CS 4O03 Assignment 1

James You
001419572
youy2

October 2, 2018

```
In [1]: import scipy.optimize as opt
        import numpy as np
        import matplotlib.pyplot as plt
        import math
```

Question 1

We are trying to formulate a plane s.t.

```
Ax >= B for all points in S1
Ax <= B for all points in S2
```

(or vice versa)
Since strict inequalities are not allowed and `A != (0, 0, 0)`, that is to say simply formulating

```
min. 0

subject to.

Ax - B >= 0
Ax - B <= 0
```

Running the above solution through an optimizer will allow for `A = (0, 0, 0)` and therefore, we reach an inadequate solution. We reformulate the problem as

```
Ax >= B + 1
Ax <= B - 1
```

This, in the context of linear classification, will determine if the set of points are *linearly separable*. After changing both inequalities to upper bounds with a constant RHS

```
Ax + B <= -1
Ax - B <= -1
```

Now we can formulate the problem
Decision variables - Let:

```
A[i] represent the coefficient corresponding to x[i]
B    represent the constant of the plane
```

Therefore our LP becomes

```
min. 0

subject to.

- 1*A[0] - 2*A[1]        - 3*A[2]       + B <= -1
- 3*A[0] - 1*A[1]        - 2*A[2]       + B <= -1
- 2*A[0] - 3*A[1]        - 1*A[2]       + B <= -1
           2*A[1]                       - B <= -1
  4*A[0] + 2*A[1]        + 4*A[2]       - B <= -1
 pi*A[0] + log(pi)*A[1] + sqrt(2)*A[2] - B <= -1
```

We are *not* trying to find an optimal hyperplane separating the two classes, we are simply trying to find *a* hyperplane which separates them.

```
In [2]: # question 1 pt. 1

        A_ub = np.array([
        [-1, -2, -3, 1],
        [-3, -1 ,-2, 1],
        [-2 ,-3 ,-1, 1],
        [0, 2, 0, -1],
        [4, 2, 4, -1],
        [math.pi, math.log(math.pi), math.sqrt(2), -1]])

        b_ub = np.array([-1, -1, -1, -1, -1, -1])

        c = np.array([0, 0, 0, 0])

        res = opt.linprog(c, A_ub=A_ub, b_ub=b_ub,
        bounds=((None, None), (None, None), (None, None), (0, None)))
        print(res)
```

```
     fun: 4.602353985700042
 message: 'Optimization failed. Unable to find a feasible starting point.'
     nit: 6
  status: 2
 success: False
       x: nan
```

We see from the above optimization run, the problem we have formulated is infeasible as the optimizer cannot find a starting point, and therefore, there exists *no* plane which can separate the points and they are not *linearly separable*. To show that no perfect plane exists with a terminating optimization problem, we will reformulate the optimization problem into a linear classification problem such that

```
Ax - B >= 1 - u
Ax - B <= - (1 - v)
```

Where u and v represent a margin from a optimal hyperplane and the points. We then formulate the LP with:
Decision variables - Let

```
A[i] represent the coefficient corresponding to x[i]
B    represent the constant of the plane
u[i] represent the distance between the plane + 1 and each point in S1
v[i] represent the distance between the plane - 1 and each point in S2

min. u[1] + u[2] + u[3] + v[1] + v[2] + v[3]
```

```
subject to.

- 1*A[0] - 2*A[1]       - 3*A[2]       + B - u[1] <= -1
- 3*A[0] - 1*A[1]       - 2*A[2]       + B - u[2] <= -1
- 2*A[0] - 3*A[1]       - 1*A[2]       + B - u[3] <= -1
           2*A[1]                      - B - v[1] <= -1
  4*A[0] + 2*A[1]       + 4*A[2]       - B - v[2] <= -1
 pi*A[0] + log(pi)*A[1] + sqrt(2)*A[2] - B - v[3] <= -1
 u[i], v[i] >= 0
```

