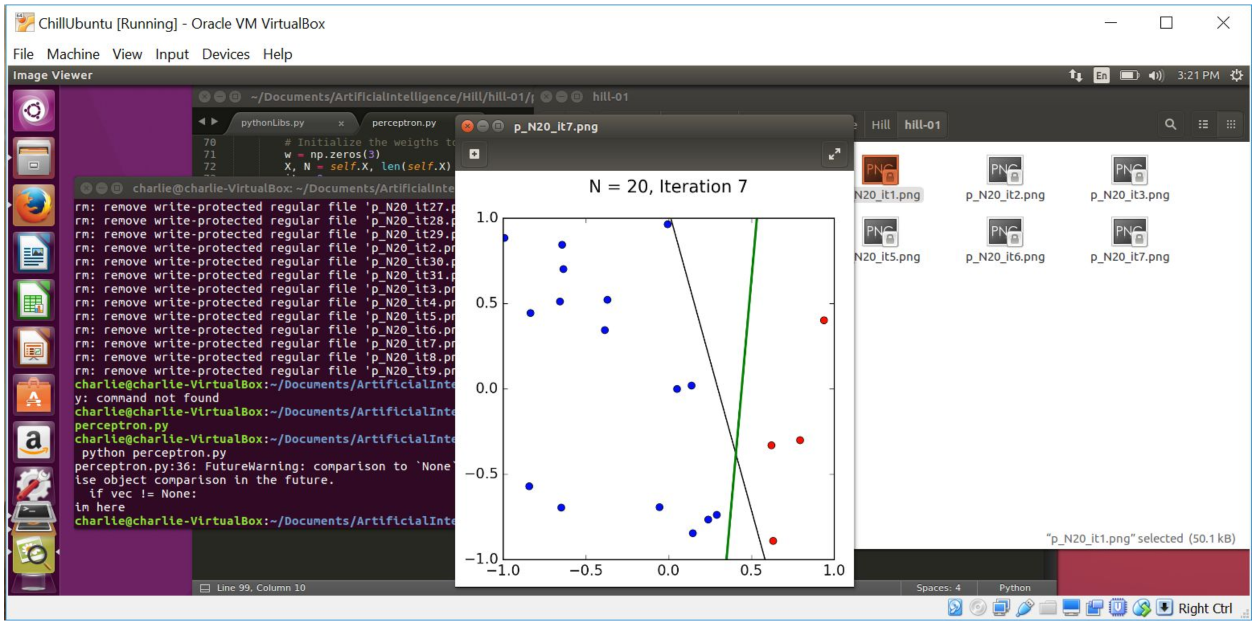
Charlie Hill
Artificial Intelligence
Pablo Rivas
19 September 2016

Homework 01

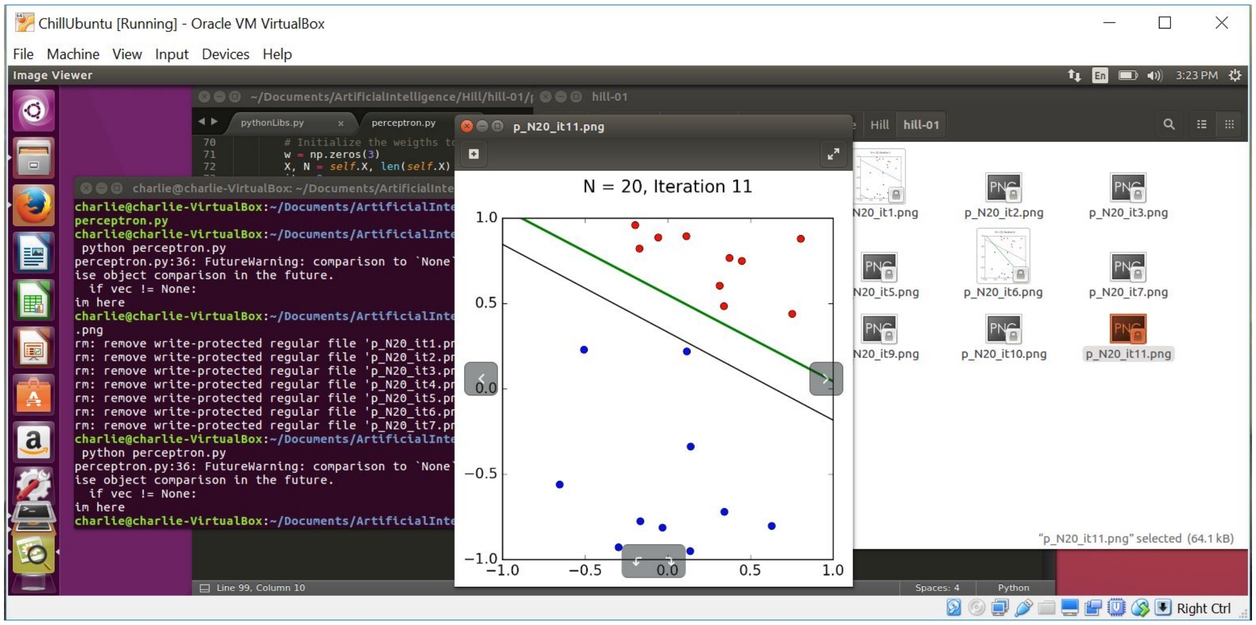
a and b) Running the perceptron with 20 data points gave me this. It took 7 iterations for the perceptron to find a solution. The perceptron does not seem close to the expected solution that is pre generated.



