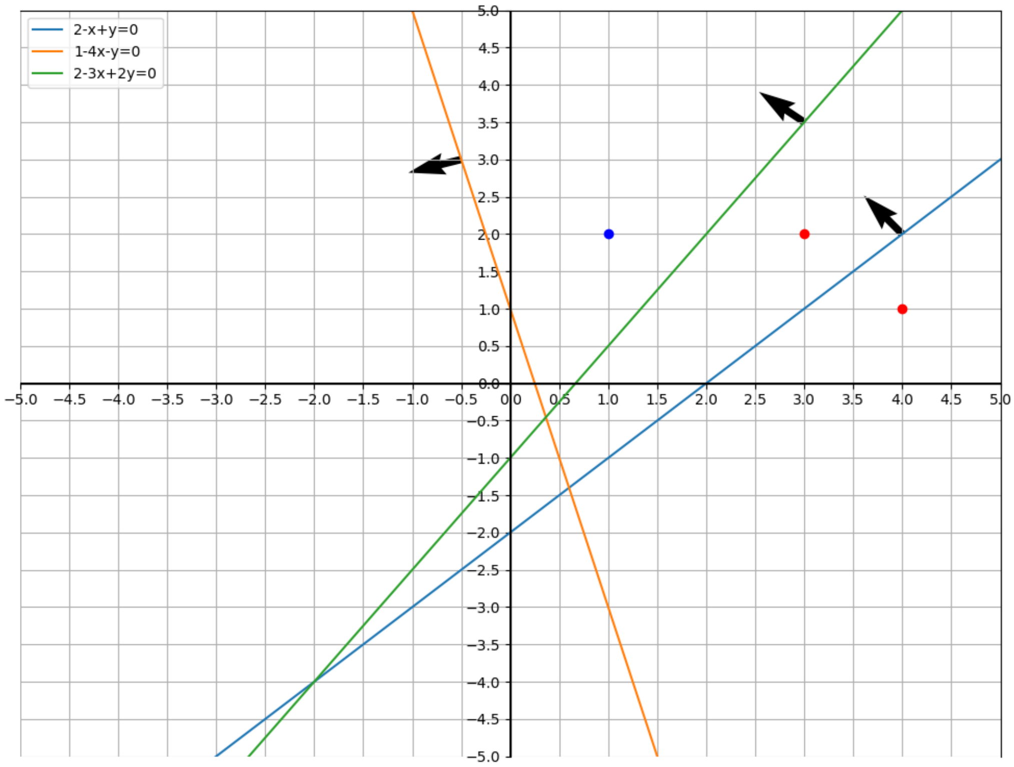
(iii)
w(0) - x2 = w(1)

$$\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ -1 \end{bmatrix}$$

$1 - 4x - y = 0$
$1 - 4(1) - 1(3) = -6 \implies y1 = -1$ ✖
$1 - 4(3) - 1(2) = -13 \implies y2 = -1$ ✓
$1 - 4(4) - 1(1) = -16 \implies y3 = 1$ ✓
Sample 1 is wrongly classified. Samples 2 and 3 are correctly classified.
(For plot see next page)
(iv)
w(1) + x1 = w(2)

$$\begin{bmatrix}1\\-4\\-1\end{bmatrix}+\begin{bmatrix}1\\1\\3\end{bmatrix}=\begin{bmatrix}2\\-3\\2\end{bmatrix}$$

$2-3x+2y=0$
$2-3(1)+2(3)=5 \implies y1=1$ ✓
$2-3(3)+2(2)=-3 \implies y2=-1$ ✓
$2-3(4)+2(1)=-8 \implies y3=-1$ ✓
All the three samples are correctly classified.

