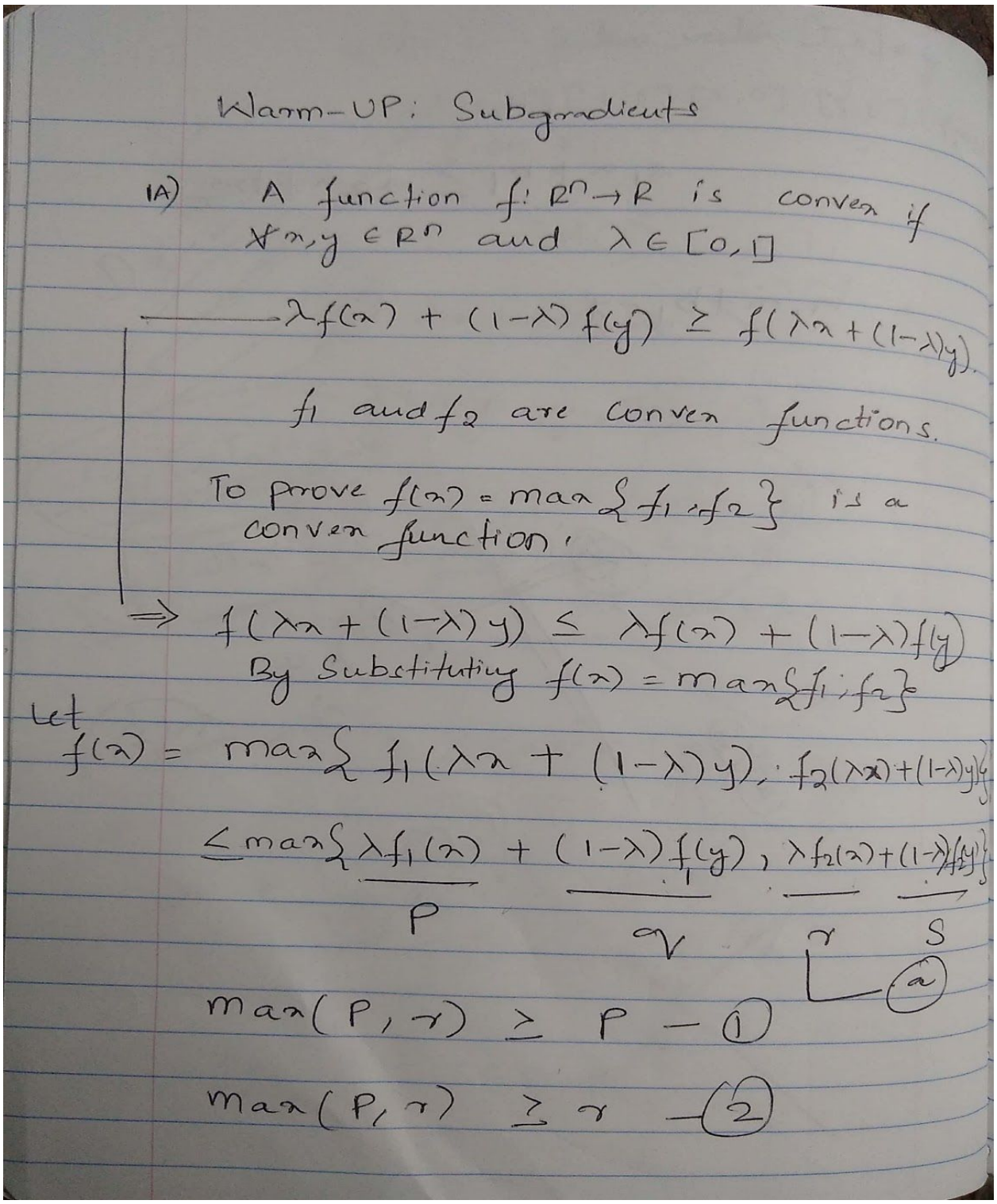
MACHINE LEARNING ASSIGNMENT

Warm-Up: Subgradients & More

1.



$\max(q, s) \geq q$ — (3)

$\max(q, s) \geq s$ — (4)

Adding 1 & 3 equations

$$p + q \leq \max(p, r) + \max(q, s)$$ — (5)

adding (2) & (4)

$$r + s \leq \max(p, r) + \max(q, s)$$ — (6)

By 5 & 6

$$\Rightarrow \max(p+q, r+s) \leq \max(p, r) + \max(q, s)$$

Substituting the values of p, q, r & s

$$\max\{\lambda f_1(x) + (1-\lambda) f_1(y),\ \lambda f_2(x) + (1-\lambda) f_2(y)\}$$

$$\leq \max\{\lambda f_1(x), \lambda f_2(x)\} + \max\{(1-\lambda) f_1(y), (1-\lambda) f_2(y)\}$$

