# SM in AI
# Assignment-1
# 201402163

## Q1(a) :Single-sample perceptron

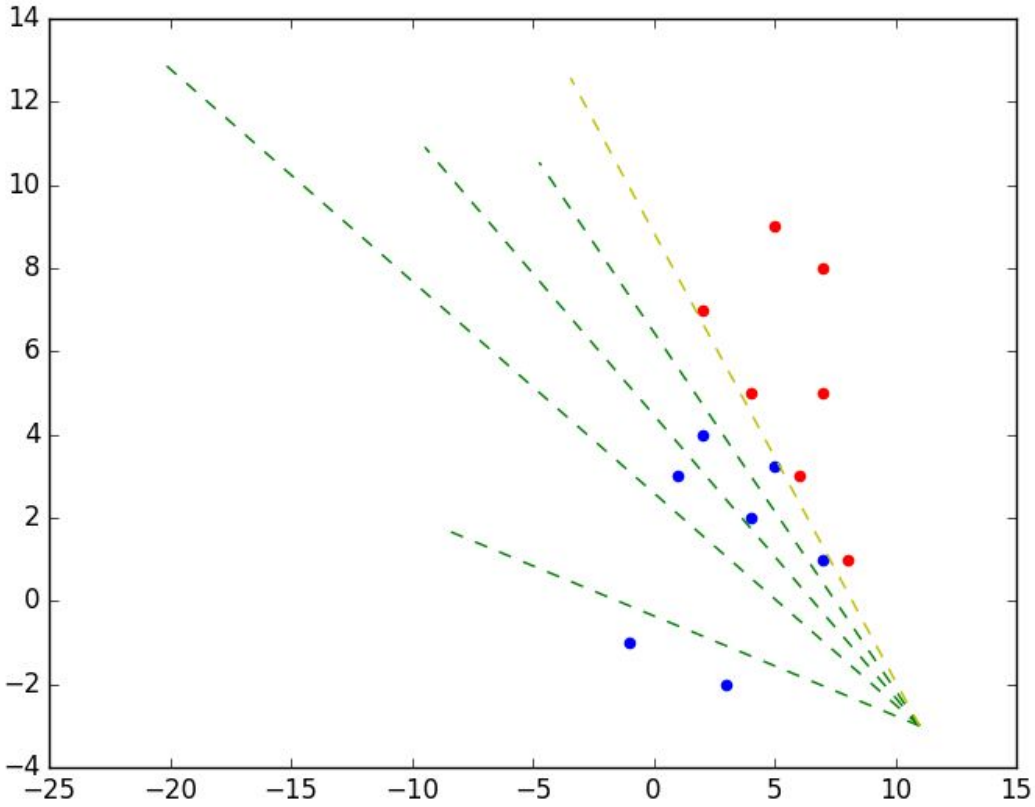



Fig -1Qa_1

w1(class)-red dot, w2(class)-blue dot
green lines(Dotted) -learning weights
Yellow line(Dotted)_final weight