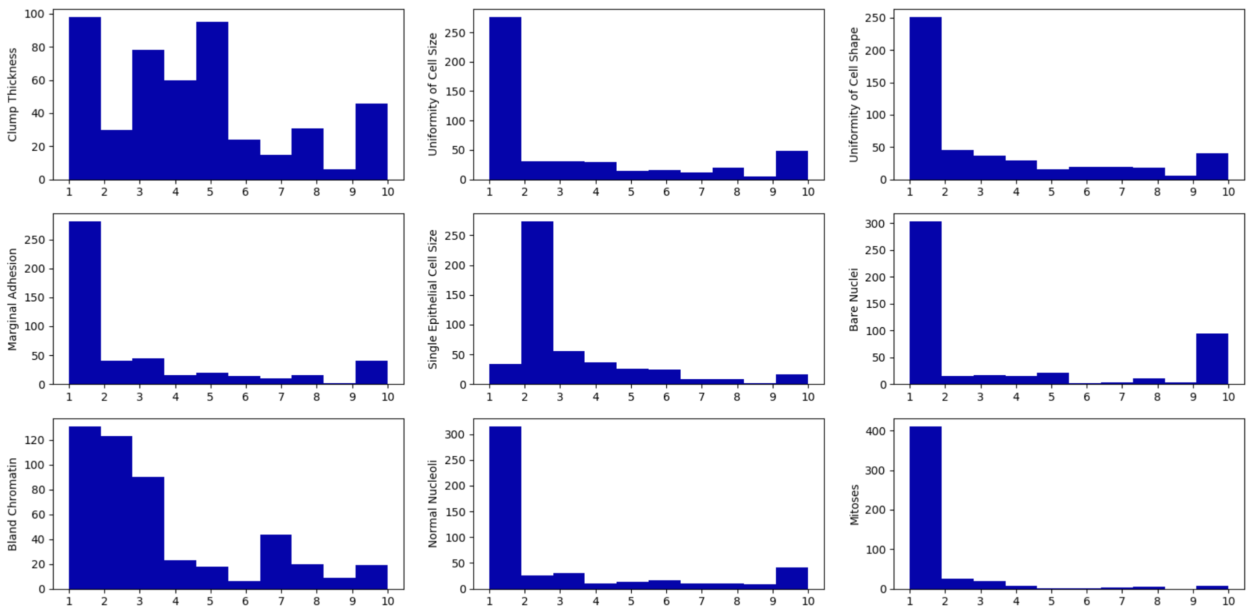


2 Problem 2

(a,i) NO, the two sample values are not equally distributed.
Number of benign cases = 330
Number of Malignant cases = 153

```
import numpy as np
import matplotlib.pyplot as plt

#Load the data set
train_set = np.loadtxt('hw1_question1_train.csv', delimiter = ',', dtype = int)
dev_set = np.loadtxt('hw1_question2_dev.csv', delimiter = ',', dtype = int)
test_set = np.loadtxt('hw1_question2_test.csv', delimiter = ',', dtype = int)

train_set_outcome = train_set[:,-1]
print('Number of benign cases = ', np.sum(train_set_outcome == 2))
print('Number of malignant cases = ', np.sum(train_set_outcome == 4))
```

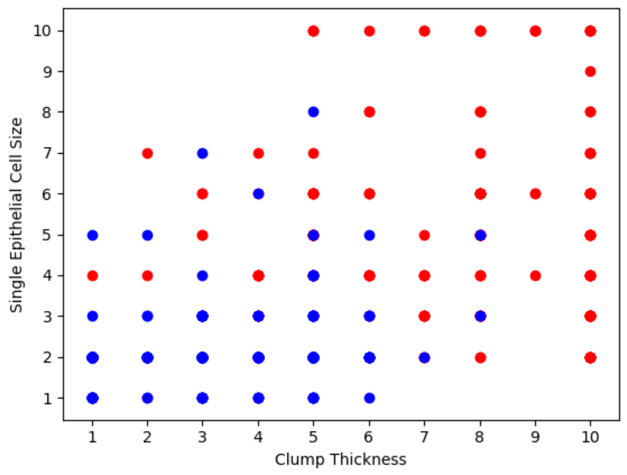
(a,ii)
Clump thinkness is distributed almost evenly.
But, most of the other sample values are not distributed equally, they are skewed to the left.
Most sample vlaues are having values less than 5.

