EE 550 HW2

Taha Küçükkatırcı - 2013400213

March 20, 2018

```
In [1]: %matplotlib notebook
        import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        from math import exp
        import itertools
```

1 Data Generation

- Firstly, I generate data points in 3D space. We are asked to pick 40 data points, 20 from 1^{st} quadrant and 20 from 8^{th} quadrant.
- I limited the point coordinates in a range of 3. So the points are defined as follows:

$$x_k, y_k, z_k \in \begin{cases} [-3,0] & \text{if } k^{th} \text{ data point lies in } 8^{th} \text{ quadrant} \\ [0,3] & \text{if } k^{th} \text{ data point lies in } 1^{st} \text{ quadrant} \end{cases} \tag{1}$$

```
In [2]: positive_data = np.array([]).reshape(0,3)
        negative_data = np.array([]).reshape(0,3)

In [3]: for _ in range(20):
            positive_data = np.vstack((positive_data, 3 * np.random.random_sample((1,3))))
            negative_data = np.vstack((negative_data, 3 * np.random.random_sample((1,3))-3))

        X = np.vstack((positive_data,negative_data))
        y = np.vstack((np.ones((20,1)), -np.ones((20,1))))

In [4]: X = X.tolist()
        y = y.tolist()
        y = list(itertools.chain.from_iterable(y))

In [5]: X   # data points

Out[5]: [[0.5858007805649135, 0.05927581707333862, 0.8459469971365126],
         [1.8171918571806729, 0.8145646504001739, 0.34532751433325337],
         [1.0694253329388452, 2.0298290414463915, 1.2341999297623383],
         [2.074954774721652, 0.25990892057417103, 1.0154887561565],
         [0.44229568774092676, 2.1995094348849316, 0.6950257685724709],
```

```
 [2.359296720568577, 0.9675688767048259, 1.1279595768867456],
 [1.385110066860887, 2.616853876017121, 0.044442862195807065],
 [0.009881683228215588, 0.01710934514123097, 2.7905680760342095],
 [0.988806281441412, 1.890513442527648, 0.5262356791933688],
 [1.7094428148493215, 2.85817807866121, 0.49889471761633286],
 [2.958726051345482, 0.23599441693405576, 0.6454278792783305],
 [0.3569315785275968, 1.9371177170719855, 2.541599986057931],
 [1.3214326467765685, 1.7057809086811937, 0.794240754523524],
 [2.469016108102827, 2.342970102678467, 0.48454270252945375],
 [0.16740824523039965, 0.19347032778387518, 1.1927551551519608],
 [0.5728071328548746, 0.4078919412505103, 2.722795008470042],
 [2.9471084393946985, 0.753675361767169, 2.896357471214022],
 [1.5199221819048974, 2.3408349931461765, 1.2602569888645663],
 [2.707975169955335, 0.9634640970489909, 2.3565455245386007],
 [0.9269931526682902, 1.970584937370557, 0.5064038382176116],
 [-1.5512637046272657, -2.2602717122075022, -1.8305329751213808],
 [-0.9302843858333119, -0.6899037934793388, -0.8338115310566314],
 [-2.22616528400732, -1.0320689171790864, -0.14156927799768582],
 [-0.7716345967816953, -2.6078893792001896, -2.3453292701191293],
 [-1.8577853840941936, -2.9726342308506553, -2.6501072058638258],
 [-1.7669721146245079, -1.7820055212358983, -0.8538027113741431],
 [-0.030280227511559232, -2.0985128737695384, -2.1950622643456312],
 [-0.21323255810357988, -0.16078566117964144, -2.5609333647412154],
 [-2.7640934177405176, -0.21333453283658566, -0.6390775597726961],
 [-2.303494905025318, -0.7790604090403814, -1.33665620831156],
 [-1.1899250578941376, -0.8313092727300537, -1.3713988569581015],
 [-2.139786596909902, -0.18356693728771045, -2.947225270988773],
 [-0.06954067084195037, -2.5548114408559477, -0.5045838212904195],
 [-0.6504326402055902, -0.6764513241508636, -0.31293358188273945],
 [-1.0217573174551227, -0.8582630273145453, -0.4510662545925985],
 [-1.9825954795998872, -2.507778190719116, -0.2732344125280277],
 [-2.306841029401398, -1.285432309244655, -2.7497260104094043],
 [-2.7909322276083204, -2.4982036077735135, -2.8875636816802315],
 [-1.9954251922986104, -1.5773680677717052, -0.7347693018862742],
 [-2.0827784689348987, -2.1048870912925777, -1.1968325247388796]]
```

```
In [6]: y    # classes
```

```
Out[6]: [1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
```

