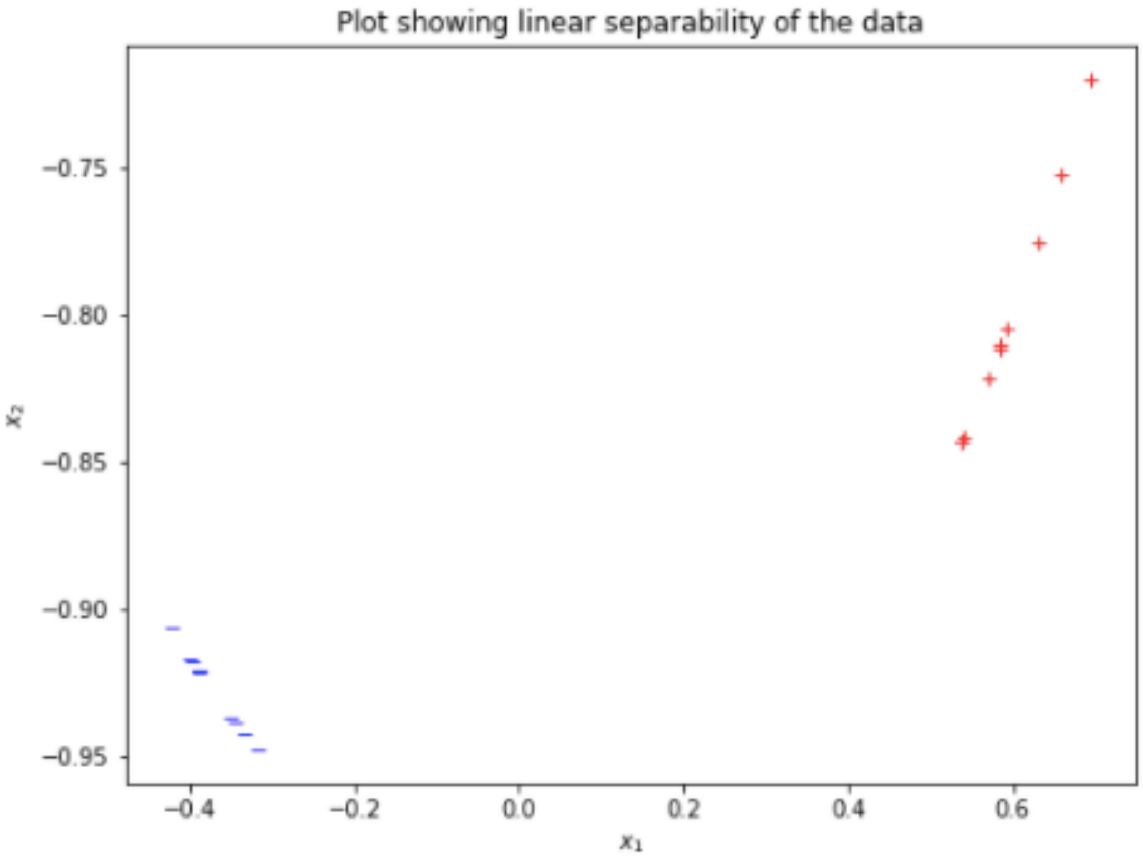
Akankshita Dash
A0132788U
T03

MA4270 Computational Exercise

Problem 1- Perceptron

1.



2.

```
Problem 1 Q2
==================================
theta^*: [ 0.99257345  0.12164684]
gamma^*: 0.431484598217
```

3. a.

```
Problem 1 Q3(a)
==================================
Number of updates, k = 2
theta = [ 0.98365688  0.18005314]
gamma = 0.377454883747
```

b.