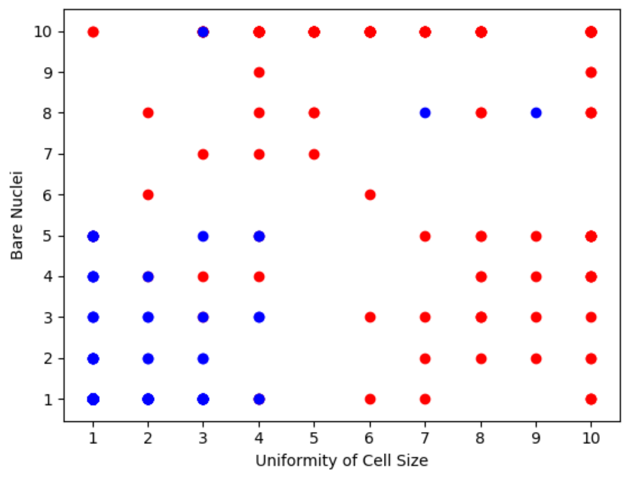


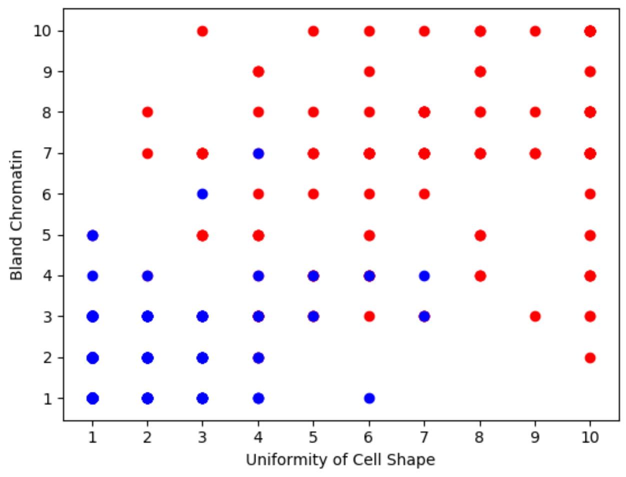
```
#Question (a.ii)
feature_names = ['Clump Thickness',
'Uniformity of Cell Size',
'Uniformity of Cell Shape',
'Marginal Adhesion',
'Single Epithelial Cell Size',
'Bare Nuclei',
'Bland Chromatin',
'Normal Nucleoli',
'Mitoses']

features = []
for i in range(9):
        features.append(train_set[:,i])

xi = [1,2,3,4,5,6,7,8,9,10]
for i in range(9):
        plt.subplot(3,3,i+1)
        plt.ylabel(feature_names[i])
        plt.xticks(xi)
        plt.hist(x=features[i], bins=10, color='#0504aa')

plt.show()
```

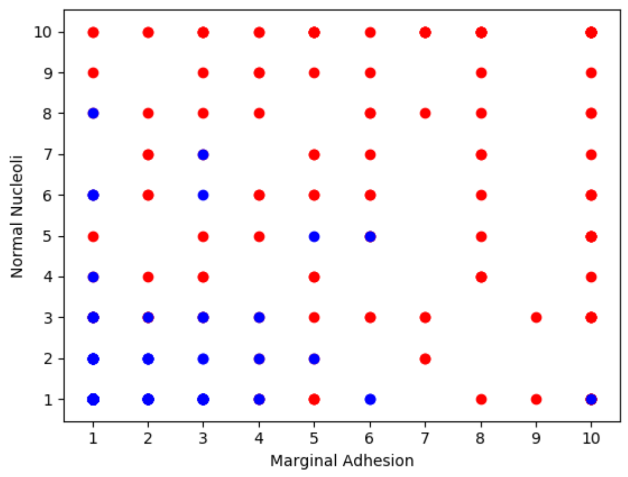
(b,i)