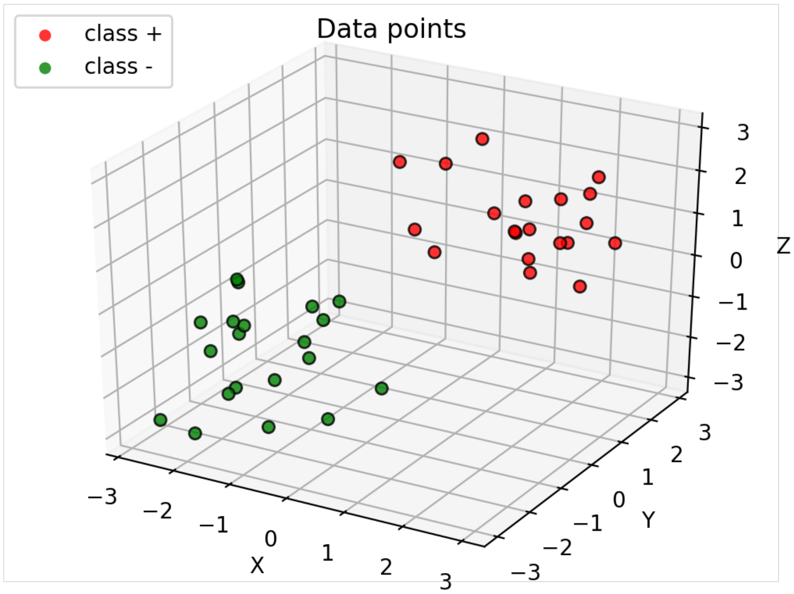
```
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0]
```

2 Plot of data points

- In this part, I am plotting 40 data points in 3D space. You can clearly see the points clustered in two quadrants.
- For this and further plots, I used interactive plots, so you can do some operations on plot like zoom in/out, rotate etc.

```
In [7]: data = (positive_data, negative_data)
        colors = ("red", "green")
        groups = ("class +", "class -")

        # Create plot
        fig = plt.figure()
        ax = fig.gca(projection='3d')

        for data, color, group in zip(data, colors, groups):
```

```
    ax.scatter(data[:,0], data[:,1], data[:,2],alpha=0.8, c=color, edgecolors='none', s=

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.title('Data points')
plt.legend(loc=2)
plt.show()
```



3 Activation Function

- As activation function I used sign function, i.e

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2}$$

```
In [8]: def predict(x):
            return 1 if x>=0 else -1

In [9]: def difference(a,b):
            return b-a
```

4 Random Initialization of Weights

- Before running the perceptron learning algorithm I initialized the weight vector w with random values from $\mathcal{N}(0, 1)$.

```
In [10]: w = np.random.randn(3,1)
         w = w.tolist()
         w = list(itertools.chain.from_iterable(w))
         w
```

```
Out[10]: [1.248887418254114, -1.3237643024481913, -0.3409428538154275]
```

5 Running the Perceptron Algorithm

- In this part, I run the perceptron algorithm.
- In one iteration, all data points are traversed and weights are updated for each data point, i.e 40 times per iteration.
- You can change alpha (learning rate) value and check the speed of convergence for different alpha values.

```
In [11]: alpha = 0.01      # learning rate
         accuracy = 0
         errors = []       # list to store error of each iteration
         while accuracy<1:
             correct=0           # will store number of correctly classified data points
             epoch_error=0       # will store the error of current iteration
             for i in range(len(X)):
                 output = 0
                 for j in range(len(X[i])):
                     output += X[i][j]*w[j]

                 output = float(format(output, '.2f'))
                 output = predict(output)
                 diff = difference(output,y[i])    # checks whether i'th data point is correctly

                 if diff==0:
                     correct+=1
                     continue

                 epoch_error += diff**2

                 # update the weights according to the error of i'th data point.
                 for j in range(len(w)):
                     w[j] += alpha*diff*X[i][j]


             accuracy = correct/len(X)    # iteration at the end of one iteration
             print(accuracy)
```

```
            print("=======")
            errors.append(epoch_error)    # iteration error added to the list
```

```
0.625
=======
0.875
=======
0.925
=======
0.95
=======
0.975
=======
0.975
=======
0.975
=======
0.975
=======
1.0
=======
```