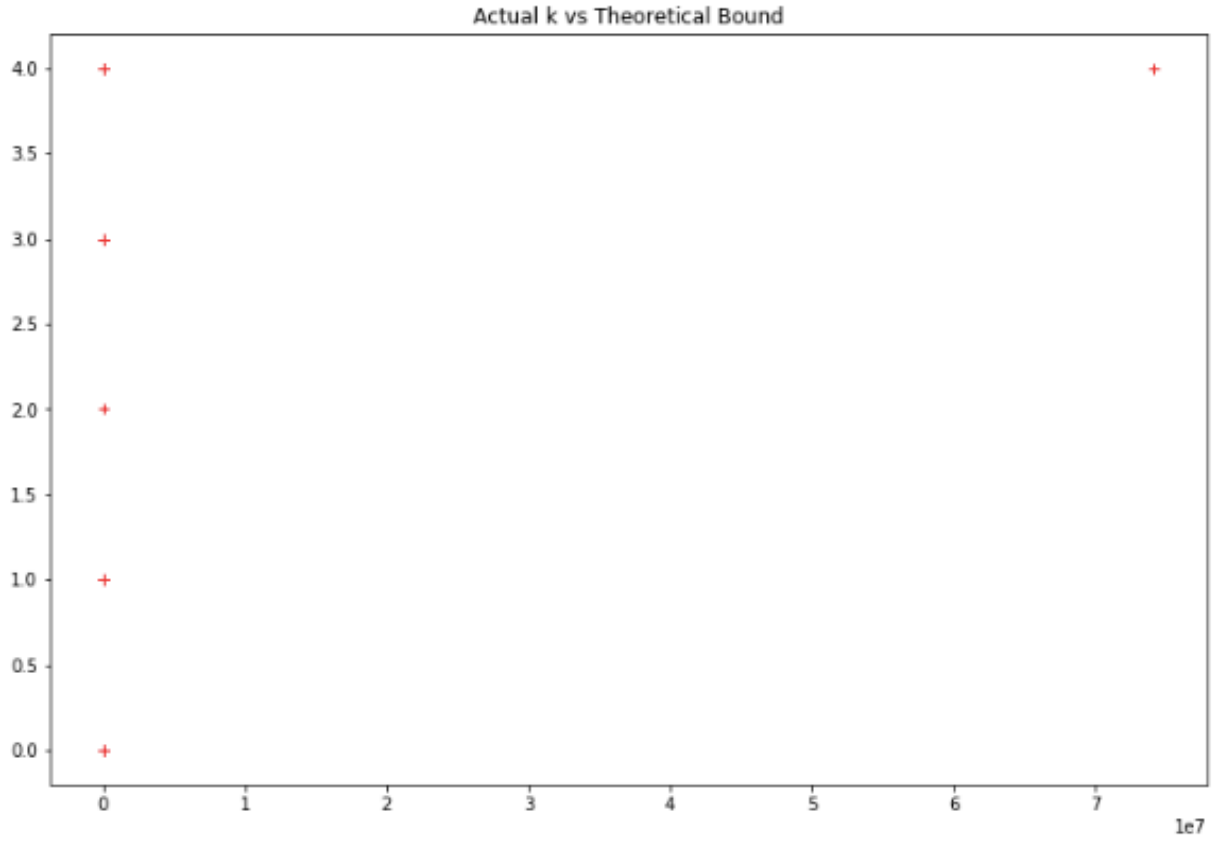
```
==================Iter 1=================
theta_zero = [ 0.74794344  0.79374993]
k = 1, theta = [1.000,0.029], gamma=0.346
==================Iter 2=================
theta_zero = [ 0.12825667  0.29218249]
k = 2, theta = [0.922,0.387], gamma=0.170
==================Iter 3=================
theta_zero = [ 0.77883043  0.0649501 ]
k = 0, theta = [0.997,0.083], gamma=0.396
```

```
==================Iter 4==================
theta_zero = [ 0.18658537  0.55412612]
k = 1, theta = [0.974,-0.228], gamma=0.094
==================Iter 5==================
theta_zero = [ 0.65364266  0.00123201]
k = 0, theta = [1.000,0.002], gamma=0.320
==================Iter 6==================
theta_zero = [ 0.07679343  0.86728292]
k = 1, theta = [0.988,0.154], gamma=0.402
==================Iter 7==================
theta_zero = [ 0.73933756  0.65419106]
k = 1, theta = [0.998,-0.070], gamma=0.251
==================Iter 8==================
theta_zero = [ 0.76393958  0.16413424]
k = 0, theta = [0.978,0.210], gamma=0.349
==================Iter 9==================
theta_zero = [ 0.22481565  0.60812585]
k = 1, theta = [0.987,-0.161], gamma=0.162
==================Iter 10==================
theta_zero = [ 0.75852004  0.05519825]
k = 0, theta = [0.997,0.073], gamma=0.386
```

The number of updates k and parameter vectors obtained are different when we have different starting points. This is expected as the perceptron algorithm doesn't always find the same (best) separator.

c.

d.