```
        # Initialize the weigths to
        w = np.zeros(3)
        X, N = self.X, len(self.X)
```

```
charlie@charlie-VirtualBox: ~/Documents/ArtificialInte
rm: remove write-protected regular file 'p_N20_it27.p
rm: remove write-protected regular file 'p_N20_it28.p
rm: remove write-protected regular file 'p_N20_it29.p
rm: remove write-protected regular file 'p_N20_it2.pn
rm: remove write-protected regular file 'p_N20_it30.p
rm: remove write-protected regular file 'p_N20_it31.p
rm: remove write-protected regular file 'p_N20_it3.pn
rm: remove write-protected regular file 'p_N20_it4.pn
rm: remove write-protected regular file 'p_N20_it5.pn
rm: remove write-protected regular file 'p_N20_it6.pn
rm: remove write-protected regular file 'p_N20_it7.pn
rm: remove write-protected regular file 'p_N20_it8.pn
rm: remove write-protected regular file 'p_N20_it9.pn
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
y: command not found
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
perceptron.py
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
 python perceptron.py
perceptron.py:36: FutureWarning: comparison to `None`
ise object comparison in the future.
  if vec != None:
im here
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
```

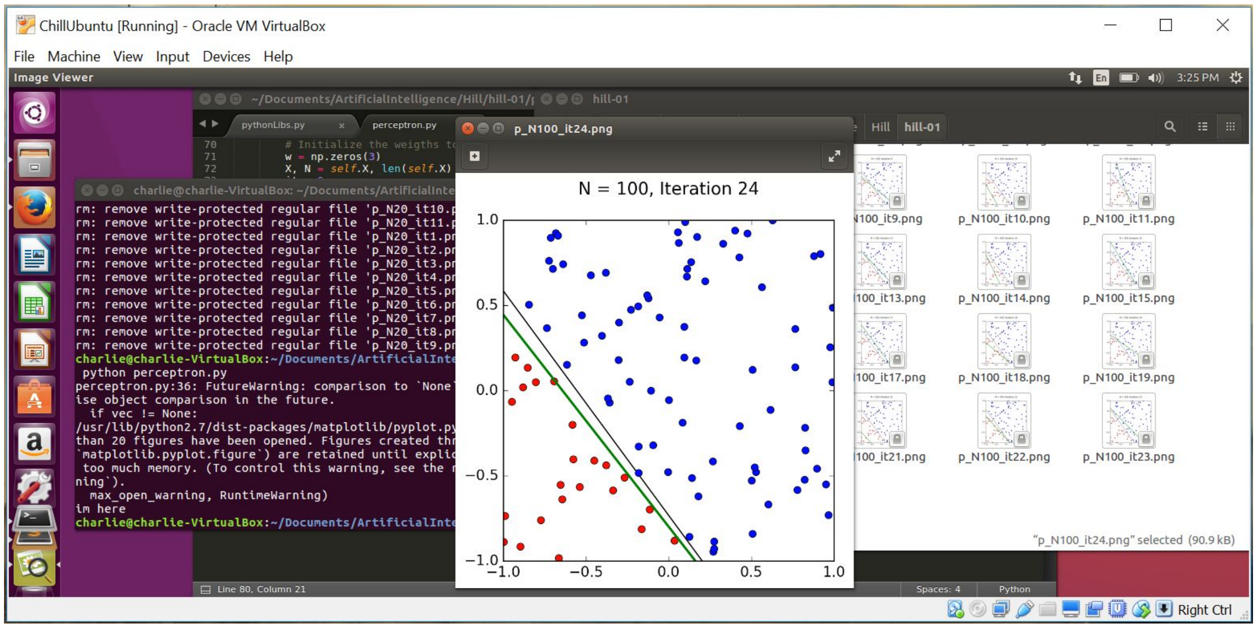
c) I ran the script again and got this.This time it took 11 iterations to find a solution. The perceptron seems much closer to the expected answer this time.