---

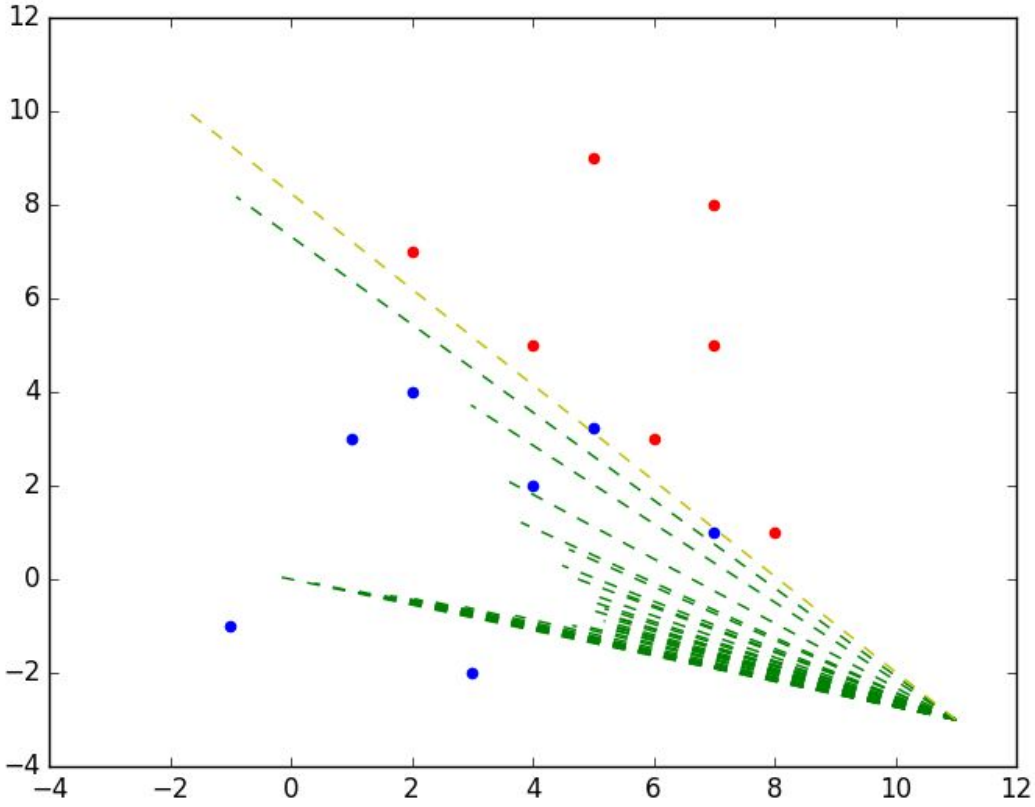
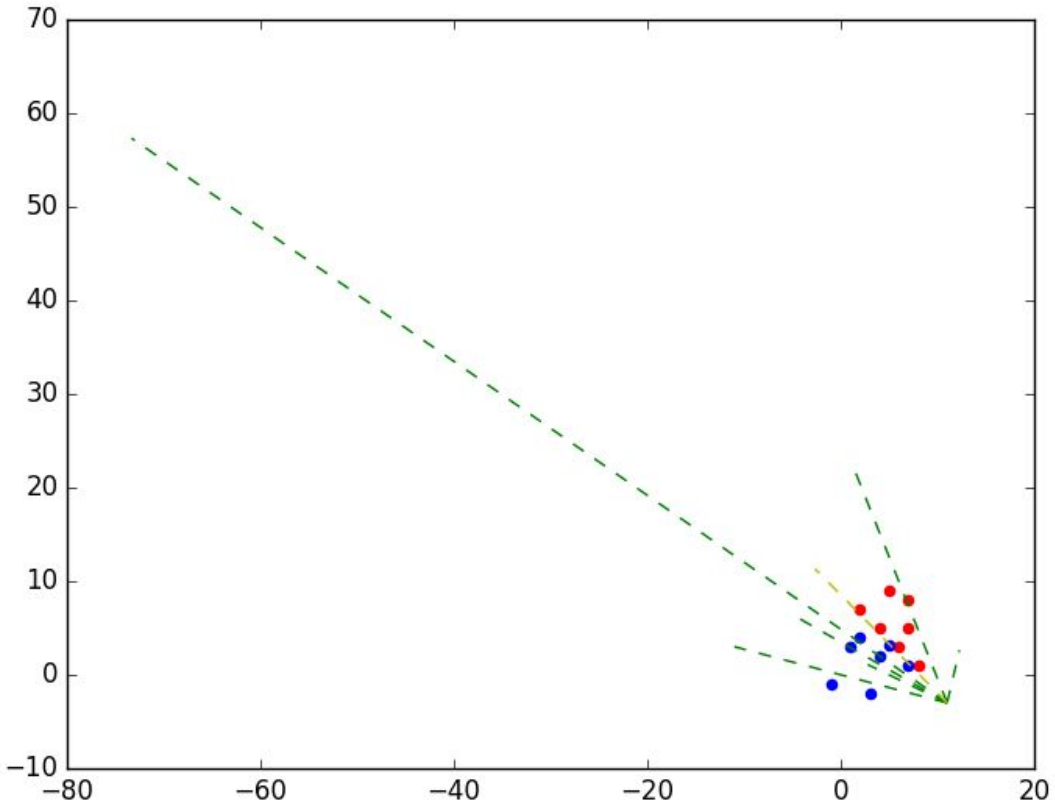



Fig 1Qa_2
With initial high weights(1,100,300)




With initial weights(1,-000078,-0.00001)

---

# Single sample perceptron Algorithm:

---

```
import numpy as np
import random
import matplotlib.pyplot as plt
w1=[(2,7),(8,1),(7,5),(6,3),(7,8),(5,9),(4,5)]
w2=[(4,2),(-1,-1),(1,3),(3,-2),(5,3.25),(2,4),(7,1)]
Y_MAT=[[-1,-4,-2],[-1,1,1],[-1,-1,-3],[-1,-3,2],[-1,-5,-3.25],[-1,-2,-4],[-1,-7,-1],[1,2,7],
[1,8,1],[1,7,5],[1,6,3],[1,7,8],[1,5,9],[1,4,5]]
#print Y_MAT
b=1000#margin
Str_WEI=[]
A_MAT=np.array(Y_MAT)
WEI_MAT=[1,-1,1]
WEI_MAT=np.array(WEI_MAT)
loop=100000
while loop:
        DotMat=np.dot(A_MAT,WEI_MAT)
        #print DotMat,'This is dot product------'
```

```
        Dot_Mat=A_MAT[DotMat<b]
        #print Dot_Mat,'This is values of y========'
        if len(Dot_Mat):
        WEI_MAT=WEI_MAT+Dot_Mat[random.randint(0,len(Dot_Mat)-1)]
        #print loop%100
        if int(loop%5000)==0:
                Str_WEI.append(WEI_MAT)
        else:
        break
        loop -=1
#print Str_WEI
Line_Mat_pt=[]#for storing weight matrix lines

x1=[]
y1=[]
x2=[]
y2=[]
for i in w1:
        x1.append(i[0])
        y1.append(i[1])
for i in w2:
        x2.append(i[0])
        y2.append(i[1])

_p1x=float(11)
_p2x=float(-3)
_p1y=float(_p1x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))
_p2y=float(_p2x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))

for i in Str_WEI:
        if i[2]!=0:
        _L1x=float(11)
        _L2x=float(-3)
        _L1y=float(_p1x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
        _L2y=float(_p2x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
        Line_Mat_pt.append([[_L1x,_L1y],[_L2x,_L2y]])

for i in Line_Mat_pt:
        plt.plot(i[0],i[1],'g--')

plt.scatter(x1,y1,color='r')
plt.scatter(x2,y2,color='b')
plt.plot([_p1x,_p1y],[_p2x,_p2y],'y--')
plt.show()
print WEI_MAT,loop
```

- **When weights are given which are solution the algorithm terminates immediately**
- **There is no relation between initial weights the time taken by algorithm ,some time it takes more iterations to converge some  time it takes less time converge**