The theoretical upper bound (x-axis, multiplied by 10^7) vs actual upper (y-axis) for first 100 runs

```
==================Iter 1==================
Sample point from unit circle = [-0.82021535  0.57205487]
k = 3.0, theta = [0.996,-0.086], gamma=0.236
==================Iter 2==================
Sample point from unit circle = [ 0.80547874 -0.59262467]
k = 1.0, theta = [0.958,0.286], gamma=0.275
==================Iter 3==================
Sample point from unit circle = [ 0.34508442 -0.93857165]
k = 1.0, theta = [1.000,-0.002], gamma=0.316
==================Iter 4==================
Sample point from unit circle = [ 0.7983284   0.60222236]
k = 1.0, theta = [0.984,-0.177], gamma=0.145
==================Iter 5==================
Sample point from unit circle = [-0.50022848 -0.86589345]
k = 3.0, theta = [0.983,0.182], gamma=0.375
==================Iter 6==================
Sample point from unit circle = [ 0.9963224   0.08568362]
k = 0.0, theta = [0.996,0.086], gamma=0.399
==================Iter 7==================
Sample point from unit circle = [-0.87314514  0.48746031]
k = 3.0, theta = [0.978,-0.208], gamma=0.115
==================Iter 8==================
Sample point from unit circle = [-0.8734605   0.48689501]
k = 3.0, theta = [0.978,-0.209], gamma=0.114
==================Iter 9==================
Sample point from unit circle = [ 0.14835785  0.98893374]
k = 1.0, theta = [0.960,0.281], gamma=0.279
==================Iter 10==================
Sample point from unit circle = [-0.89775206  0.44050111]
k = 3.0, theta = [0.961,-0.278], gamma=0.043
```

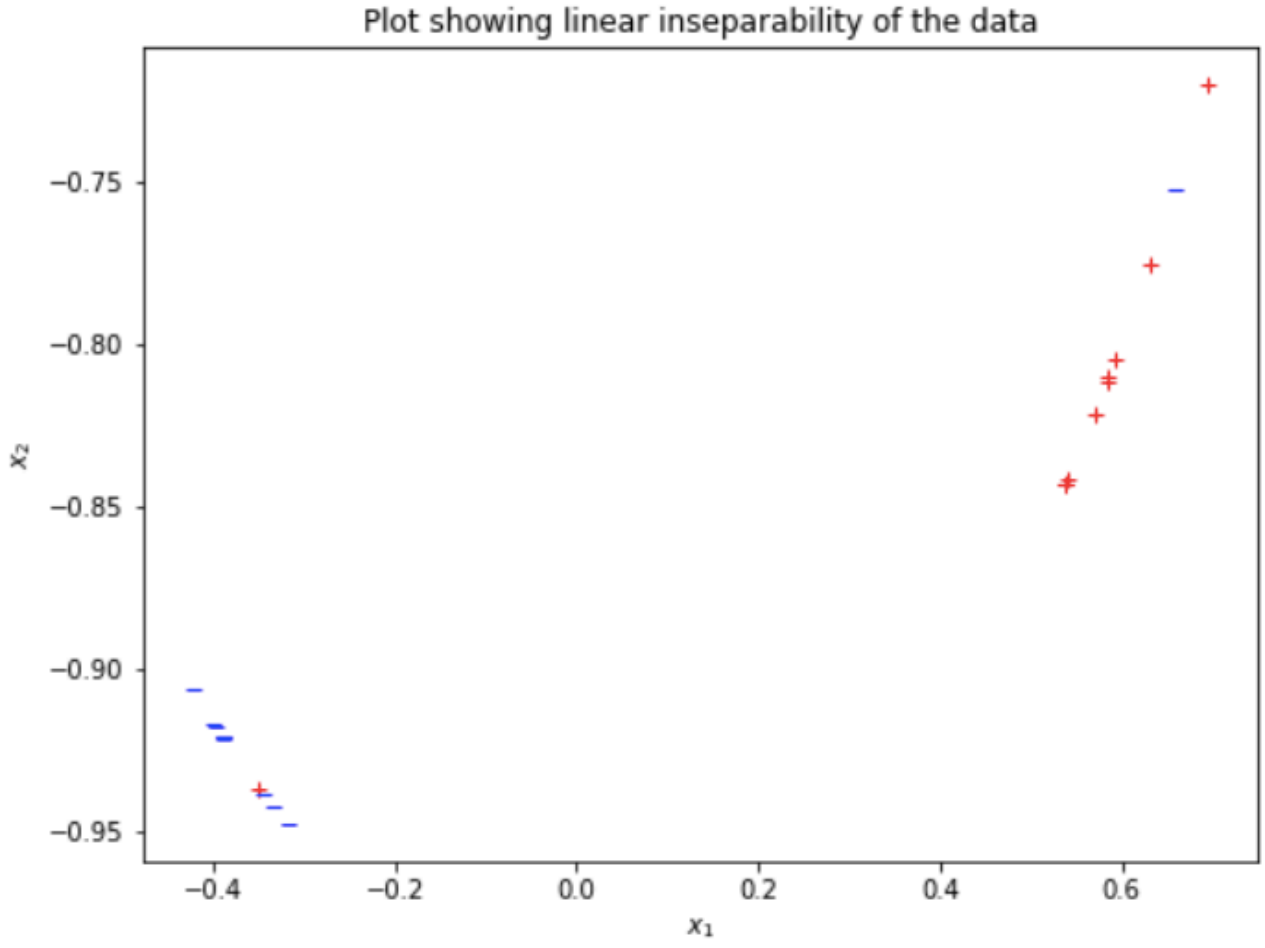
```
==================Iter 9995==================
Sample point from unit circle = [ 0.87111095  0.49108625]
k = 0.0, theta = [0.871,0.491], gamma=0.055
==================Iter 9996==================
Sample point from unit circle = [-0.55664833 -0.83074824]
k = 3.0, theta = [0.972,0.236], gamma=0.323
==================Iter 9997==================
Sample point from unit circle = [-0.62196517  0.78304491]
k = 3.0, theta = [0.991,0.134], gamma=0.421
==================Iter 9998==================
Sample point from unit circle = [-0.36178698 -0.93226079]
k = 3.0, theta = [0.996,0.091], gamma=0.403
==================Iter 9999==================
Sample point from unit circle = [-0.99840732 -0.05641647]
k = 4.0, theta = [0.978,0.208], gamma=0.351
==================Iter 10000==================
Sample point from unit circle = [-0.92891551 -0.37029174]
k = 4.0, theta = [0.993,-0.116], gamma=0.207
```