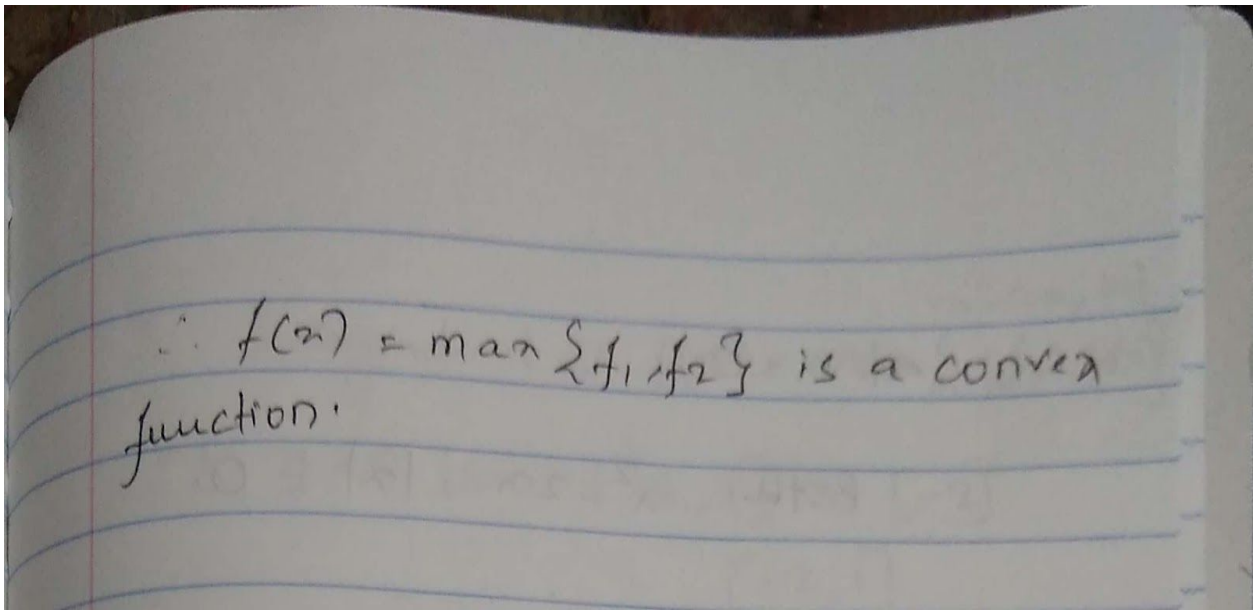
$$\leq \lambda \max\{f_1(x), f_2 x\} + (1-\lambda) \max\{f_1(y), f_2(y)\}$$

$$\leq \lambda f(x) + (1-\lambda) f(y) \quad \text{— (b)}$$

By (a) & (b) equations

$$f(x) = f(\lambda x + (1-\lambda) y) \leq \lambda f(x) + (1-\lambda) f(y)$$

$\therefore f(x) = \max\{f_1, f_2\}$ satisfies the property of convex function.

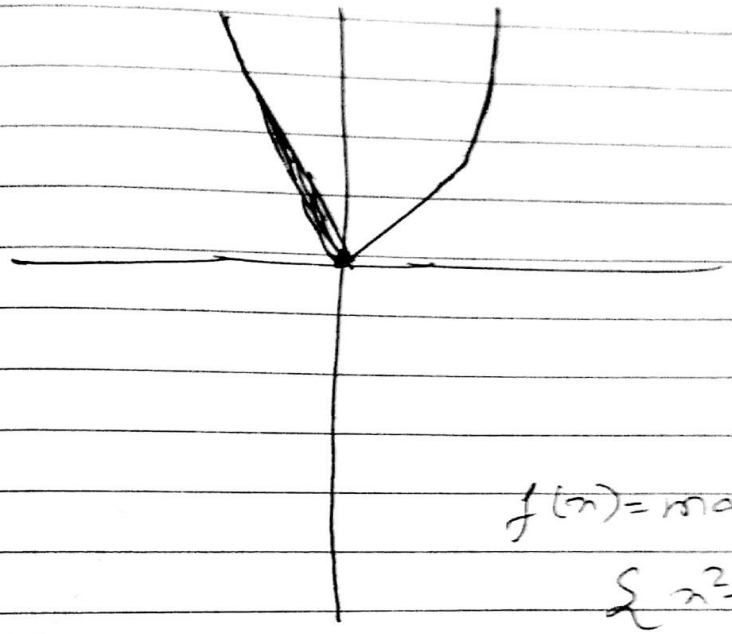


Warmup problem 2 next page

2.

2A) a) $f(x) = \max\{x^2 - 2x, |x|\}$ at $x = 0$

The graph of $f(x)$ is



$f(x) = \max\{x^2 - 2x, |x|\}$

at $x=0$,

Both $x^2-2x = |x| = 0$.

∴

Subgradient of both eqns should be taken.

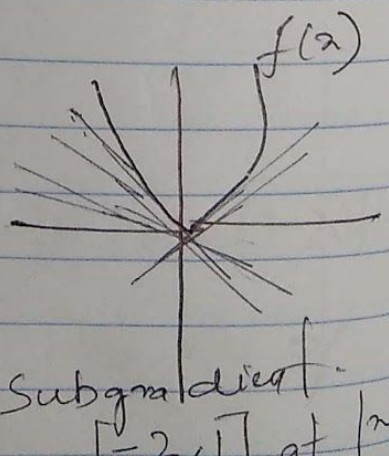
Subgradient of x^2-2x at $x=0$. w.r.to $f(x)$

$$\frac{\partial(x^2-2x)}{\partial x} = 2x-2$$

$$= 2(0)-2 = -2.$$

Subgradient of $|x|$ at $x=0$ w.r.to is given by

$$[-2, 1]$$



Subgradient $[-2,1]$ at $|x|$

$\therefore$ Subgradient of $f(x) = \max\{x^2-2x, |x|\}$

at $x=0$ is $[-2,1] \cup [-2]$

$= [-2,1]$

Subgradient of $f(x)$ at $x=-2$

At $x=-2$

$x^2-2x > |x|$

$\therefore \left.\frac{\partial f(x)}{\partial x}\right|_{(x=-2)} = \frac{\partial(x^2-2x)}{\partial x} = 2x-2$

$= 2(-2)-2 = -6$

Subgradient at $x=-2$ is -6

(b) $g(x) = \max\{(x-1)^2, (x-2)^2\}$
at $x=1.5$

At $x = 1.5$

$$(x-1)^2 = (x-2)^2 = 0.25$$

∴ Subgradients of $(x-1)^2$ and $(x-2)^2$ should be taken.

$$\frac{\partial((x-1)^2)}{\partial x}_{x=1.5} = 2(x-1)$$

$$= 2(1.5-1)$$

$$= 2 \times 0.5$$

$$= 1$$

$$\frac{\partial((x-2)^2)}{\partial x}_{x=1.5} = 2(x-2)$$

$$= 2(1.5-2)$$

$$= -2 \times 0.5$$

$$= -1$$

∴ Subgradient of $g(x)$ at $x = 1.5$ is $[-1] \cup [1]$

Subgradient of $g(x)$ at $x=0$.