---

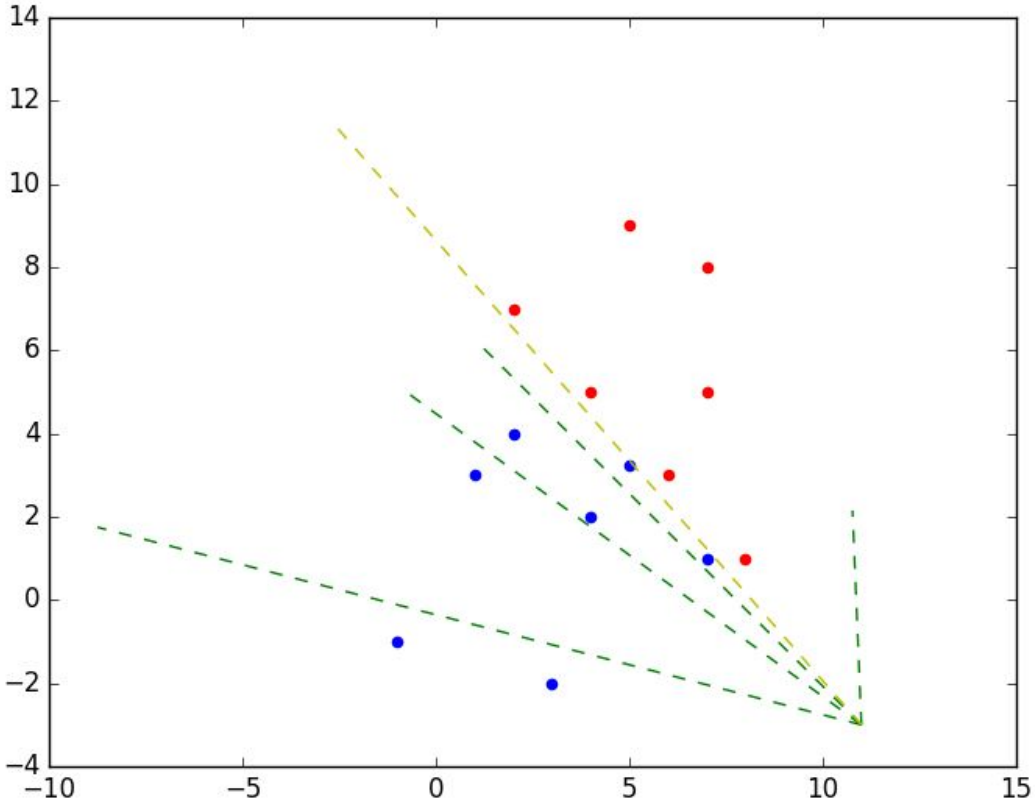
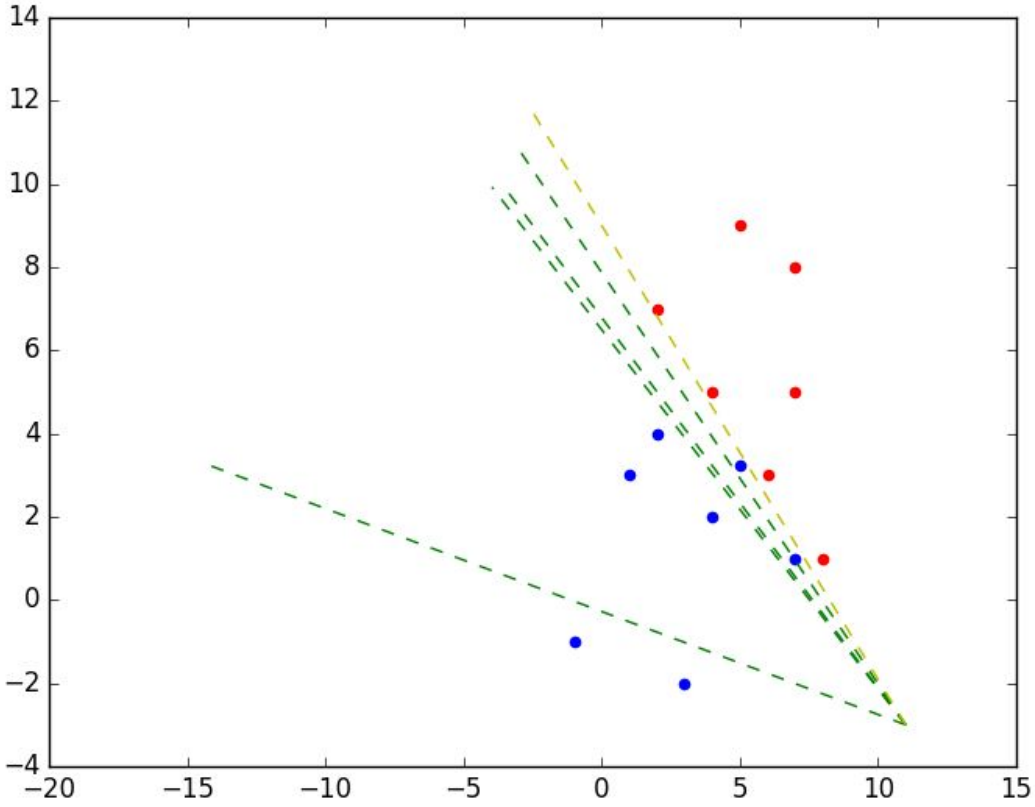
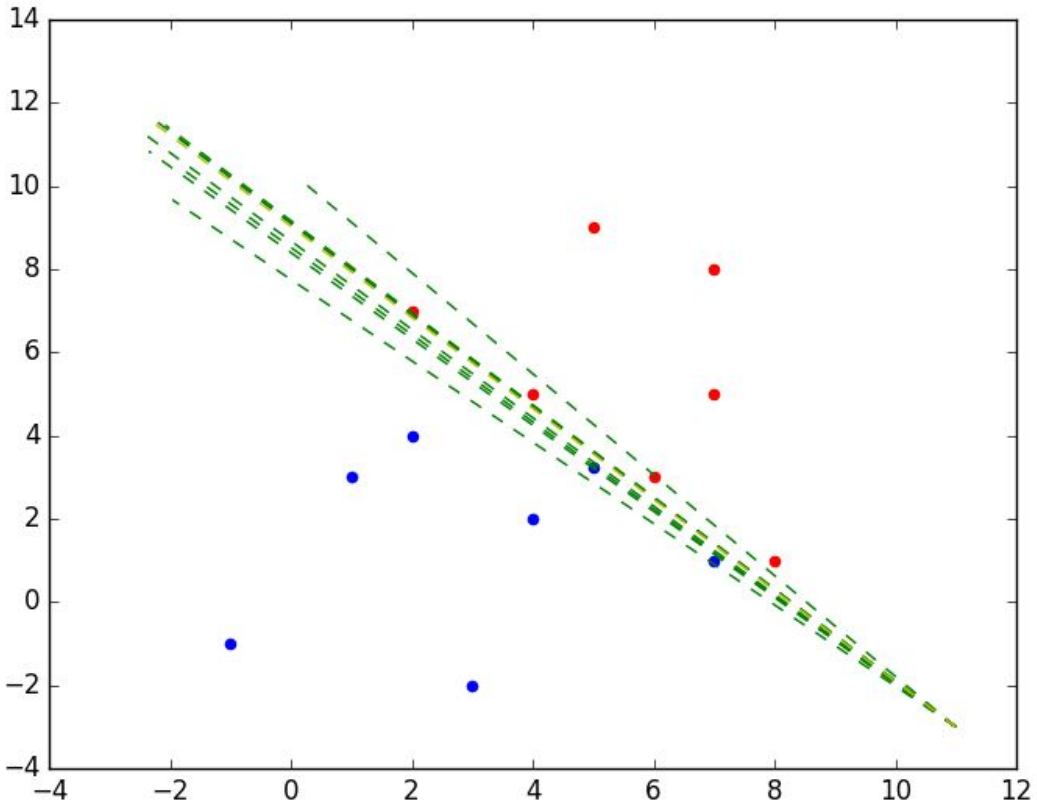