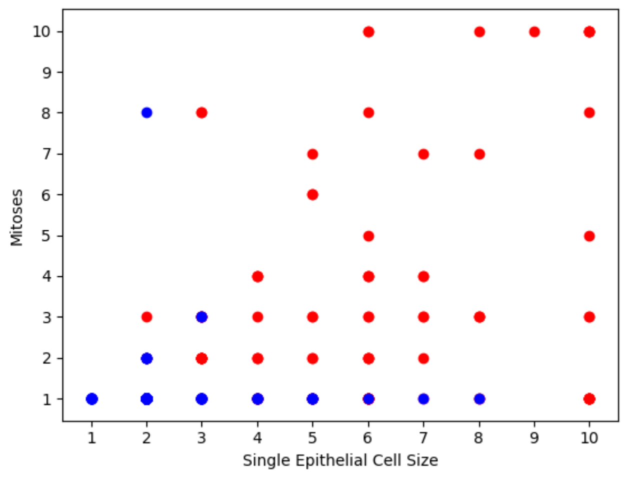
In all the cases shown below the data is not perfectly separable because there are few data points (of different class) with same feature vaules.
There are few outliers in dataset due to which separating the two classes perfectly with a straight line is hard. However, benign and malignant classes of data are clustered in two regions, which show some trend in the data.











```
#Question (a.iii)
train_set_outcome = train_set[:,-1]

for i in range(5):

        j = i + 4

        x = features[i]
```

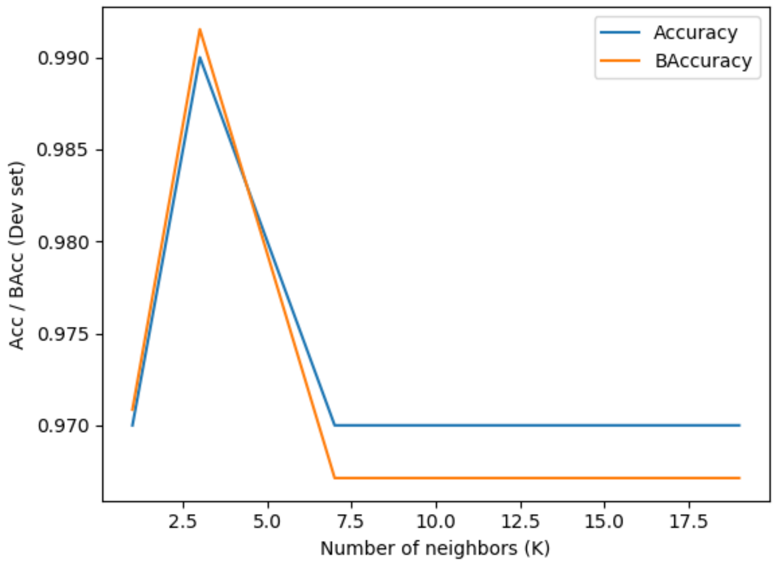
```
y = features[j]

x_benign = []
y_benign = []
x_malign = []
y_malign = []

for k,outcome in enumerate(train_set_outcome):
        if outcome == 2:
                x_benign.append(x[k])
                y_benign.append(y[k])
        if outcome == 4:
                x_malign.append(x[k])
                y_malign.append(y[k])

plt.xlabel(feature_names[i])
plt.ylabel(feature_names[j])
plt.xticks(xi)
plt.yticks(xi)
plt.scatter(x_malign,y_malign, c='red')
plt.scatter(x_benign, y_benign, c='Blue')
plt.show()
```

(b,ii)