At $x=0$ $(x-2)^2$ is maximum

$\Downarrow$ $(x-2)^2 > (x-1)^2$

$$\frac{\partial(g(x))}{\partial x} = \frac{\partial((x-2)^2)}{\partial x}_{x=0} = 2(x-2)$$

$$= 2(0-2) = -4$$

Subgradient of $g(x)$ at $x=0$ is -4

Problem 1: Perceptron Learning

1.

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import pandas as pd
data = pd.read_csv('perceptron.data',header = None)
x = data[[0, 1, 2, 3]]

learningStep = 1

w = [0, 0, 0, 0]
b = 0
def predicted(x):
    wx = 0
    for i in range(len(x)):
        wx = wx + (w[i]*(x.iloc[i]))
    wx = wx + b
    return wx

def perceptronLoss():
    loss = 0
    for i in range(len(data)):
        f = predicted(x.iloc[i])
        z = -1*(f)*(data.iloc[i][4])
        loss = loss + max(0, z)
    return loss
```

```
def subGradient():
    lossw = [0, 0, 0, 0]
    lossb = 0
    for i in range(len(data)):
        z = -(data.iloc[i][4])*predicted(x.iloc[i])
        if(z >= 0):
            lossw = lossw + data.iloc[i][4]*(x.iloc[i])
            lossb = lossb + data.iloc[i][4]
    return (lossw, lossb)

iterations = 10
it= 1
print("Weight and Bias for the first 3 iterations are ")
while(True):
    learningStep = 1/(1+it);
    Deltaw, Deltab = subGradient()
    #print("Deltaw ",Deltaw)
    #print("Deltab ",Deltab)
    w = w + learningStep*Deltaw
    b = b + learningStep*Deltab
    pLoss = perceptronLoss()
    print(pLoss)
    if(it <= 3):
        print("Weight\n",w)
        print("bias\n",b)
     #stopping condition subgradient should be zer0
    if(all(v == 0 for v in Deltaw) and Deltab==0):
        break
    print(it)
    it = it+1
```

```
print("Total number of Iterations is ",it)
pLoss = perceptronLoss()
print("Loss ",pLoss)
print("Final weight ",w)
print("Final bias ",b)
```

Problem 1 : Perceptron Learning

1. perceptron algorithm

$$\min \sum_{i=1}^{n} \max\{0, -y_i \cdot (w^T x^{(i)} + b)\}$$

Initial values of $w^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ and $b^0 = 0$

λ (learning rate) $= 1$

Using standard gradient descent

The number of iterations that it takes to find a perfect classifier for the data is 47.

The values of w and b for 1st 3 iterations

1st iteration $w = \begin{bmatrix} 1278.99 \\ 460.06 \\ -108.55 \\ -1672.31 \end{bmatrix}$ $b = -354.0$

2nd iteration

$$w = \begin{bmatrix} 1307.29 \\ 432.74 \\ -27.55 \\ -1523.79 \end{bmatrix} \quad b = -493.0$$

3rd iteration

$$w = \begin{bmatrix} 1255.18 \\ 425.50 \\ 18.79 \\ -1434.66 \end{bmatrix} \quad b = -625.0$$

Final weight and bias

$$w = \begin{bmatrix} 685.79 \\ 243.89 \\ 8.24 \\ -797.62 \end{bmatrix} \quad b = -1485.0$$

2. Stochastic Gradient Descent

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
#stochastic
import pandas as pd

data = pd.read_csv('perceptron.data',header = None)




x = data[[0, 1, 2, 3]]

learningStep = 1
#[149.277140, 52.533473, 1.67163, -172.891940] w
#-322.0 b
w = [0, 0, 0, 0]
b = 0




def predicted(x):
    wx = 0
    for i in range(len(x)):
        wx = wx + (w[i]*(x.iloc[i]))
    wx = wx + b
    return wx
```

```
def perceptronLoss():
    loss = 0
    for i in range(len(data)):
        f = predicted(x.iloc[i])
        z = -1*(f)*(data.iloc[i][4])
        loss = loss + max(0, z)
    return loss


def subGradient(i):
    lossw = [0, 0, 0, 0]
    lossb = 0
    z = -(data.iloc[i][4])*predicted(x.iloc[i])
    if(z >= 0):
        lossw = lossw + data.iloc[i][4]*(x.iloc[i])
        lossb = lossb + data.iloc[i][4]
    return (lossw, lossb)




iterations = 1000
i = 0
k= 0

print("Weight and Bias for the first 3 iterations are ")
while(True):
#    print("i",i)
    Deltaw, Deltab = subGradient(i)
    w = w + learningStep*Deltaw
    b = b + learningStep*Deltab
```

```
    if(i <= 2 and k<1):
        print("Weight\n",w)
        print("bias\n",b)
    if(i%1000 == 0):
        pLoss = perceptronLoss()
        #print("pLoss \t",pLoss)
        if(pLoss == 0):
            print(k)
            print(i)
            break
    i = i+1
    if(i == 1000):
        i=0
        k = k + 1
        print(k)
print("Total number of iterations is ",(k+1)*1000)
print("Loss ",pLoss)
print("Final weight ",w)
print("Final bias ",b)

"""i = 0
while(i<100):
    print(predicted(x.iloc[i]))
    i = i + 1"""
```

2. Stochastic gradient descent

Step size $\gamma_t = 1$

The number of iterations at that it takes to find a perfect classifier for the data is 1092×10^3 iterations.