Fig -1Qb_1
w1(class)-red dot, w2(class)-blue dot
green lines(Dotted) -learning weights
Yellow line(Dotted)_final weight




With medium margin(10)




With High margin(100)

---

# Single-sample perceptron with margin_Alogrithm

---

```
import numpy as np
import random
import matplotlib.pyplot as plt
w1=[(2,7),(8,1),(7,5),(6,3),(7,8),(5,9),(4,5)]
w2=[(4,2),(-1,-1),(1,3),(3,-2),(5,3.25),(2,4),(7,1)]
Y_MAT=[[-1,-4,-2],[-1,1,1],[-1,-1,-3],[-1,-3,2],[-1,-5,-3.25],[-1,-2,-4],[-1,-7,-1],[1,2,7],
[1,8,1],[1,7,5],[1,6,3],[1,7,8],[1,5,9],[1,4,5]]
#print Y_MAT
b=1000#margin
Str_WEI=[]
A_MAT=np.array(Y_MAT)
WEI_MAT=[1,-1,1]
```

```
WEI_MAT=np.array(WEI_MAT)
loop=100000
while loop:
        DotMat=np.dot(A_MAT,WEI_MAT)
        #print DotMat,'This is dot product------'
        Dot_Mat=A_MAT[DotMat<b]
        #print Dot_Mat,'This is values of y========'
        if len(Dot_Mat):
        WEI_MAT=WEI_MAT+Dot_Mat[random.randint(0,len(Dot_Mat)-1)]
        #print loop%100
        if int(loop%5000)==0:
                Str_WEI.append(WEI_MAT)
        else:
        break
        loop -=1
#print Str_WEI
Line_Mat_pt=[]#for storing weight matrix lines

x1=[]
y1=[]
x2=[]
y2=[]
for i in w1:
        x1.append(i[0])
        y1.append(i[1])
for i in w2:
        x2.append(i[0])
        y2.append(i[1])

_p1x=float(11)
_p2x=float(-3)
_p1y=float(_p1x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))
_p2y=float(_p2x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))

for i in Str_WEI:
        if i[2]!=0:
        _L1x=float(11)
        _L2x=float(-3)
        _L1y=float(_p1x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
        _L2y=float(_p2x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
        Line_Mat_pt.append([[_L1x,_L1y],[_L2x,_L2y]])

for i in Line_Mat_pt:
        plt.plot(i[0],i[1],'g--')

plt.scatter(x1,y1,color='r')
plt.scatter(x2,y2,color='b')
plt.plot([_p1x,_p1y],[_p2x,_p2y],'y--')
plt.show()
print WEI_MAT,loop
```