```
        # Initialize the weigths to
        w = np.zeros(3)
        X, N = self.X, len(self.X)
```

```
charlie@charlie-VirtualBox: ~/Documents/ArtificialInte
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
perceptron.py
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
 python perceptron.py
perceptron.py:36: FutureWarning: comparison to `None`
ise object comparison in the future.
  if vec != None:
im here
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
.png
rm: remove write-protected regular file 'p_N20_it1.pn
rm: remove write-protected regular file 'p_N20_it2.pn
rm: remove write-protected regular file 'p_N20_it3.pn
rm: remove write-protected regular file 'p_N20_it4.pn
rm: remove write-protected regular file 'p_N20_it5.pn
rm: remove write-protected regular file 'p_N20_it6.pn
rm: remove write-protected regular file 'p_N20_it7.pn
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
 python perceptron.py
perceptron.py:36: FutureWarning: comparison to `None`
ise object comparison in the future.
  if vec != None:
im here
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
```

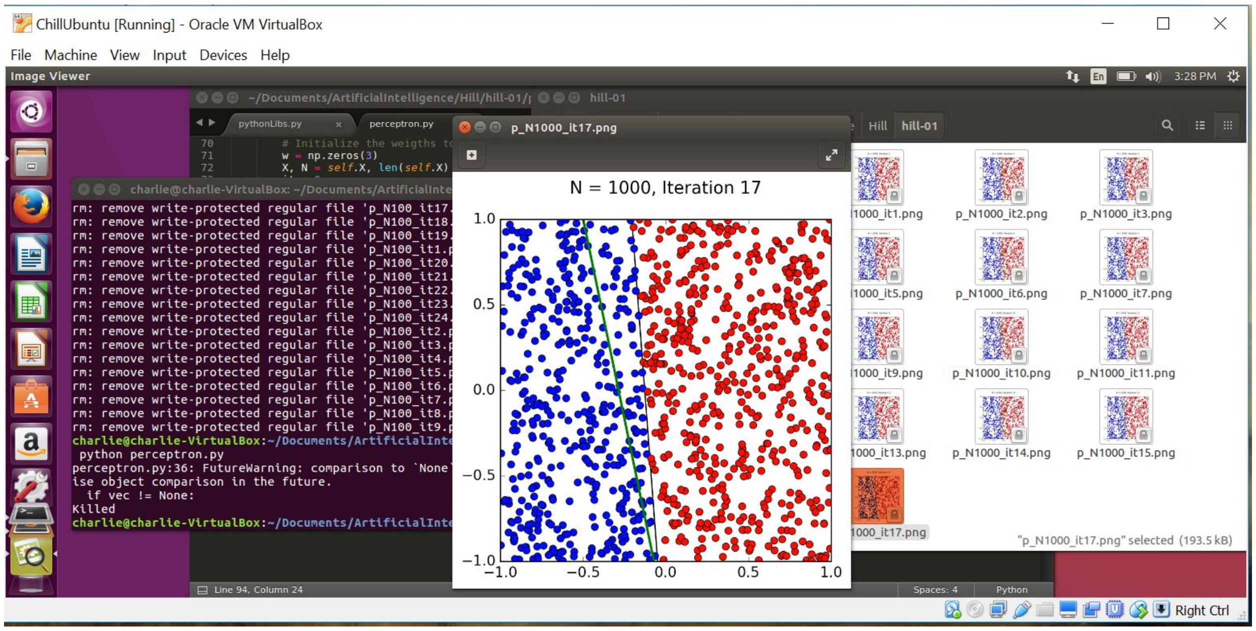
d) I changed the data points to 100 and ran the script again. This time it took 24 iterations and the perceptron is very close to the expected solution.


ChillUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Image Viewer
p_N100_it24.png
N = 100, Iteration 24
"p_N100_it24.png" selected (90.9 kB)

```
70        # Initialize the weigths to
71        w = np.zeros(3)
72        X, N = self.X, len(self.X)
```

```
rm: remove write-protected regular file 'p_N20_it10.p
rm: remove write-protected regular file 'p_N20_it11.p
rm: remove write-protected regular file 'p_N20_it1.pn
rm: remove write-protected regular file 'p_N20_it2.pn
rm: remove write-protected regular file 'p_N20_it3.pn
rm: remove write-protected regular file 'p_N20_it4.pn
rm: remove write-protected regular file 'p_N20_it5.pn
rm: remove write-protected regular file 'p_N20_it6.pn
rm: remove write-protected regular file 'p_N20_it7.pn
rm: remove write-protected regular file 'p_N20_it8.pn
rm: remove write-protected regular file 'p_N20_it9.pn
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
 python perceptron.py
perceptron.py:36: FutureWarning: comparison to `None`
ise object comparison in the future.
  if vec != None:
/usr/lib/python2.7/dist-packages/matplotlib/pyplot.py
than 20 figures have been opened. Figures created thr
`matplotlib.pyplot.figure`) are retained until explic
 too much memory. (To control this warning, see the r
ning`).