The values of w and b for first 3 iterations are

1st iteration $w \begin{bmatrix} 4.617544 \\ 2.469679 \\ 1.967661 \\ -1.813356 \end{bmatrix}$ $b = -1.0$

2nd iteration $w \begin{bmatrix} 4.617544 \\ 2.469679 \\ 1.967661 \\ -1.813356 \end{bmatrix}$ $b = -1.0$

3rd iteration $w \begin{bmatrix} 3.453223 \\ 0.169435 \\ 2.628016 \\ -4.647099 \end{bmatrix}$ $b = -2.0$

Find weight and bias

$$w = \begin{bmatrix} 149.277140 \\ 52.533473 \\ 1.673673 \\ -172.891940 \end{bmatrix}$$

$$b = \begin{bmatrix} -322.0 \end{bmatrix}$$

3.

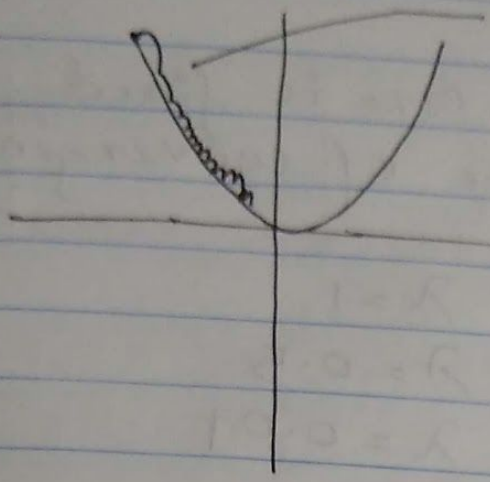
3. Rate of convergence is different based upon the step sizes taken as follow

1) Step size = $\frac{1}{1+c*t}$

c - constant // diminishing step size
t - iterations

Rate of convergence is very slow on diminishing step size.

3rd continued next page



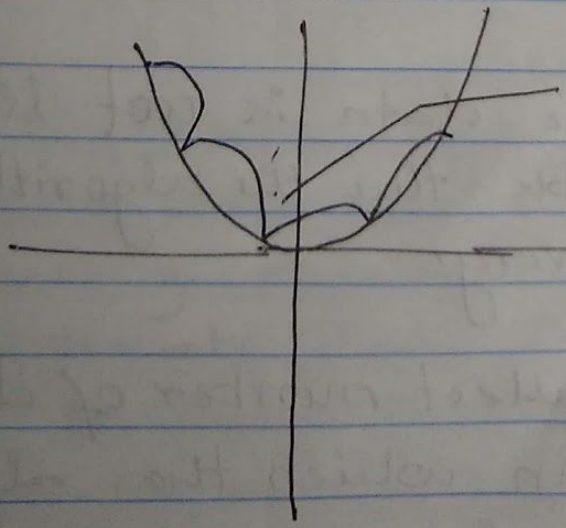
converging slowly

$Y = \frac{1}{1+ct}$

2) Step size = $\frac{1}{1+t}$

t — iterations

Rate of convergence will never come to an end as weight and bias will be jumping over minimum value.



convergence rate not coming to an end.

$y = \frac{1}{1+t}$

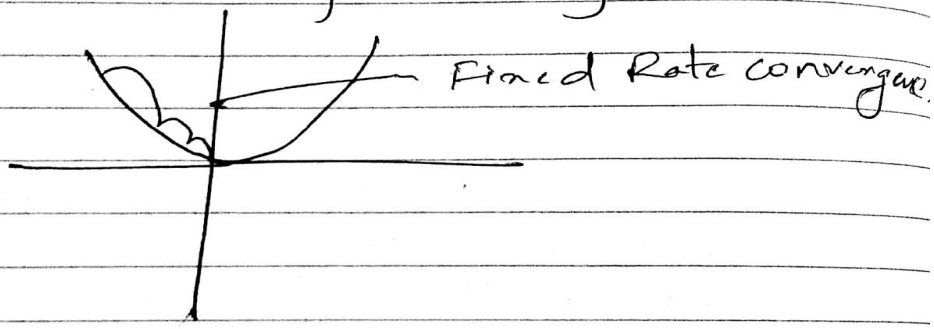
3) If step size is fixed and low then rate of convergence is fixed.

Ex :- $\lambda = 1$

$\lambda = 0.5$

$\lambda = 0.01$

Fixed Rate of Convergence.



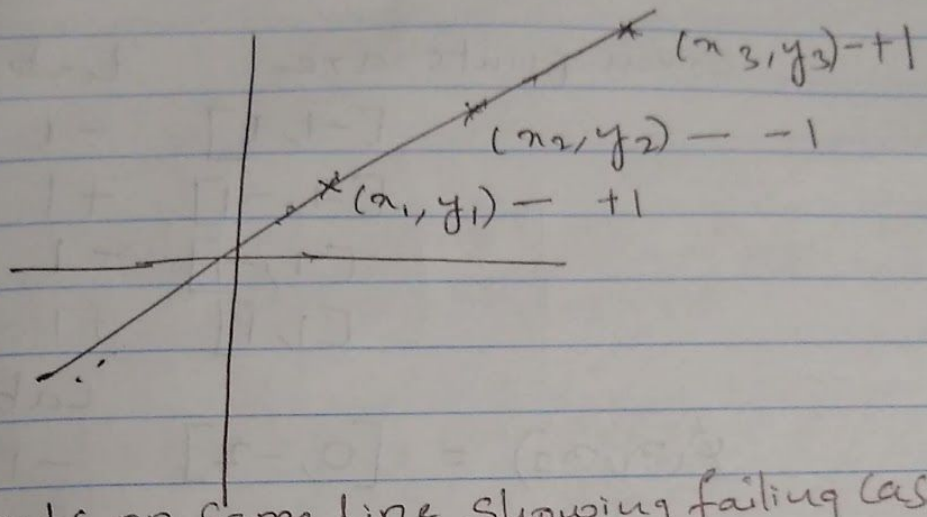
4.