We are now trying the minimize the number of points which are on the wrong side of the hyperplane (or in other words, misclassified). If the optimal objective value is `0`, then we have found a perfect separating hyperplane. Otherwise the optimal objective value after optimization is the best we can do in terms of classification. Therefore, there is no perfect separating hyperplane *and* the parameters returned is *best* plane which separates as many points correctly as possible.

```
In [3]: # question 2 pt. 2

        A_ub = np.array([
        [-1, -2, -3, 1, -1, 0, 0, 0, 0, 0],
        [-3, -1, -2, 1, 0, -1, 0, 0, 0, 0],
        [-2, -3, -1, 1, 0, 0, -1, 0, 0, 0],
        [0, 2, 0, -1, 0, 0, 0, -1, 0, 0],
        [4, 2, 4, -1, 0, 0, 0, 0, -1, 0],
        [math.pi, math.log(math.pi), math.sqrt(2), -1, 0, 0, 0, 0, 0, -1]])


        b_ub = np.array([-1, -1, -1, -1, -1, -1])

        c = np.array([0, 0, 0, 0, 1, 1, 1, 1 ,1 ,1])

        res = opt.linprog(c, A_ub=A_ub, b_ub=b_ub,
        bounds=((None, None), (None, None), (None, None), (0, None), (0, None),
                (0, None), (0, None), (0, None), (0, None), (0, None)))
        print(res)
```

```
     fun: 4.602353985700042
 message: 'Optimization terminated successfully.'
     nit: 11
   slack: array([0., 0., 0., 0., 0., 0.])
  status: 0
 success: True
       x: array([0.04823451, 1.25294248, 0.6505885 , 3.50588496, 0.        ,
       1.80706196, 0.        , 0.        , 2.79529203, 0.        ])
```

The optimal function value is ~`4.6`, therefore there exists no perfect separating hyperplane. The difference between the current LP and the previous LP is that the previous LP *required* a perfect

separating hyperplane, whereas the current LP requires the most *optimal* hyperplane. If the problem could be formulated as a quadratic program, we could add a hyperparameter which regulates the margins.
Sources consulted:
https://en.wikipedia.org/wiki/Support_vector_machine
https://yalmip.github.io/tutorial/linearprogramming/

Question 2

Decision variables - Let

```
x[0] represent the number of tables we sell
x[1] represent the number of desks we sell
x[2] represent the number of chairs we sell
```

We formulate the LP as:

```
max. 100*x[0] + 50*x[1] + 10*x[2]

subject to.

25*[0] + 15*x[1] + 10*x[2] <= 4000
50*[0] + 30*x[1] + 10*x[2] <= 5000
90*[0] + 50*x[1] + 25*x[2] <= 6000
4*x[0] + x[1] <= x[2] # 4x chairs as tables + 1x chairs as desks <= total number of chairs we se
x[0], x[1], x[2] >= 0
```

```
In [4]: # question 2 pt. 2

        A_ub = np.array([
        [25, 15, 10],
        [50, 30 ,10],
        [90 ,50 ,25],
        [4, 1, -1]])

        b_ub = np.array([4000, 5000, 6000, 0])

        c = np.array([100, 50, 10])

        # default is a min problem. max f(x) = -min f(-x)
        res = opt.linprog(-c, A_ub=A_ub, b_ub=b_ub,
        bounds=(0, None))
        print(res)
```

```
     fun: -4800.0
 message: 'Optimization terminated successfully.'
     nit: 3
   slack: array([2000., 1800.,    0.,    0.])
```

```
  status: 0
 success: True
       x: array([ 0., 80., 80.])
```

The maximum revenue is $4800. The optimal product mix is 80 desks and 80 chairs.

Question 3

Decision variables - Let

```
x[i] be the number of trees we can buy normally in week i

z[3] be the number of extra trees we can buy in week 3
z[4] be the number of extra trees we can buy in week 4
```

Other variables - Let

```
y[1] be the number of trees left over in week 1
y[2] be the number of trees left over in week 2
y[3] be the number of trees left over in week 3
* there should be no trees left over in week 4
```

We are trying to minimize the cost in order to meet the demand of trees every week, therefore We formulate the LP as:

```
min. 100*(x[1] + x[2]) + 150*(x[3] + x[4]) + 3*(y[1] + y[2] + y[3]) + 200*(z[3] + z[4])

subject to.

x[1] - 70                == y[1]
x[2] + y[1] - 80         == y[2]
x[3] + y[2] + z[3] - 90  == y[3]
x[4] + y[3] + z[4] - 100 == 0   # no leftover trees after 4th week
x[1], x[2], x[3], x[4]   <= 90
z[3], z[4]               <= 20
x[1], x[2], x[3], x[4]   >= 0
y[1], y[2], y[3]         >= 0
z[3], z[4]               >= 0
```