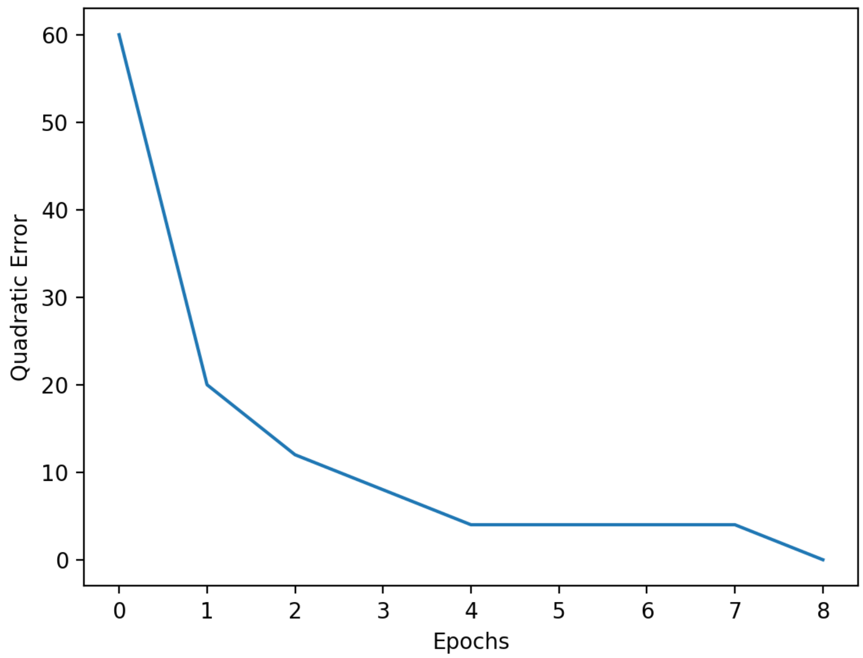
6 Final Weights

```
In [12]: w
```

```
Out[12]: [1.5471104542771004, -0.07567405880603528, 0.36614487463374285]
```

7 Plot of Iteration vs Error Curve

```
In [13]: from matplotlib.ticker import MaxNLocator
         ax = plt.figure().gca()
         ax.xaxis.set_major_locator(MaxNLocator(integer=True))
         ax.set_xlabel('Epochs')
         ax.set_ylabel('Quadratic Error')
         plt.plot(np.arange(len(errors)), errors)
```



Out[13]: [<matplotlib.lines.Line2D at 0x10960e710>]

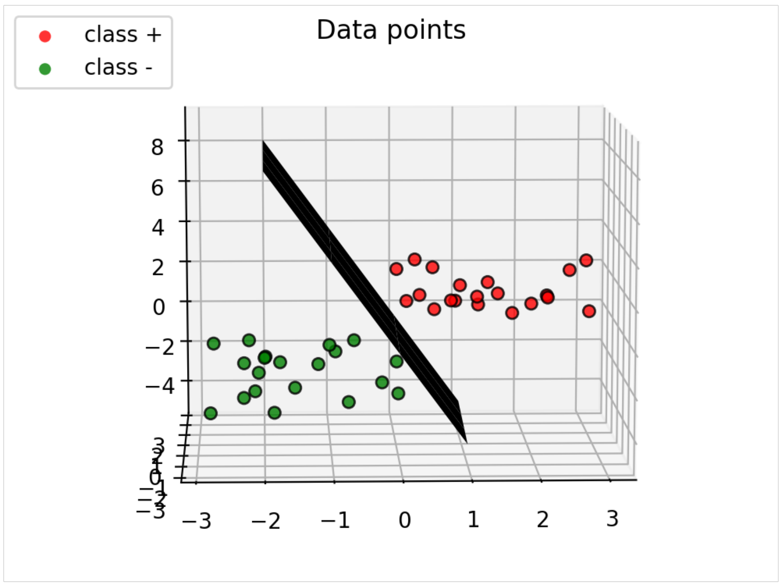
8 Plot of Data Points and Separating Hyperplane