4. If the data is not linearly separable then the algorithm fails to converge.

The smallest number of datapoints is 3 in which the algo fails to converge.

Example :-

consider 3 points on a line equation

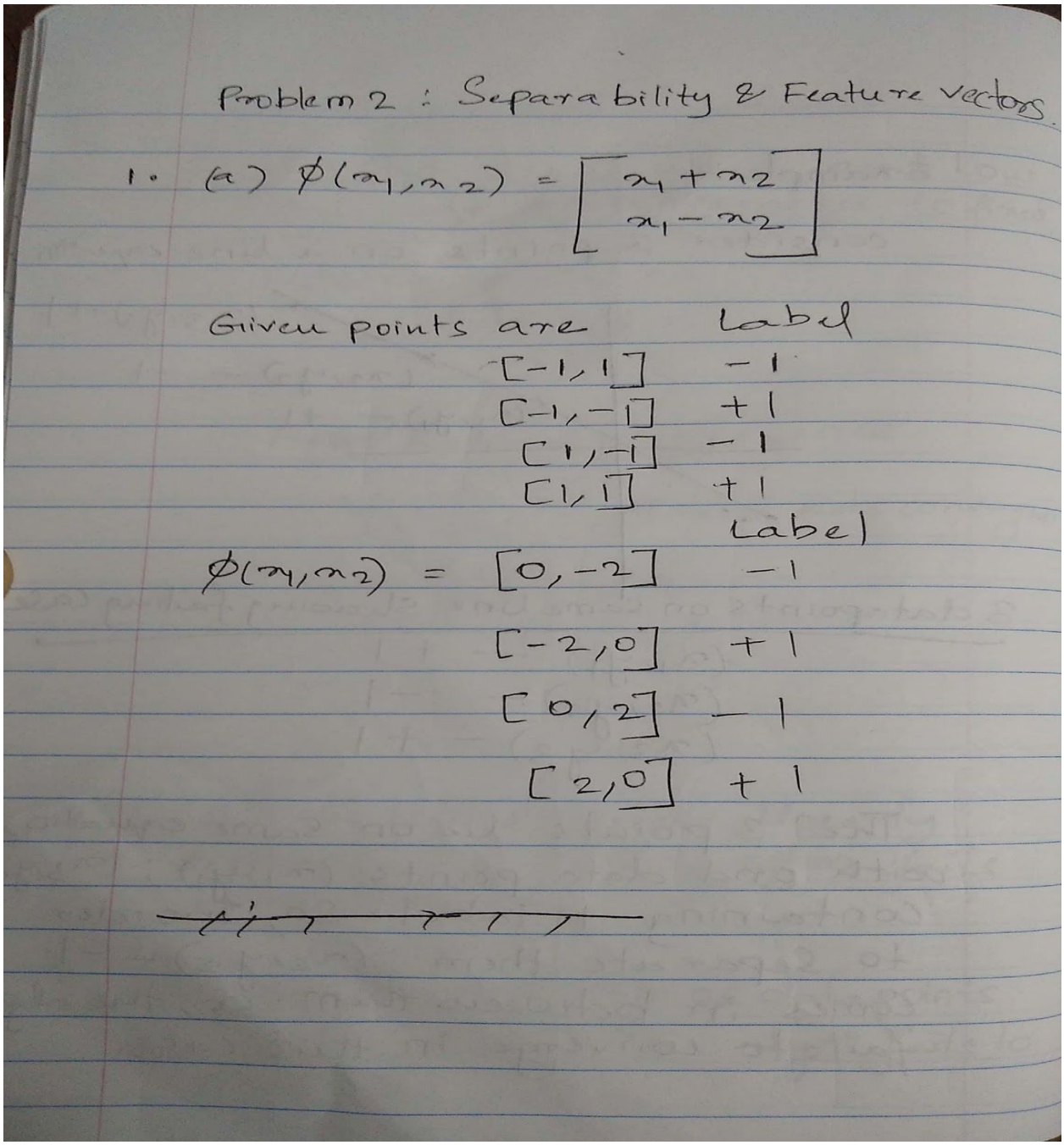


3 datapoints on same line showing failing case.

(x_1, y_1) - +1
(x_2, y_2) - -1
(x_3, y_3) - +1

The 3 points lie on same equation with end data points (x_1, y_1); (x_3, y_3) containing +1 label. So, Inorder to separate them (x_2, y_2) - -1 comes in between them. So, the algo fails to converge in this case.

Problem 2: Separability & Feature Vectors



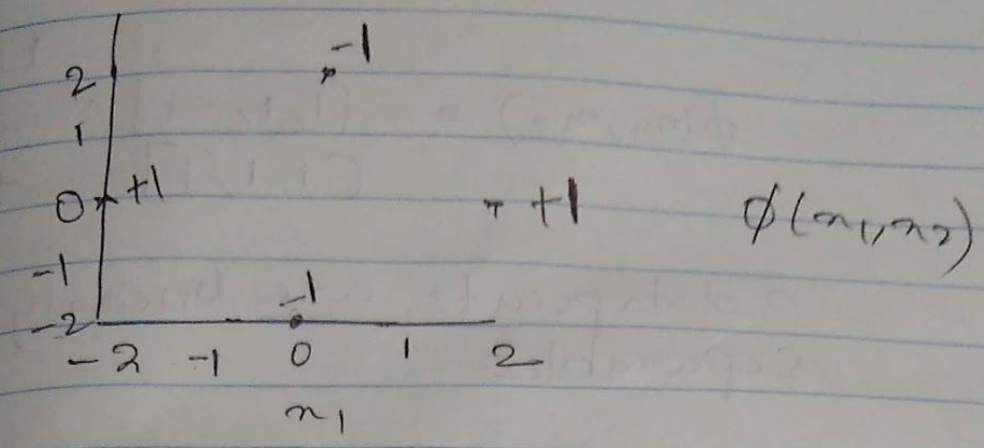
Problem 2 : Separability & Feature vectors.

