Homework 1

Amy Pitts

2/5/19

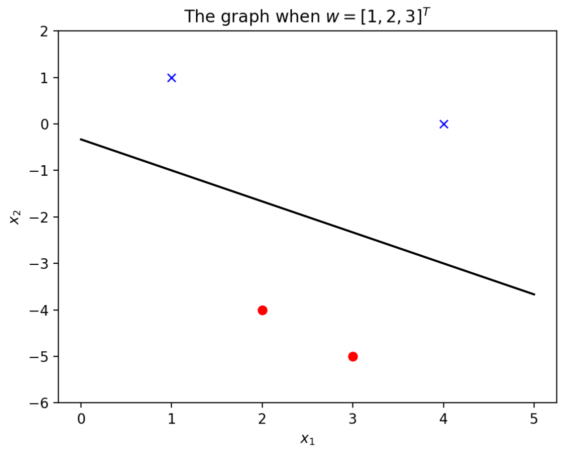
1 Problem 1.2

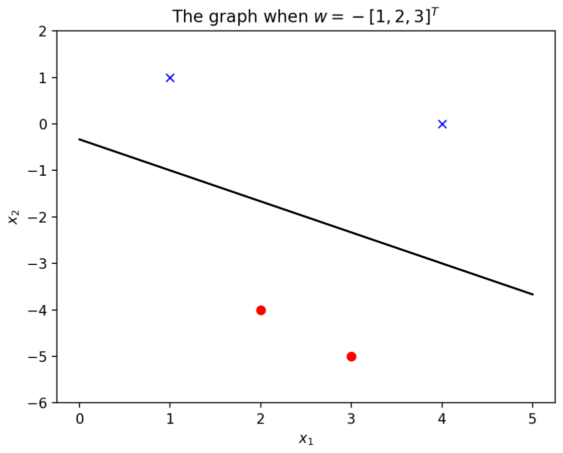
Consider the perceptron in two dimensions: $h(x) = sign(w^T x)$ where $w = [w_0, w_1, w_2]^T$ and $x = [1, x_1, x_2]^t$. Technically, x has three coordinates, but we call this perception two-dimensional because the first coordinates is fixed at 1.
(a) Show that the regions on the plane where $h(x) = +1$ and $h(x) = -1$ are separated by a line. If we express this line by the equation $x_2 = ax_1 + b$ what are the slope a and intercept b in terms of w_0, w_1, x_2?

Solution: If we have $h(x) = 1$ and $h(x) = -1$ this implies that for $h(x) = 1$ we have $w^T x > 0$ and with $h(x) = -1$ we have $w^T x < 0$ so the separation between the two is just the x-axis or when $w^T x = 0$.
For the equation $x_2 = ax_1 + b$ we have that $a = -\frac{w_1}{w_2}$ and $b = -\frac{w_0}{w_2}$

(b) Draw a picture for the cases $w = [1, 2, 3]^T$ and $w = -[1, 2, 3]^T$.
In more than two dimensions, the $+1$ and -1 regions are separated by a hyperplane, the generalization of a line.

Solution:





```
import numpy as np
import matplotlib.pyplot as plt

def pltPer(X, y, W):
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2], 'ro')
        else:
            plt.plot(X[n,1],X[n,2], 'bx')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(0,5)
    plt.plot(l, m*l+b, 'k-')
```

```

#Postive W
X = np.array([[1,1,1],
              [1,4,0],
              [1,2,-4],
              [1,3,-5]])
y = np.array([1, 1, -1, -1])
W = np.array([1,2,3]) #w0w1w2
pltPer(X,y,W)

plt.ylim(-6,2)
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.title(r'The graph when $w=[1,2,3]^T$')
plt.savefig('hwplot_postive_w.pdf', bbox_inches='tight')
plt.show()

#Negative W
X = np.array([[1,1,1],
              [1,4,0],
              [1,2,-4],
              [1,3,-5]])
y = np.array([1, 1, -1, -1])
W = np.array([-1,-2,-3]) #w0w1w2
pltPer(X,y,W)

plt.ylim(-6,2)
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.title(r'The graph when $w=-[1,2,3]^T$')
plt.savefig('hwplot_negative_w.pdf', bbox_inches='tight')
plt.show()
```

2 Problem 1.4

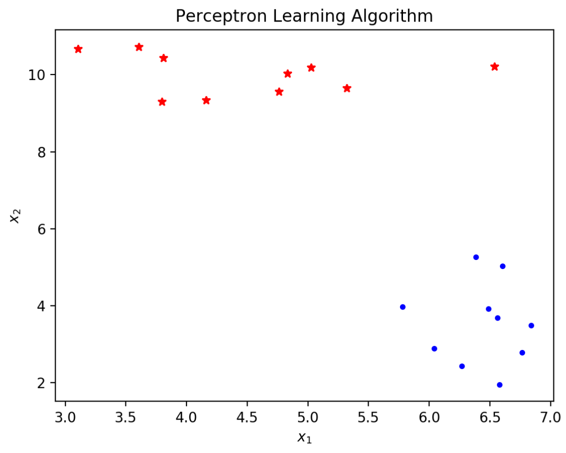
In Exercise 1.4 we use an artificial data set to study the perceptron learning algorithm. This problem leads you to explore the algorithm further with data sets of different sizes and dimensions.