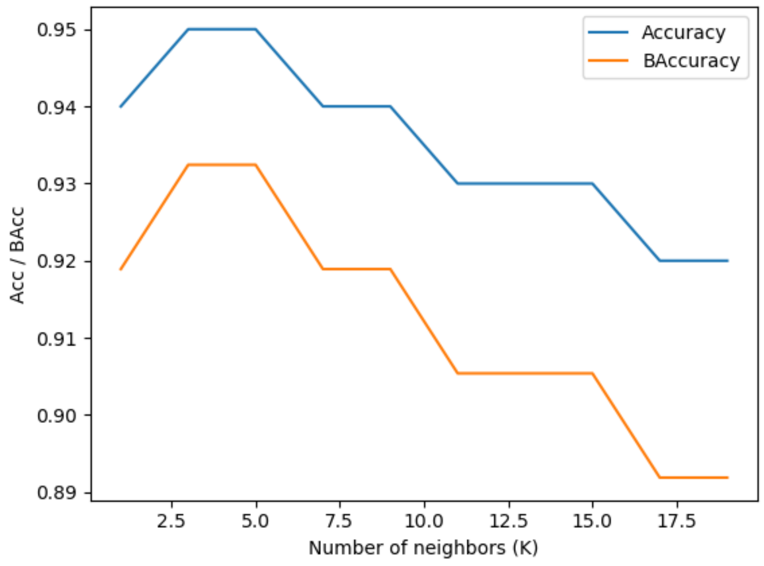
```
def L2_Norm( sample1 , sample2 ):
        delta = sample1 - sample2
        delta = delta**2
        sum_ = np.sum(delta)
        return sum_**0.5

def norm(element):
        return element[0]

def predict(NormsAndOutComesList, K=3):

        class_dict = dict()

        for i in range(K):

                outcome = NormsAndOutComesList[i][1]

                if outcome not in class_dict:
                        class_dict[outcome] = 1
                else:
                        class_dict[outcome] += 1

        max_class = None
        max_count = 0

        for classX in class_dict:

                if class_dict[classX] > max_count:
                        max_class = classX
                        max_count = class_dict[classX]

        return max_class



def run(dataSet, trainDataSet, K=3):

        predictions = []

        for inputSample in dataSet:

                ListOfAllNorms_and_outcomes = []
                for trainSample in trainDataSet:

                        L2Norm = L2_Norm(inputSample[:-1], trainSample[:-1])
                        ListOfAllNorms_and_outcomes.append( (L2Norm , trainSample[-1]) )

                ListOfAllNorms_and_outcomes.sort(key=norm)
                predictions.append(predict(ListOfAllNorms_and_outcomes, K))

        #print(ListOfAllNorms_and_outcomes)
        #print(predictions)
        return np.array(predictions)

def accuracy(predictions, actual):
        correct = np.sum(predictions == actual)
        return correct / predictions.shape[0]

def Baccuracy(predictions, actual):
        correct_1 = np.sum( np.logical_and( np.equal(predictions, 2) , np.equal(actual, 2)) )
        correct_2 = np.sum( np.logical_and( np.equal(predictions, 4) , np.equal(actual, 4)) )

        samples_1 = np.sum(np.equal(actual, 2))
        samples_2 = np.sum(np.equal(actual, 4))

        print(correct_1, samples_1)
        print(correct_2, samples_2)

        return ((correct_1 / samples_1) + (correct_2 / samples_2))/2
```

```
for i in range(20):

        if i%2 == 0:
                continue

        print('K = ', i)
        preds = run(dev_set, train_set, K=i)
        print('Acc = ', accuracy(preds, dev_set[:,-1]))
        print('BAcc = ', Baccuracy(preds, dev_set[:,-1]))

for i in range(20):

        if i%2 == 0:
                continue

        print('K = ', i)
        preds = run(test_set, train_set, K=i)
        print('Acc = ', accuracy(preds, test_set[:,-1]))
        print('BAcc = ', Baccuracy(preds, dev_set[:,-1]))
```

(b,ii)
K1 = 3
K2 = 3
Both Accuracy and Baccuracy have their highest value at K=3.
However, Baccuracy is lower than accuracy for higher values of K.



(b,iii)
$K1 = 3$
$K1\ Accuracy = 0.95$
$K1\ BAccuracy = 0.9324324324324325$
$K2 = 3$
$K2\ Accuracy = 0.95$
$K2\ BAccuracy = 0.9324324324324325$



(b,iv) (Bonus)
From the results in the jupyter notebook for L1 norm, we can see that BAccuracy is generally lower than Accuracy.
L2 Norm performs best compared to L1 and Cosine similarity.
The K1 and K2 values for different distance functions are mostly similar (but not the same).
Cosine similarity seems to perform poorly for this dataset for classification (Very low accuracy).
see jupyter notebook for outputs