1. (a) $\phi(x_1, x_2) = \begin{bmatrix} x_1 + x_2 \\ x_1 - x_2 \end{bmatrix}$

Given points are

	Label
[-1, 1]	-1
[-1, -1]	+1
[1, -1]	-1
[1, 1]	+1

$\phi(x_1, x_2)$ =	Label
[0, -2]	-1
[-2, 0]	+1
[0, 2]	-1
[2, 0]	+1



The above graph depicts they cannot be linearly Separable.

Sub Substituting the datapoints of $\phi(x_1, x_2)$ in the standard gradient descent problem, the algo is failing to converge. So it is not linearly separable.

(b) $\phi(x_1, x_2) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$

$\phi(x_1, x_2) =$	Label
[1, 1, -1]	-1
[1, 1, 1]	+1
[1, 1, -1]	-1
[1, 1, 1]	+1

$\phi(x_1, x_2)$ =		Label
	$[1, 1, -1]$	-1
	$[1, 1, 1]$	$+1$

2 datapoints are linearly separable.

(weight) $w = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$ (bias) $b = 0$

For $\phi(x_1, x_2) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$, the data is linearly separable.

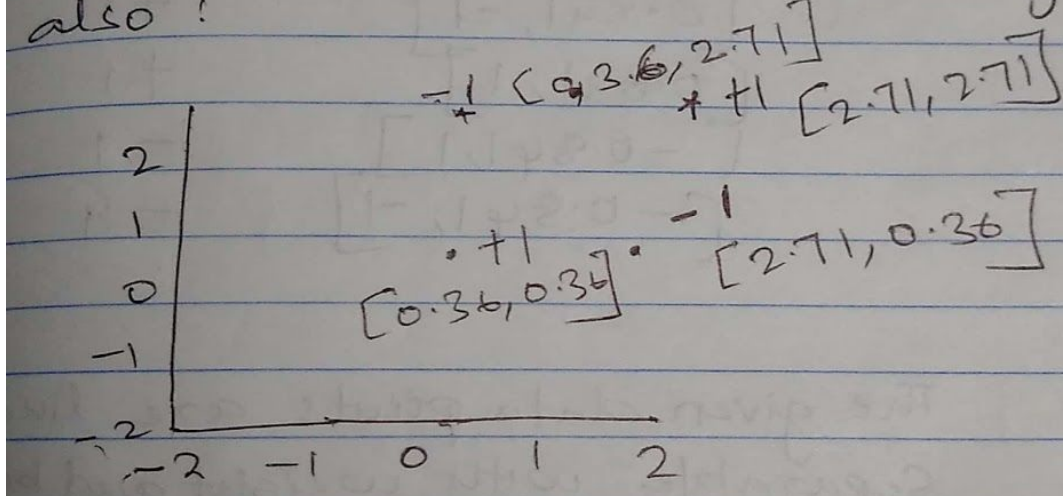
(c) $\phi(x_1, x_2) = \begin{bmatrix} \exp(x_1) \\ \exp(x_2) \end{bmatrix}$

Datapoints

$\phi(x_1, x_2)$ =		
	$[0.36, 2.71]$	-1
	$[0.36, 0.36]$	$+1$
	$[2.71, 0.36]$	-1
	$[2.71, 2.71]$	$+1$

Substituting the datapoints in the dataset and applying the standard gradient descent algo, the algo fails to converge.

The failure can be depicted from graph also:



Hence by program and graph, it can be seen that it is not linearly separable.

(d) $\phi(x_1, x_2) = \begin{bmatrix} x_1 \cdot \sin(x_2) \\ x_1 \end{bmatrix}$

datapoints of $\phi(x_1, x_2)$

	Label
$[0.841, -1]$	+1
$[0.841, 1]$	+1
$[-0.841, 1]$	−1
$[-0.841, -1]$	−1

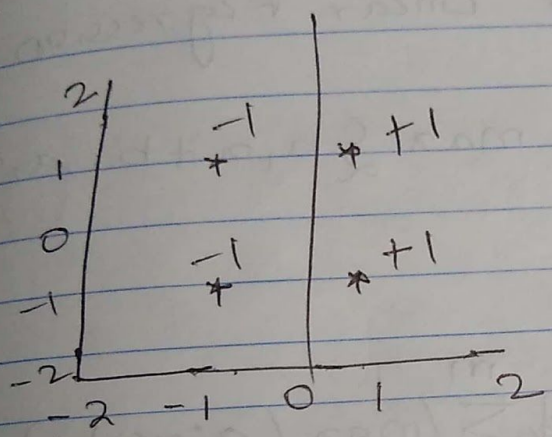
The given data points are linearly separable with weight and bias as follows