---

# 1Qc:

# Relaxation algorithm  with margin

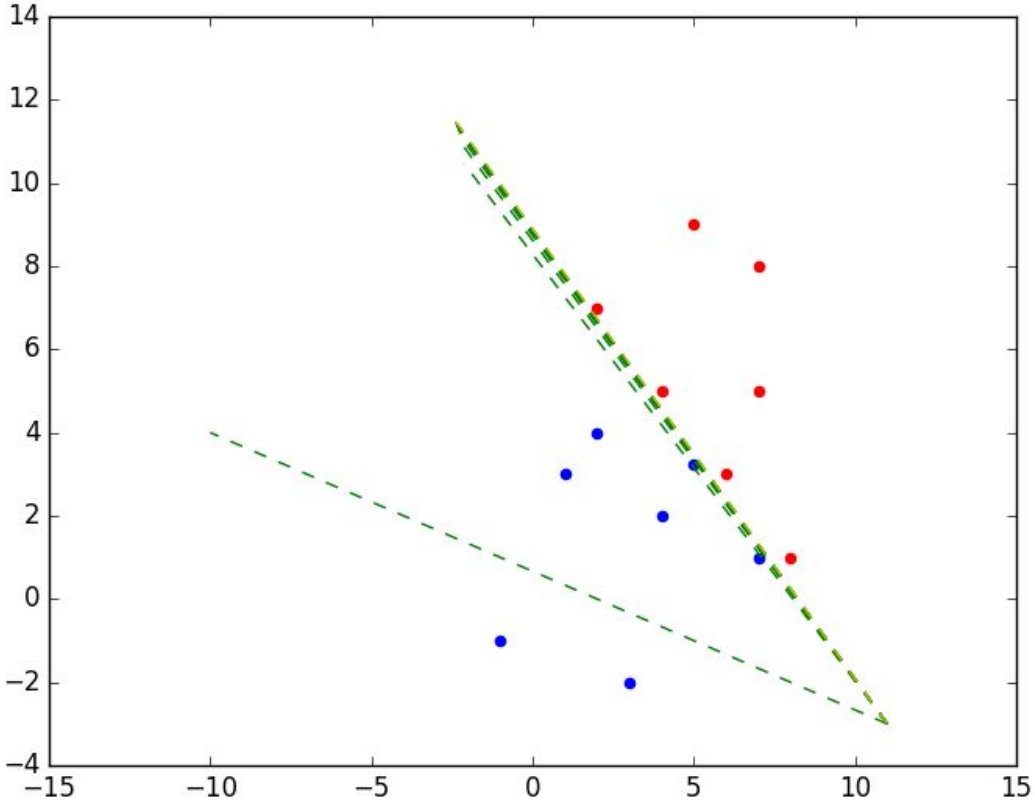
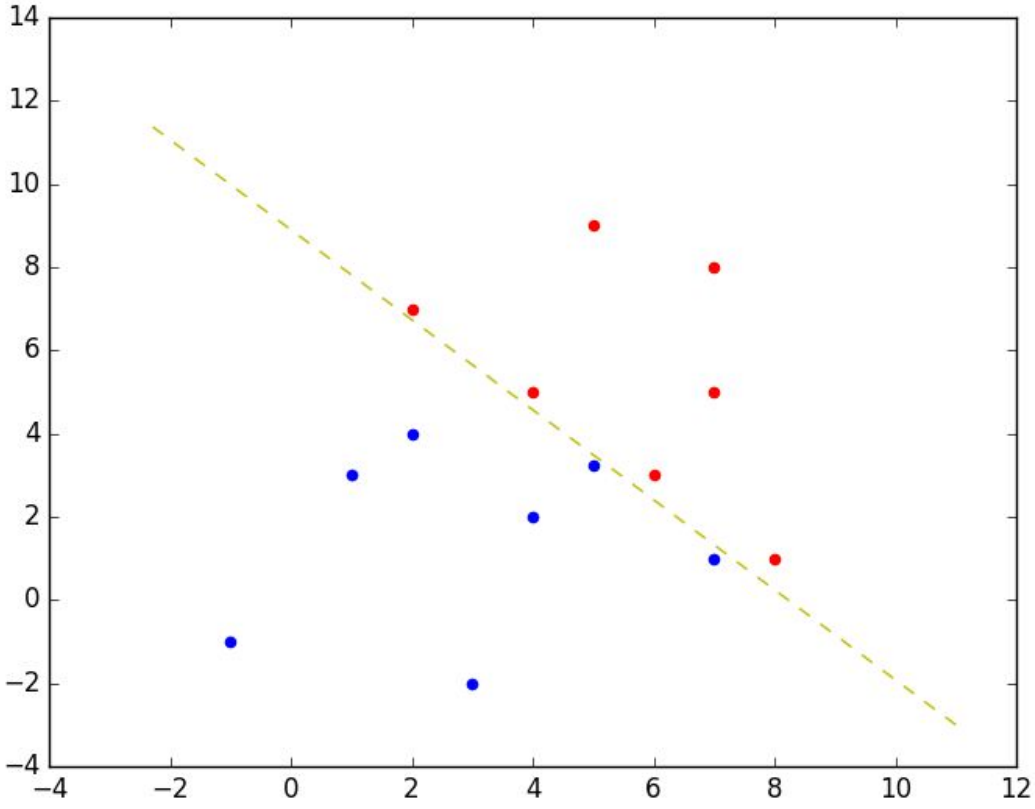