  max_open_warning, RuntimeWarning)
im here
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
```

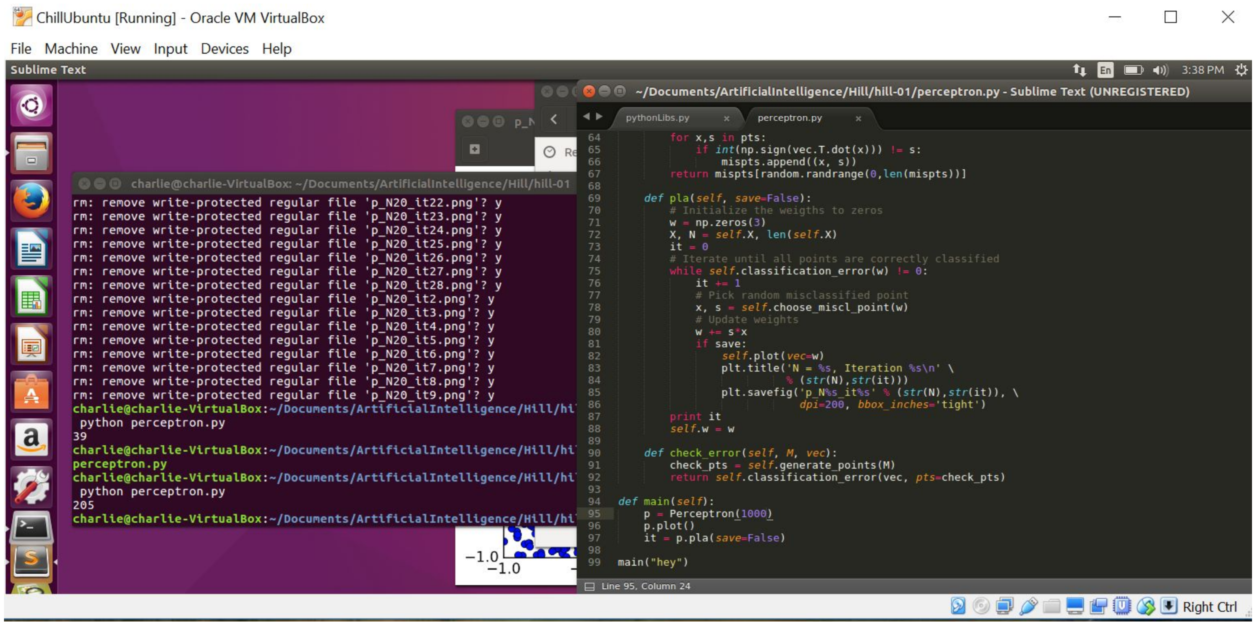
e) I changed the data points to 1000 and ran the script. The first time I ran it, the script was killed after a bit, most likely because it was using too much memory.


ChillUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Image Viewer
p_N1000_it17.png
N = 1000, Iteration 17
"p_N1000_it17.png" selected (193.5 kB)

```
70        # Initialize the weigths to
71        w = np.zeros(3)
72        X, N = self.X, len(self.X)
```

```
rm: remove write-protected regular file 'p_N100_it17.
rm: remove write-protected regular file 'p_N100_it18.
rm: remove write-protected regular file 'p_N100_it19.
rm: remove write-protected regular file 'p_N100_it1.p
rm: remove write-protected regular file 'p_N100_it20.
rm: remove write-protected regular file 'p_N100_it21.
rm: remove write-protected regular file 'p_N100_it22.
rm: remove write-protected regular file 'p_N100_it23.
rm: remove write-protected regular file 'p_N100_it24.
rm: remove write-protected regular file 'p_N100_it2.p
rm: remove write-protected regular file 'p_N100_it3.p
rm: remove write-protected regular file 'p_N100_it4.p
rm: remove write-protected regular file 'p_N100_it5.p
rm: remove write-protected regular file 'p_N100_it6.p
rm: remove write-protected regular file 'p_N100_it7.p
rm: remove write-protected regular file 'p_N100_it8.p
rm: remove write-protected regular file 'p_N100_it9.p
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
 python perceptron.py
perceptron.py:36: FutureWarning: comparison to `None`
ise object comparison in the future.