weight w $\begin{bmatrix} 3.364 \\ 0 \end{bmatrix}$

bias $b = 0.0$

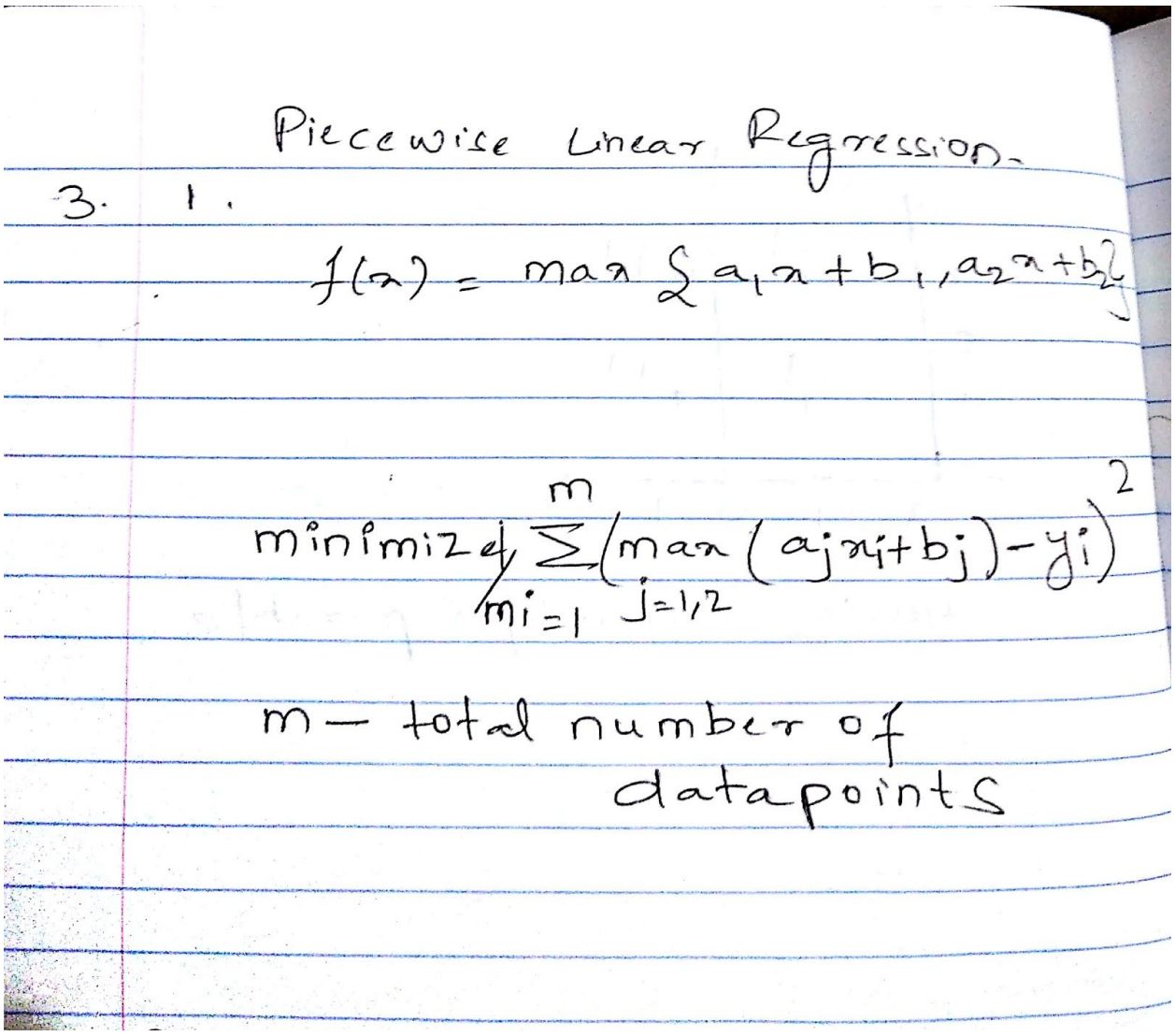
also

It can be depicted by graph as follows



Hence, Linearly Separable.

Problem 3: Piecewise Linear Regression



Piecewise Linear Regression.

3. 1.

$$f(x) = \max\{a_1 x + b_1, a_2 x + b_2\}$$

$$\text{minimize} \sum_{i=1}^{m} \left(\max_{j=1,2}(a_j x_i + b_j) - y_i\right)^2$$

m — total number of datapoints

2)

2. $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \end{bmatrix}$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{m} \sum_{i=1}^{m} 2(\max_{j=1,2}(a_j x_i + b_j) - y_i) x_i \\ \frac{1}{m} \sum_{i=1}^{m} 2(\max_{j=1,2}(a_j x_i + b_j) - y_i) \end{bmatrix}$$

Using Subgradient descent to minimize loss function.

If $a_1 x + b_1 \geq a_2 x + b_2$

γ - step size

$$a_{1(t+1)} = a_{1(t)} - \gamma\left(\frac{2}{m} \sum_{i=1}^{m} ((a_1 x_i + b_1) - y_i) x_i\right)$$

$$b_{1(t+1)} = b_{1(t)} - \gamma\left(\frac{2}{m} \sum_{i=1}^{m} (a_1 x_i + b_1 - y_i)\right)$$

If $a_2x + b_2 > a_1x + b_1$

$a_{2(t+1)} = a_{2(t)} - \gamma\left(\frac{2}{m}\sum_{i=1}^{m}((a_2x_i + b_2) - y_i)\cdot x_i\right)$

$b_{2(t+1)} = b_{2(t)} - \gamma\left(\frac{2}{m}\sum_{i=1}^{m}(a_2x_i + b_2 - y_i)\right)$

usuage of Subgradients

3.

3. optimization problem

$$\min \sum_{i=1}^{m} \left(\max_{j=1,2} (a_j x_i + b_j) - y_i \right)^2$$

yes, the above optimization problem is convex

It is convex because the optimization problem turns out to be

$$\min \sum_{i=1}^{m} \left(\max_{j=1,2} (a_j x_i + b_j)^2 \right) \quad \ldots\ldots$$

This function is convex,

Hence the optimization problem is convex.