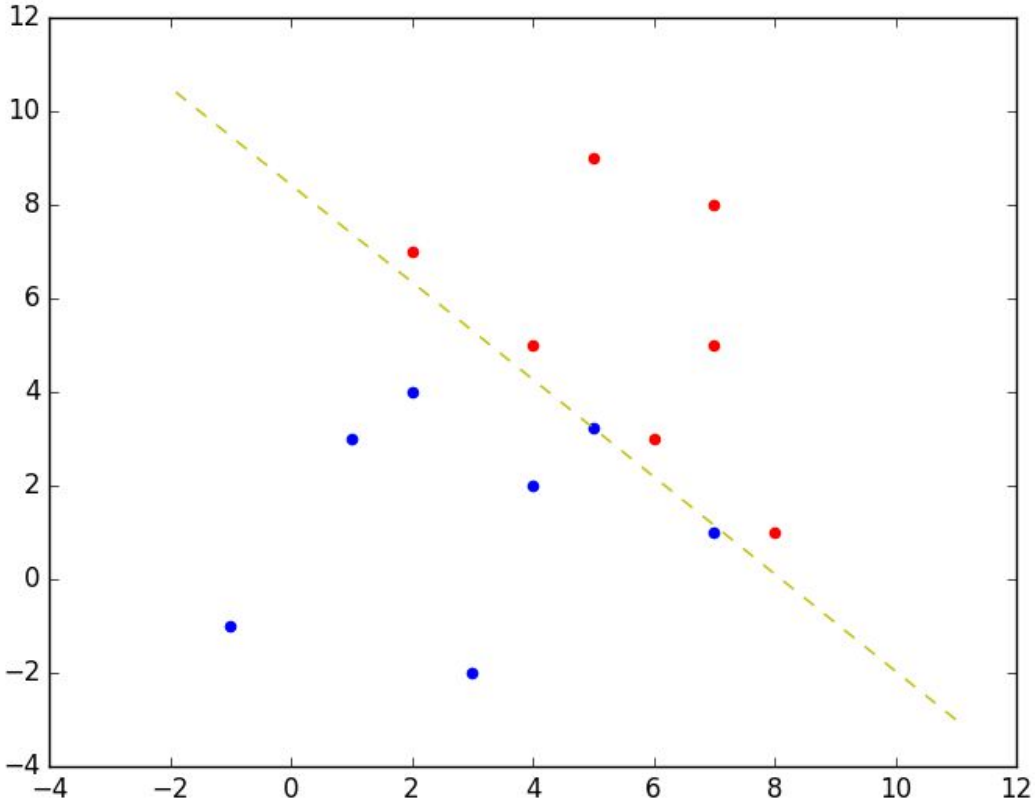

Fig -1Qc_1
w1(class)-red dot, w2(class)-blue dot
green lines(Dotted) -learning weights
Yellow line(Dotted)_final weight

---



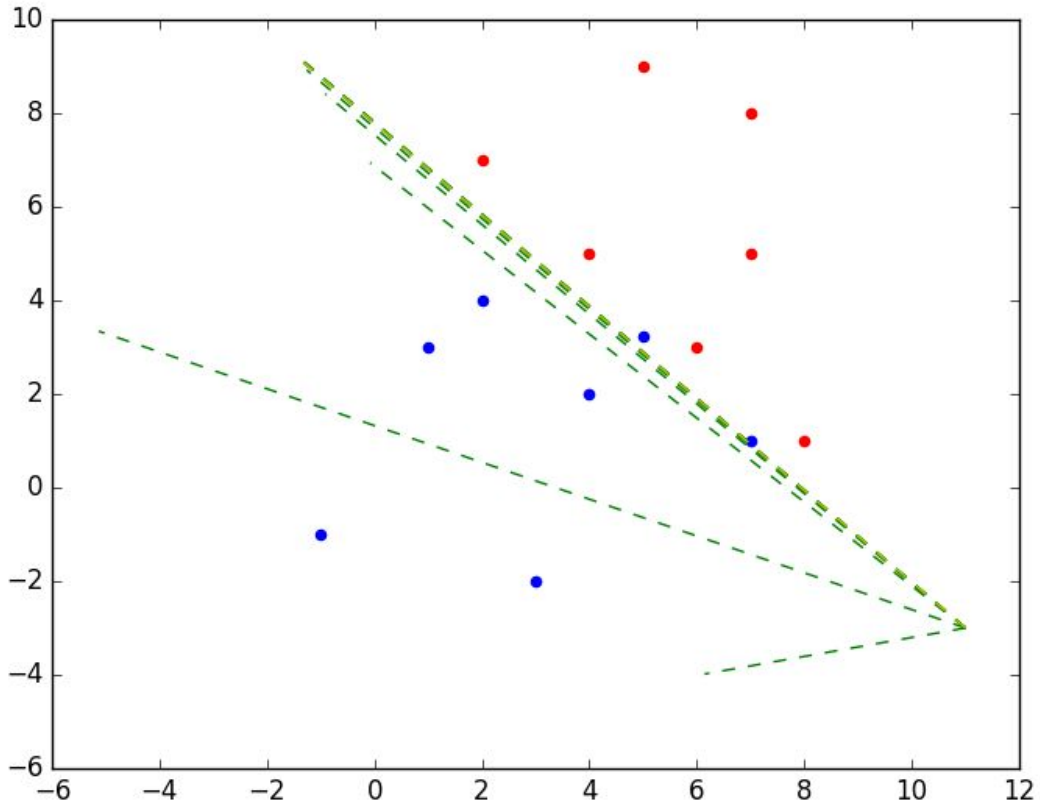

With different margins

---




With different margin

---




With margin(0.001)