  if vec != None:
Killed
charlie@charlie-VirtualBox:~/Documents/ArtificialInte
```

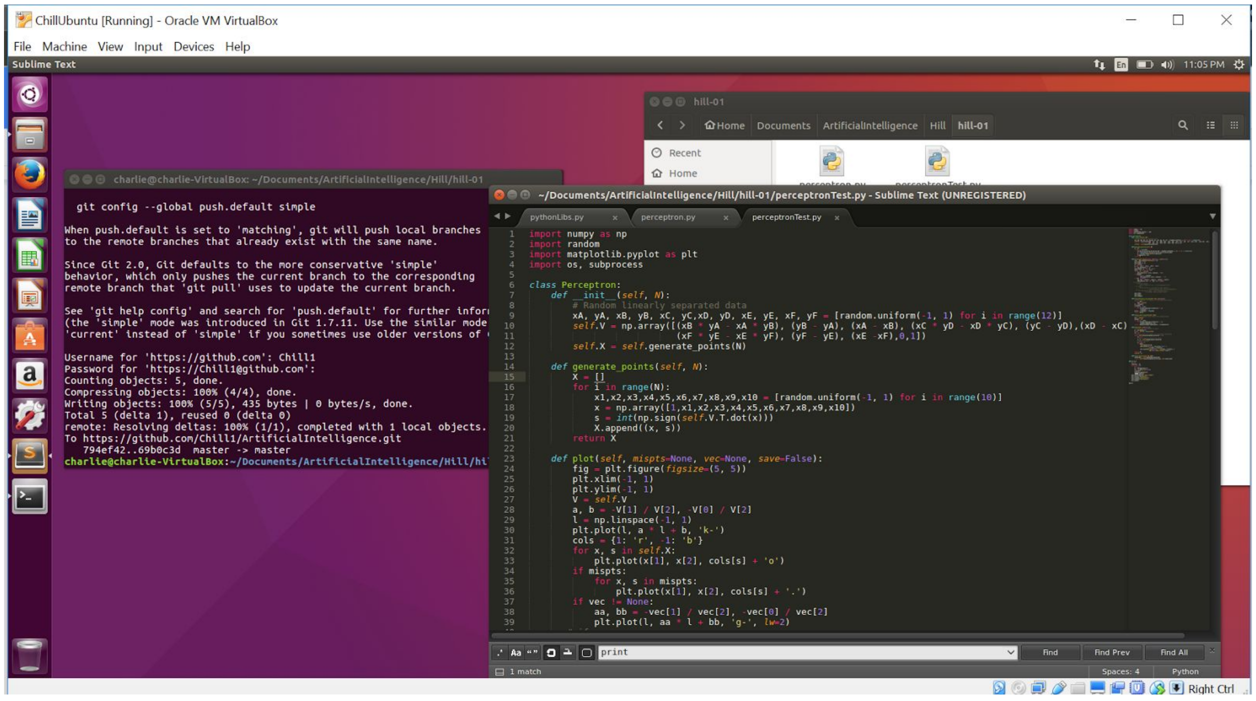
So then I decided to not save the plots so I could find out how many iterations it would actually take. It would end up taking 205 iterations to find a solution. If I had to guess I can imagine the perceptron probably looks really similar to the expected solution because with so many data points the line has to be exact because there is only a little room for solutions. The more we increase the data points the closer the perceptron will get to the expected solution.



```
rm: remove write-protected regular file 'p_N20_it22.png'? y
rm: remove write-protected regular file 'p_N20_it23.png'? y
rm: remove write-protected regular file 'p_N20_it24.png'? y
rm: remove write-protected regular file 'p_N20_it25.png'? y
rm: remove write-protected regular file 'p_N20_it26.png'? y
rm: remove write-protected regular file 'p_N20_it27.png'? y
rm: remove write-protected regular file 'p_N20_it28.png'? y
rm: remove write-protected regular file 'p_N20_it2.png'? y
rm: remove write-protected regular file 'p_N20_it3.png'? y
rm: remove write-protected regular file 'p_N20_it4.png'? y
rm: remove write-protected regular file 'p_N20_it5.png'? y
rm: remove write-protected regular file 'p_N20_it6.png'? y
rm: remove write-protected regular file 'p_N20_it7.png'? y
rm: remove write-protected regular file 'p_N20_it8.png'? y
rm: remove write-protected regular file 'p_N20_it9.png'? y
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hi
 python perceptron.py
39
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hi
perceptron.py
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hi
 python perceptron.py
205
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hi
```

```python
        for x,s in pts:
            if int(np.sign(vec.T.dot(x))) != s:
                mispts.append((x, s))
        return mispts[random.randrange(0,len(mispts))]

    def pla(self, save=False):
        # Initialize the weigths to zeros
        w = np.zeros(3)
        X, N = self.X, len(self.X)
        it = 0
        # Iterate until all points are correctly classified
        while self.classification_error(w) != 0:
            it += 1
            # Pick random misclassified point
            x, s = self.choose_miscl_point(w)
            # Update weights
            w += s*x
            if save:
                self.plot(vec=w)
                plt.title('N = %s, Iteration %s\n' \
                          % (str(N),str(it)))
                plt.savefig('p_N%s_it%s' % (str(N),str(it)), \
                            dpi=200, bbox_inches='tight')
        print it
        self.w = w

    def check_error(self, M, vec):