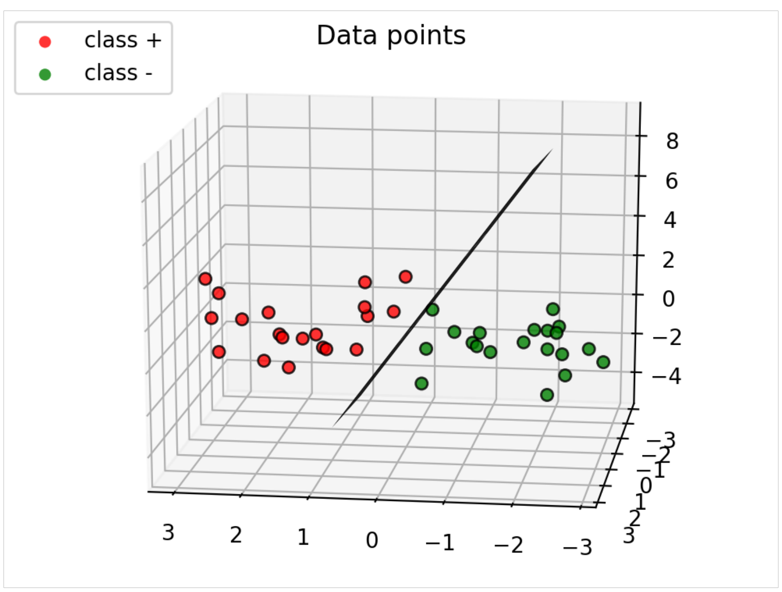
- You can rotate the plot and see the data points and hyperplane from different angles.

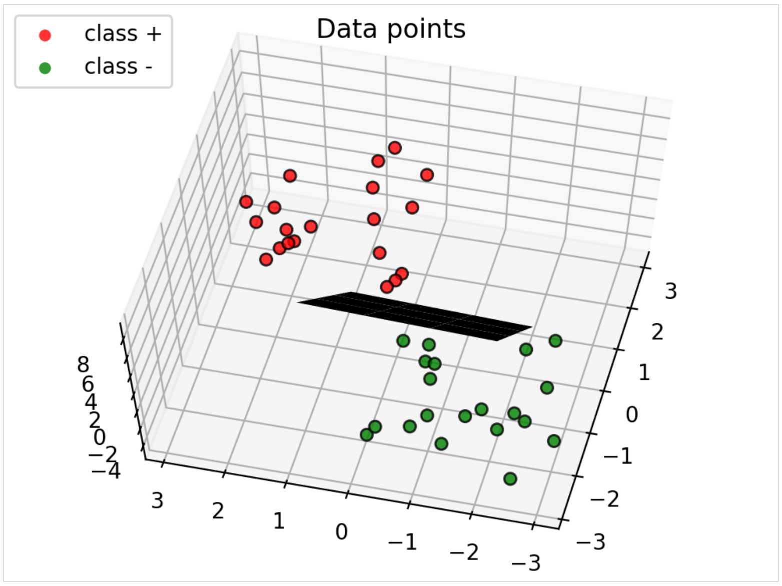
In [14]:
```
data = (positive_data, negative_data)
colors = ("red", "green")
groups = ("class +", "class -")

# Create plot
fig = plt.figure()
ax = fig.gca(projection='3d')

for data, color, group in zip(data, colors, groups):
    ax.scatter(data[:,0], data[:,1], data[:,2],alpha=0.8, c=color, edgecolors='none', s

plt.title('Data points')
plt.legend(loc=2)
#plt.show()

a = np.arange(-2, 2, 1)
b = np.arange(-2, 2, 1)

xx, yy = np.meshgrid(a,b, sparse=True)
z = (-w[0] * xx - w[1] * yy) * (1. /w[2])
```

```python
#plt3d = plt.figure().gca(projection='3d')
ax.plot_surface(xx, yy, z, shade=True, color='black')
plt.show()
```

- You can see below data points and separating plane from different angles.






Data points
class +
class -
8
6
4
2
0
−2
−4
3
2
1
0
−1
−2
−3
3
2
1
0
−1
−2
−3