---

# Relaxation algorithm  with margin

---

```
import numpy as np
import random
import matplotlib.pyplot as plt
import math
w1=[(2,7),(8,1),(7,5),(6,3),(7,8),(5,9),(4,5)]
w2=[(4,2),(-1,-1),(1,3),(3,-2),(5,3.25),(2,4),(7,1)]
Y_MAT=[[-1,-4,-2],[-1,1,1],[-1,-1,-3],[-1,-3,2],[-1,-5,-3.25],[-1,-2,-4],[-1,-7,-1],[1,2,7],
[1,8,1],[1,7,5],[1,6,3],[1,7,8],[1,5,9],[1,4,5]]
#print Y_MAT
b=0.001#margin
A_MAT=np.array(Y_MAT)
WEI_MAT=[1,-1,-1]
Str_WEI=[]#for storing weight matrix
WEI_MAT=np.array(WEI_MAT)
```

```
loop=8000000
while loop:
	DotMat=np.dot(A_MAT,WEI_MAT)
	#print DotMat,'This is dot product------'
	Dot_Mat=A_MAT[DotMat<b]
	#print Dot_Mat,'This is values of y========'
	p=random.randint(0,len(Dot_Mat)-1)

divisor=float(math.pow(math.pow(Dot_Mat[p][0],2)+math.pow(Dot_Mat[p][1],2)+math.pow(Dot_Mat[
p][2],2),0.5))
	#print divisor
	#print Dot_Mat[p]
	if len(Dot_Mat):
	dotproduct=np.dot(Dot_Mat[p],WEI_MAT)
	x=0.01*math.pow(((dotproduct)-b),2)/divisor
	#print x
	Dot_Mat[p]=Dot_Mat[p]*x
	WEI_MAT=WEI_MAT+Dot_Mat[p]
	if int(loop%1000000)==0:
		Str_WEI.append(WEI_MAT)
	else:
	break
	loop -=1
Line_Mat_pt=[]#for storing weight matrix lines
x1=[]
y1=[]
x2=[]
y2=[]
for i in w1:
	x1.append(i[0])
	y1.append(i[1])
for i in w2:
	x2.append(i[0])
	y2.append(i[1])
_p1x=float(11)
_p2x=float(-3)
_p1y=float(_p1x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))
_p2y=float(_p2x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))

for i in Str_WEI:
	if i[2]!=0:
	_L1x=float(11)
	_L2x=float(-3)
	_L1y=float(_p1x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
	_L2y=float(_p2x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
	Line_Mat_pt.append([[_L1x,_L1y],[_L2x,_L2y]])

for i in Line_Mat_pt:
	plt.plot(i[0],i[1],'g--')

plt.scatter(x1,y1,color='r')
plt.scatter(x2,y2,color='b')
plt.plot([_p1x,_p1y],[_p2x,_p2y],'y--')
plt.show()
print WEI_MAT,loop
```

- **If given margin less the algorithm took less iteration to terminate whereas in case of margin given high it takes more iteration to converge**

---

# 1Qd:
# Widrow-hoff Or Least mean square(LMS) rule

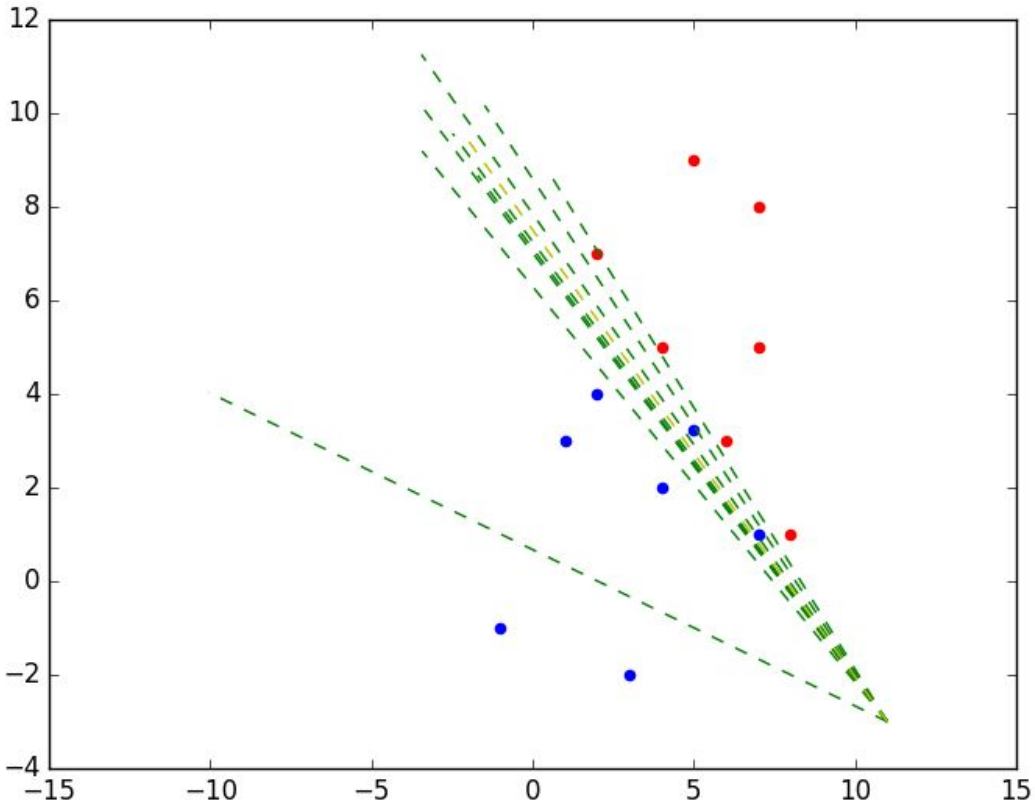
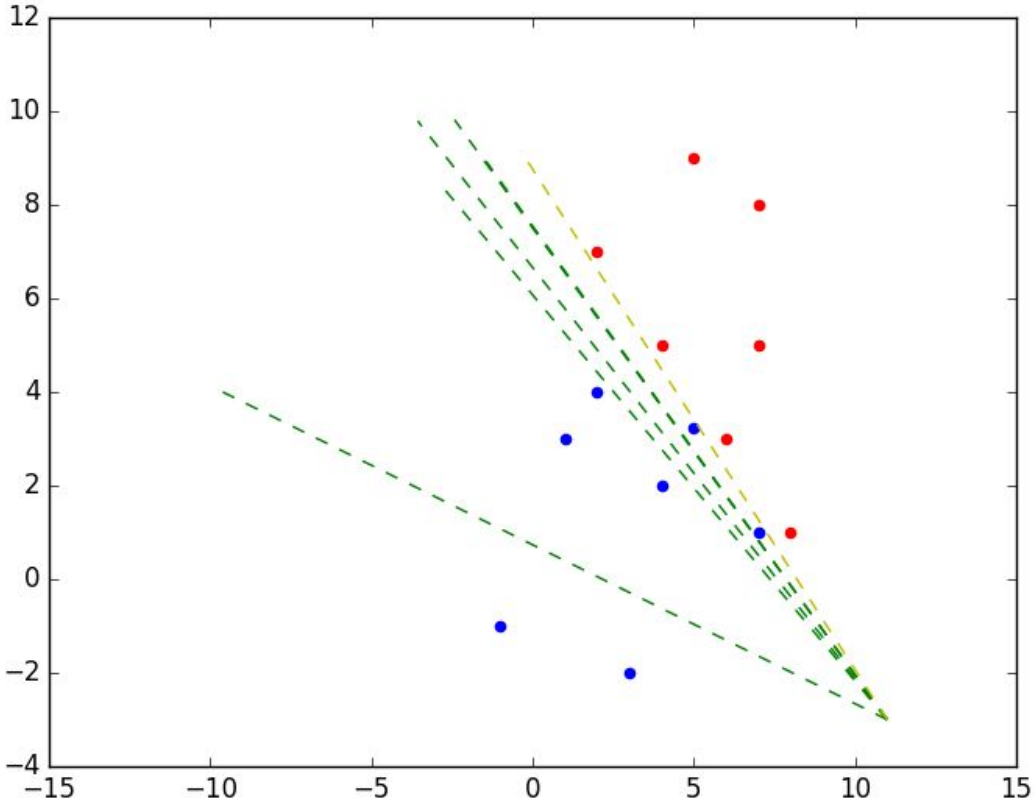