        check_pts = self.generate_points(M)
        return self.classification_error(vec, pts=check_pts)

def main(self):
    p = Perceptron(1000)
    p.plot()
    it = p.pla(save=False)

main("hey")
```

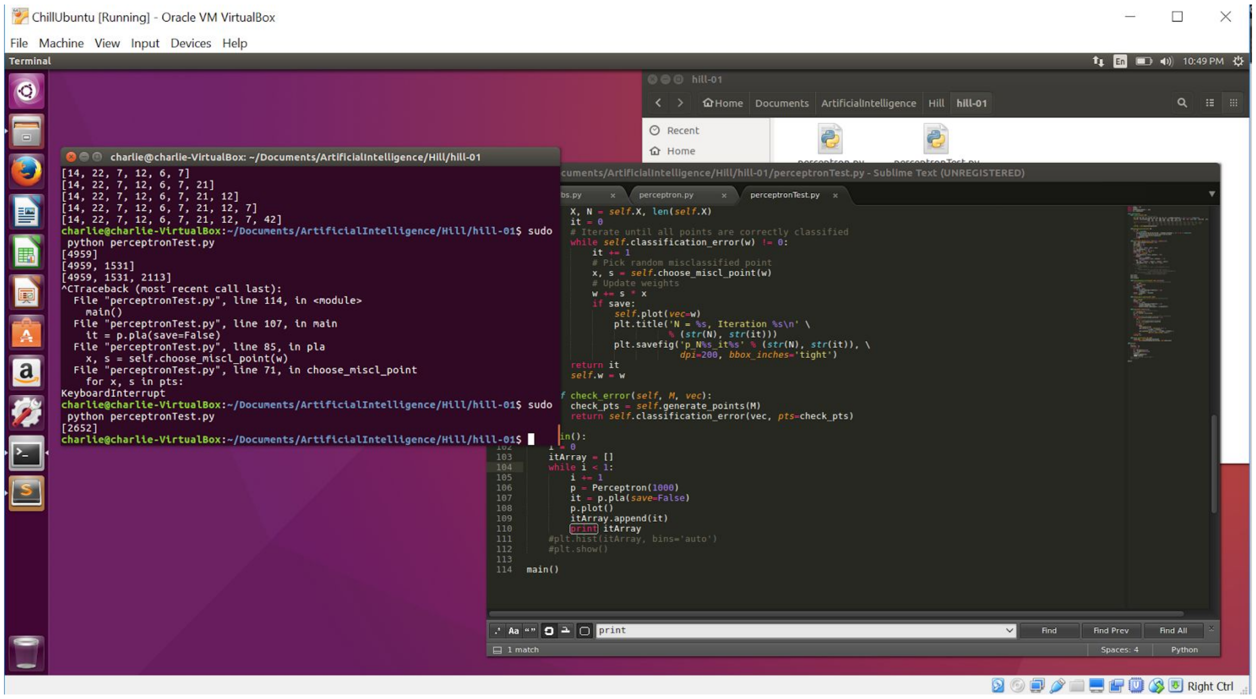
f)This was by far the hardest part of the homework to figure out in my opinion. Altering the program to allow for 10 dimensions was not easy. However, I was eventually able to figure it out and get it working. From there running the program became much slower. I imagine the perceptron has much more thinking to do in 10 dimensions.



```
  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further infor
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of

Username for 'https://github.com': Chill1
Password for 'https://Chill1@github.com':
Counting objects: 5, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 435 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/Chill1/ArtificialIntelligence.git
   794ef42..69b0c3d  master -> master
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hi
```

```python
import numpy as np
import random
import matplotlib.pyplot as plt
import os, subprocess

class Perceptron:
    def __init__(self, N):
        # Random linearly separated data
        xA, yA, xB, yB, xC, yC,xD, yD, xE, yE, xF, yF = [random.uniform(-1, 1) for i in range(12)]
        self.V = np.array([(xB * yA - xA * yB), (yB - yA), (xA - xB), (xC * yD - xD * yC), (yC - yD),(xD - xC)
                           (xF * yE - xE * yF), (yF - yE), (xE -xF),0,1])
        self.X = self.generate_points(N)

    def generate_points(self, N):
        X = []
        for i in range(N):
            x1,x2,x3,x4,x5,x6,x7,x8,x9,x10 = [random.uniform(-1, 1) for i in range(10)]
            x = np.array([1,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10])
            s = int(np.sign(self.V.T.dot(x)))
            X.append((x, s))
        return X

    def plot(self, mispts=None, vec=None, save=False):
        fig = plt.figure(figsize=(5, 5))
        plt.xlim(-1, 1)
        plt.ylim(-1, 1)
        V = self.V
        a, b = -V[1] / V[2], -V[0] / V[2]
        l = np.linspace(-1, 1)
        plt.plot(l, a * l + b, 'k-')
        cols = {1: 'r', -1: 'b'}
        for x, s in self.X:
            plt.plot(x[1], x[2], cols[s] + 'o')
        if mispts:
            for x, s in mispts:
                plt.plot(x[1], x[2], cols[s] + '.')
        if vec != None:
            aa, bb = -vec[1] / vec[2], -vec[0] / vec[2]
            plt.plot(l, aa * l + bb, 'g-', lw=2)
```

I ran the program with 1000 data points. It took 2652 iterations to find the perceptron in this screenshot. I ran it a few times and found around 4000 iterations seemed to be pretty common.