Problem 4: Support Vector Machines

Feature vector is given by

15 dimensional feature vector

[x0**3, x1**3, x2**3, x3**3, x0** 2, x1 ** 2, x2 ** 2, x3 ** 2, x0 * x1, x1 * x2, x2 * x3, x0, x1, x2, x3]

Weight [32.9956879, 36.96199139, -74.79636297, 50.74497417 7.93749232, -75.21478862, -55.25938636, -71.42179087, 279.09873519 46.33505831, 6.97734404, 115.5689007, 33.27909418, -31.86586959 ,22.52862852]

bias 0.9278756602700334

Margin 0.002879826974815148

The 3 Support Vectors are

1. [0.03115492 0.00113515 0.22933492 0.0024597 0.09901123 0.01088183
0.37466817 0.01822168 0.03282412 0.06385197 0.08262617 0.3146605
0.10431599 0.61210144 0.13498771 -1.]

2. [0.02368381 0.02675403 0.36206655 0.65426046 0.08247096 0.08945256
0.50799441 0.75364551 0.08589085 0.21316989 0.61874688 0.28717757
0.29908621 0.71273727 0.86812759 -1.]

3. [4.48759156e-05 5.11540543e-05 1.55201337e-03 4.60944298e-01
1.26282222e-03 1.37801319e-03 1.34049019e-02 5.96713181e-01
1.31916097e-03 4.29792177e-03 8.94364671e-02 3.55362100e-02
3.71216000e-02 1.15779540e-01 7.72472123e-01 1.00000000e+00]

Program for SVM in next page

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Sep  9 14:06:28 2018

@author: sahith
"""
import pandas as pd

import numpy as np
import quadprog as quad

 ##Quadratic programming Reference as posted in piazza but using
quadprog
 #package instead of CVXOPT
 ## https://courses.csail.mit.edu/6.867/wiki/images/a/a7/Qp-cvxopt.pdf

data = pd.read_csv('mystery.data',header = None)
rows = len(data)
G = np.zeros((rows,16))



def Solve(P, q, G=None, h=None):  #Quad Prog Solving
    return quad.solve_qp(.5 * P, q, -G.T,  -h, 0)[0]



 # Feature vector [x0**3, x1**3,x2**3,x3**3,x0 ** 2,x1 ** 2,x2 ** 2,x3 ** 2,x0
* x1,x1 * x2,x2 * x3,x0,x1,x2,x3]
def features(train_data, rows):
    featuredData = np.empty((1000,16), dtype=float)
    for k in range(rows):
        x0 = data.iloc[k, 0];
        x1 = data.iloc[k, 1];
```

```
x2 = data.iloc[k, 2];
x3 = data.iloc[k, 3];
y = data.iloc[k, 4]
featuredData[k, 0] = x0 ** 3
featuredData[k, 1] = x1 ** 3
featuredData[k, 2] = x2 ** 3
featuredData[k, 3] = x3 ** 3
featuredData[k, 4] = x0 ** 2
featuredData[k, 5] = x1 ** 2
featuredData[k, 6] = x2 ** 2
featuredData[k, 7] = x3 ** 2
featuredData[k, 8] = x0 * x1
featuredData[k, 9] = x1 * x2
featuredData[k, 10] = x2 * x3
featuredData[k, 11] = x0
featuredData[k, 12] = x1
featuredData[k, 13] = x2
featuredData[k, 14] = x3
featuredData[k, 15] = y
G[k, 0] = -featuredData[k, 0] * y
G[k, 1] = -featuredData[k, 1] * y
G[k, 2] = -featuredData[k, 2] * y
G[k, 3] = -featuredData[k, 3] * y
G[k, 4] = -featuredData[k, 4] * y
G[k, 5] = -featuredData[k, 5] * y
G[k, 6] = -featuredData[k, 6] * y
G[k, 7] = -featuredData[k, 7] * y
G[k, 8] = -featuredData[k, 8] * y
G[k, 9] = -featuredData[k, 9] * y
G[k, 10] = -featuredData[k, 10] * y
G[k, 11] = -featuredData[k, 11] * y
G[k, 12] = -featuredData[k, 12] * y
G[k, 13] = -featuredData[k, 13] * y
```

```
        G[k, 14] = -featuredData[k, 14] * y
        G[k, 15] = -y

    return featuredData, G

data, G = features(data, rows)
#print(data[0,1])

P = np.zeros((16,16))
for i in range(16):
    P[i, i] = 1;
q = np.zeros((16,1)).reshape((16,))
h = -np.ones((rows,1)).reshape((len(data),))
#print(P)




W = Solve(P,q,G,h)   #Quadratic prog solve

print("Weight ", W[:15])  #Weight
print("bias ", W[15])   #bias

print("Margin ",  (1 / np.sqrt((W[0:15] ** 2).sum())))   #Margin = 1/||W||


i = 0
```

```
#Test
supportVectors = []
predicted = []
while(i < 1000):
    j = 0
    k = 0
    while(j < 15):
        k = k + W[j]*data[i, j]
        j = j+ 1
    k = k+W[15]
    if(k > 0 and k <= 1.00000000000002):
        supportVectors.append(data[i])
        predicted.append(k)
    elif(int(k) == -1):
        predicted.append(k)
        supportVectors.append(data[i])
    #print("predicted ",k," actual ", data[i, j])
    i = i+1

#print(supportVectors)
#print(predicted)
supportVectorLength = len(supportVectors)
i = 0
print("\nThe Support Vectors are ")
while(i < supportVectorLength):
    print(supportVectors[i],"\n")
    i += 1
```

----------END----------