Fig -1Qd_1
w1(class)-red dot, w2(class)-blue dot
green lines(Dotted) -learning weights
Yellow line(Dotted)-final weights

---




**With different eta,theta values**

---

# Widrow-hoff Or Least mean square(LMS) rule Algorithm

---

```
import numpy as np
import random
import matplotlib.pyplot as plt
import math
w1=[(2,7),(8,1),(7,5),(6,3),(7,8),(5,9),(4,5)]
```

```
w2=[(4,2),(-1,-1),(1,3),(3,-2),(5,3.25),(2,4),(7,1)]
Y_MAT=[[-1,-4,-2],[-1,1,1],[-1,-1,-3],[-1,-3,2],[-1,-5,-3.25],[-1,-2,-4],[-1,-7,-1],[1,2,7],
[1,8,1],[1,7,5],[1,6,3],[1,7,8],[1,5,9],[1,4,5]]
#print Y_MAT
b=2#margin
eta=0.001#converging rate
Str_WEI=[];#for storing weight matrix
A_MAT=np.array(Y_MAT);# class points
WEI_MAT=[1,-1,-1]#initial weight matrix
WEI_MAT=np.array(WEI_MAT)
loop=8000000
while loop:
        DotMat=np.dot(A_MAT,WEI_MAT)
        p=random.randint(0,len(A_MAT)-1)#selecting randomly
        dotproduct=np.dot(A_MAT[p],WEI_MAT)#calculating |aTy|
        x=eta*(b-(dotproduct))#calculating |N(b-aTy)|

mod_va=float(math.pow(math.pow(A_MAT[p][0],2)+math.pow(A_MAT[p][1],2)+math.pow(A_MAT[p][2],2
),0.5))#calculating |Y| value
        term=abs(x*mod_va)#calculatin terminating condition
        if float(term)>0.00000001:
        #print x
        Dot_Mat=A_MAT[p]*x
        WEI_MAT=WEI_MAT+Dot_Mat#adding change to weight matrix
        eta=eta#/float(2)
        if int(loop%1000000)==0:
                Str_WEI.append(WEI_MAT)#storing  weight vector for ploting change
        else:
        break
        loop -=1
#print Str_WEI
Line_Mat_pt=[]#for storing weight matrix linesx1=[]
x1=[]
y1=[]
x2=[]
y2=[]
for i in w1:
        x1.append(i[0])
        y1.append(i[1])
for i in w2:
        x2.append(i[0])
        y2.append(i[1])
_p1x=float(11)
_p2x=float(-3)
_p1y=float(_p1x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))
_p2y=float(_p2x*(float(-WEI_MAT[1])/WEI_MAT[2])+(float(-WEI_MAT[0])/WEI_MAT[2]))
for i in Str_WEI:
        if i[2]!=0:
        _L1x=float(11)
        _L2x=float(-3)
        _L1y=float(_p1x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
        _L2y=float(_p2x*(float(-i[1])/i[2])+(float(-i[0])/i[2]))
        Line_Mat_pt.append([[_L1x,_L1y],[_L2x,_L2y]])

for i in Line_Mat_pt:
        plt.plot(i[0],i[1],'g--')
plt.scatter(x1,y1,color='r')
plt.scatter(x2,y2,color='b')
plt.plot([_p1x,_p1y],[_p2x,_p2y],'y--')
plt.show()
```

```
print WEI_MAT,loop
```

- **With low eta value algorithm took more cycles with to converge but with high eta values the algorithm doesn’t converge and terminates after specified iterations**
- **With low theta values algorithm took more iterations to terminate but produced good results whereas if theta value given high algorithm terminates fastly but produces compromised results**