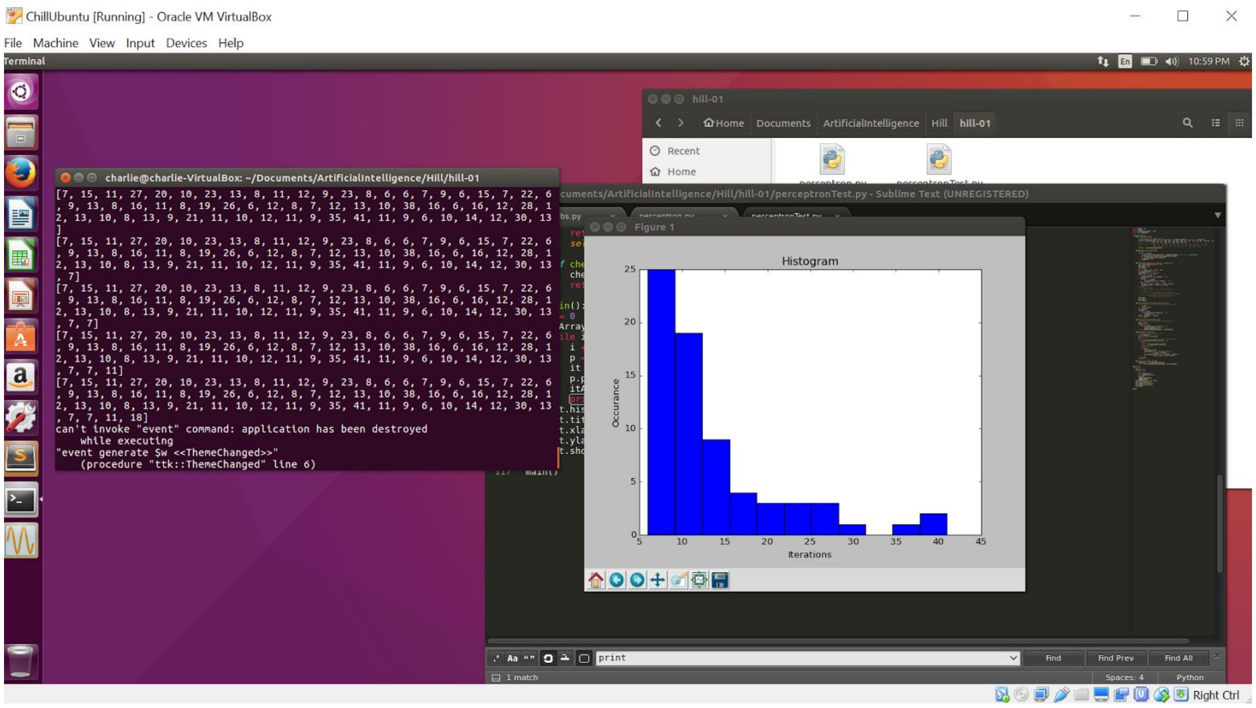
ChillUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
10:49 PM
hill-01
Home Documents ArtificialIntelligence Hill hill-01
Recent
Home
bs.py
perceptron.py
perceptronTest.py
Find
Find Prev
Find All
Spaces: 4
Python
Right Ctrl

```
charlie@charlie-VirtualBox: ~/Documents/ArtificialIntelligence/Hill/hill-01
[14, 22, 7, 12, 6, 7]
[14, 22, 7, 12, 6, 7, 21]
[14, 22, 7, 12, 6, 7, 21, 12]
[14, 22, 7, 12, 6, 7, 21, 12, 7]
[14, 22, 7, 12, 6, 7, 21, 12, 7, 42]
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hill-01$ sudo
 python perceptronTest.py
[4959]
[4959, 1531]
[4959, 1531, 2113]
^CTraceback (most recent call last):
  File "perceptronTest.py", line 114, in <module>
    main()
  File "perceptronTest.py", line 107, in main
    it = p.pla(save=False)
  File "perceptronTest.py", line 85, in pla
    x, s = self.choose_miscl_point(w)
  File "perceptronTest.py", line 71, in choose_miscl_point
    for x, s in pts:
KeyboardInterrupt
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hill-01$ sudo
 python perceptronTest.py
[2652]
charlie@charlie-VirtualBox:~/Documents/ArtificialIntelligence/Hill/hill-01$
```

```
cuments/ArtificialIntelligence/Hill/hill-01/perceptronTest.py - Sublime Text (UNREGISTERED)
        X, N = self.X, len(self.X)
        it = 0
        # Iterate until all points are correctly classified
        while self.classification_error(w) != 0:
            it += 1
            # Pick random misclassified point
            x, s = self.choose_miscl_point(w)
            # Update weights
            w += s * x
            if save:
                self.plot(vec=w)
                plt.title('N = %s, Iteration %s\n' \
                          % (str(N), str(it)))
                plt.savefig('p_N%s_it%s' % (str(N), str(it)), \
                            dpi=200, bbox_inches='tight')
        return it
        self.w = w

    f check_error(self, M, vec):
        check_pts = self.generate_points(M)
        return self.classification_error(vec, pts=check_pts)

    in():
    i = 0
    itArray = []
    while i < 1:
        i += 1
        p = Perceptron(1000)
        it = p.pla(save=False)
        p.plot()
        itArray.append(it)
        print itArray
    #plt.hist(itArray, bins='auto')
    #plt.show()

main()
```

```
print
1 match
```

g) I ran the algorithm 100 times with 20 data points. By the time it got to the end the process was being killed so I had to lower the number of times it was being run. It was able to be run 70 times pretty consistently. I built the histogram and here is a screenshot of it.