After 10,000 iterations

```
Average number of updates for 10,000 updates is: 1.8454
Average gamma for 10,000 updates is: 0.241698538882
```

The average number of updates is 1.8454, while the average γ is 0.2417, smaller than the γ^* = 0.4315 obtained in part 2.

4.



Visually, we can see that the data is not linearly separable. Mathematically, we can prove that the convex hull for the two sets intersects.

After running part 2 again

<u>For SVM without slack and offset</u>

```
---------------------------------------------------------
ValueError                                Traceback (most r
<ipython-input-20-635f96b36ab3> in <module>()
     20             G_new[i,j] = (-1*y[i]*x2[i])
     21
---> 22 theta_star_modified = quadprog_solve_qp(P_new, q_ne
     23 #theta_star_modified = solvers.qp(matrix(P_new), ma
(h_new))
     24

<ipython-input-16-29ad15ba491b> in quadprog_solve_qp(P, q,
     47         qp_b = -h
     48         meq = 0
---> 49     return quadprog.solve_qp(qp_G, qp_a, qp_C, qp_b
     50
     51 ################################################

quadprog/quadprog.pyx in quadprog.solve_qp()

ValueError: constraints are inconsistent, no solution
```

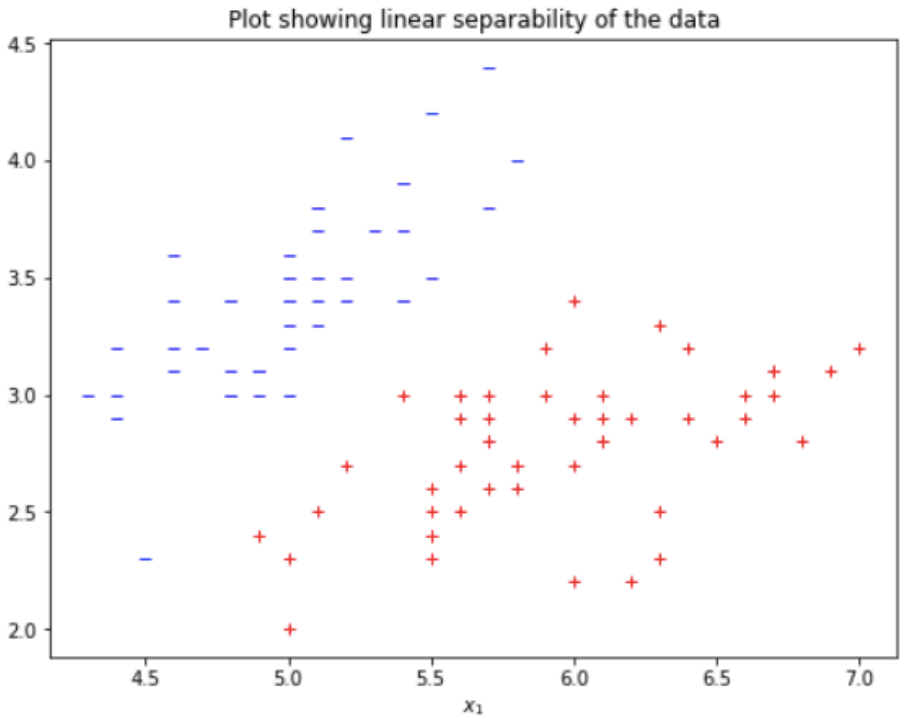
<u>For perceptron</u>

```
Max iterations exceeded; data is non-separable!
Iterations taken to classify current non-separable data: 1003
```