```
charlie@charlie-VirtualBox: ~/Documents/ArtificialIntelligence/Hill/hill-01
[7, 15, 11, 27, 20, 10, 23, 13, 8, 11, 12, 9, 23, 8, 6, 6, 7, 9, 6, 15, 7, 22, 6
, 9, 13, 8, 16, 11, 8, 19, 26, 6, 12, 8, 7, 12, 13, 10, 38, 16, 6, 16, 12, 28, 1
2, 13, 10, 8, 13, 9, 21, 11, 10, 12, 11, 9, 35, 41, 11, 9, 6, 10, 14, 12, 30, 13
]
[7, 15, 11, 27, 20, 10, 23, 13, 8, 11, 12, 9, 23, 8, 6, 6, 7, 9, 6, 15, 7, 22, 6
, 9, 13, 8, 16, 11, 8, 19, 26, 6, 12, 8, 7, 12, 13, 10, 38, 16, 6, 16, 12, 28, 1
2, 13, 10, 8, 13, 9, 21, 11, 10, 12, 11, 9, 35, 41, 11, 9, 6, 10, 14, 12, 30, 13
, 7]
[7, 15, 11, 27, 20, 10, 23, 13, 8, 11, 12, 9, 23, 8, 6, 6, 7, 9, 6, 15, 7, 22, 6
, 9, 13, 8, 16, 11, 8, 19, 26, 6, 12, 8, 7, 12, 13, 10, 38, 16, 6, 16, 12, 28, 1
2, 13, 10, 8, 13, 9, 21, 11, 10, 12, 11, 9, 35, 41, 11, 9, 6, 10, 14, 12, 30, 13
, 7, 7]
[7, 15, 11, 27, 20, 10, 23, 13, 8, 11, 12, 9, 23, 8, 6, 6, 7, 9, 6, 15, 7, 22, 6
, 9, 13, 8, 16, 11, 8, 19, 26, 6, 12, 8, 7, 12, 13, 10, 38, 16, 6, 16, 12, 28, 1
2, 13, 10, 8, 13, 9, 21, 11, 10, 12, 11, 9, 35, 41, 11, 9, 6, 10, 14, 12, 30, 13
, 7, 7, 11]
[7, 15, 11, 27, 20, 10, 23, 13, 8, 11, 12, 9, 23, 8, 6, 6, 7, 9, 6, 15, 7, 22, 6
, 9, 13, 8, 16, 11, 8, 19, 26, 6, 12, 8, 7, 12, 13, 10, 38, 16, 6, 16, 12, 28, 1
2, 13, 10, 8, 13, 9, 21, 11, 10, 12, 11, 9, 35, 41, 11, 9, 6, 10, 14, 12, 30, 13
, 7, 7, 11, 18]
can't invoke "event" command: application has been destroyed
    while executing
"event generate $w <<ThemeChanged>>"
    (procedure "ttk::ThemeChanged" line 6)
```

It seems that the perceptron finds the solution under 15 tries generally when there are only 20 data points.

h) Both *N* and *d* seem to both have positive correlation with respect to time. As the number of data points increases, the amount of iterations the perceptron takes to find a solution increases, as seen by the 1000 data points experiments. As the number of dimensions increase the time it takes for the perceptron to calculate its next move seems to increase. This was noticeable when the 10 dimensions were used to try and calculate the 1000 datapoint perceptron. When there were only two dimensions it seemed to find the solution much faster.