Thus, SVM doesn't provide a feasible solution while the perceptron algorithm doesn't converge (in the code, it is terminated after a certain number of max_iterations are exceeded)

Problem 2- Hard-SVM

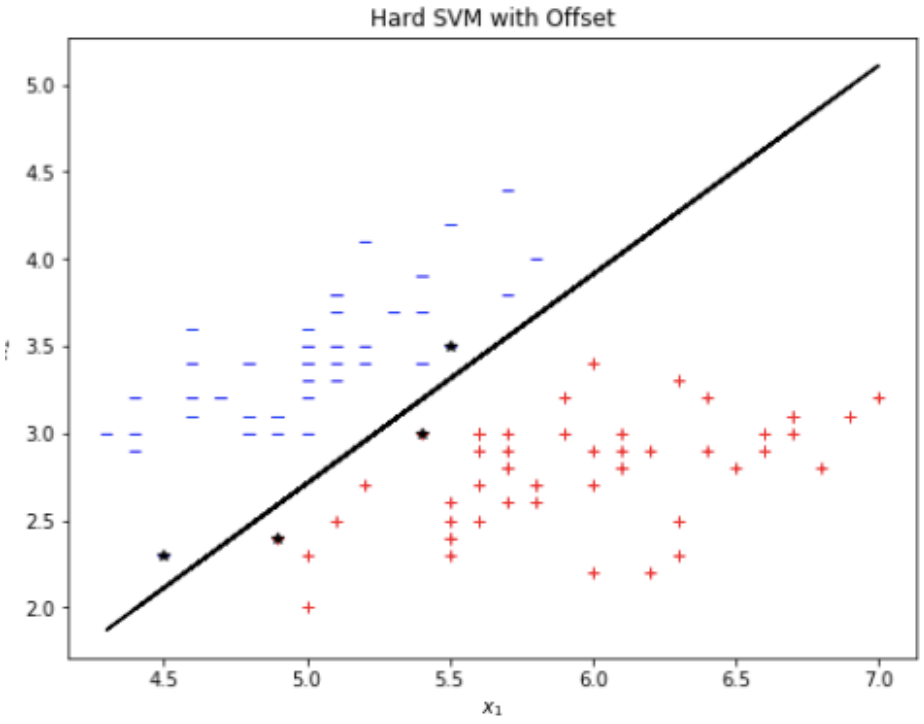
1. **It is clearly linearly separable.**



2.

```
Problem 2 Q2
================================
theta_0*: -17.3157894737
theta^*: [ 6.31578947 -5.26315789]
Optimal objective value is: 33.7950138504
```

3. **The support vectors are asterisk(*) in black.**



4.

```
Objective value is: 33.7949973533
Alpha from quadprog is: 15.9002661526  at index: 36
Alpha from quadprog is: 17.8947302452  at index: 41
Alpha from quadprog is: 16.3988888621  at index: 57
Alpha from quadprog is: 17.3961076312  at index: 84
```

The α larger than 10^{-6} are reported at their respective indices. The objective value is the same as the primal.

Primal: `Execution time is 0.0003820000000001045`

Dual: `Execution time is 0.005471999999999255`

The primal takes a shorter time to solve than the dual (CPU time measured from `time.clock()` in Python)

The primal will take a shorter time to solve since the number of points (n=100) is significantly larger than number of dimensions (d=2). In the primal case, we can solve in $O(d)$ while dual will take $O(n)$ as a bigger dimension matrix (n X n) needs to be computed.

5.

```
First sum which gives theta's value is: [ 6.31578606 -5.26315523]
Index is 84 and the second sum which gives theta_0's value is: -17.3157
790286
```

For the second sum, a random index is picked from [36, 41, 57, 84].

Since these formulas satisfy the optimal KKT conditions, these sums give us the value of the optimal θ and θ_o respectively.

Problem 3 - Soft-SVM and Cross-validation

1.
```
Problem 3 Q1
==================================
theta_0*: -20.4130434783
theta^*: [ -1.84782609  -3.26086957   4.67391304  10.86956522]
Optimal value is: 654.194234405
Misclassified points are 3 at: [20, 33, 83]
```

2.
```
C = 1, Av. test error = 0.4
C = 100, Av. test error = 0.6
C = 10000, Av. test error = 0.7
Optimal C = 1
```