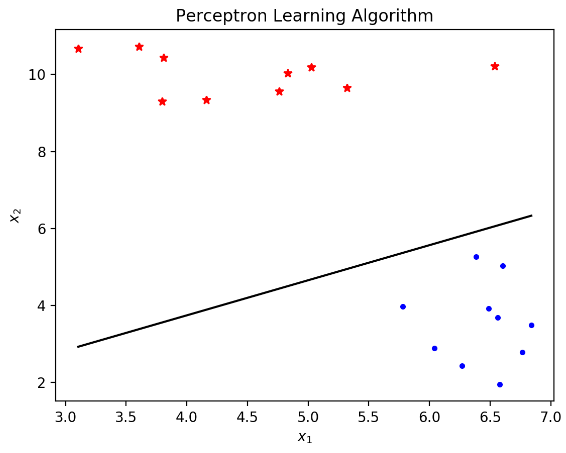
For the following question I will be using and modifying this code:

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.datasets.samples_generator import make_blobs

def pltPer(X, y, W):
    f = plt.figure()
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1])) #could just use max and min
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")
    plt.show()

# Some attempt to do the PLA
def main():
    N = 1000

    # data
    X, y = make_blobs(n_samples=N, centers=2, n_features=2)
    y[y==0] = -1 #changing all the 0 to negative one
    X = np.append(np.ones((N,1)), X, 1) #adding a column of ones


    # initialize the weigths to zeros
    w = np.zeros(3) #change this to 11 for 10 dims
    it = 0
    pltPer(X,y,w) # initial solution (bad!)

    # Iterate until all points are correctly classified
    while classification_error(w, X, y) != 0:
        it += 1
        # Pick random misclassified point
        x, s = choose_miscl_point(w, X, y)
        # Update weights
        w += s*x

    pltPer(X,y,w) #commit this out before running 10 dims
    print("Total interations:" + str(it))

def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    print(err_cnt)
    return err_cnt

def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

main()
```

(a): Generate a linearly separable data set of size 20 as indicated in Exercise 1.4. Plot the examples

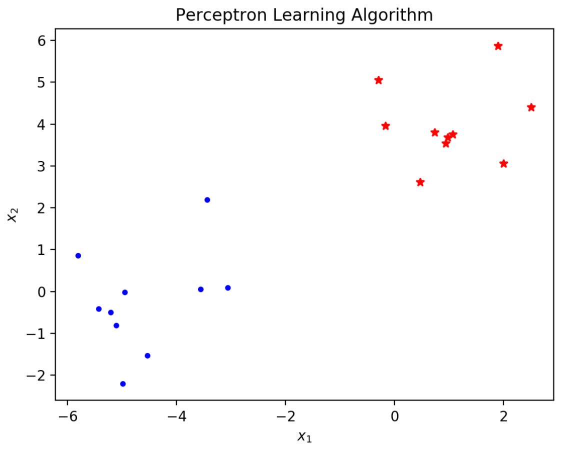
$\{(x_n, y_n)\}$ as well as the target function f on a plane. Be sure to mark the examples from different classes differently, and add labels to the axes of the plot.

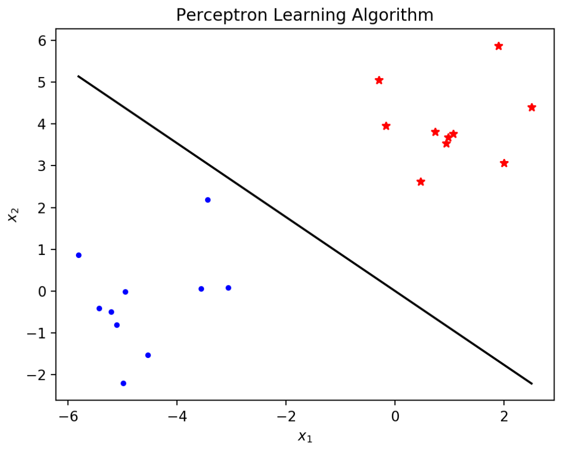




The program took 5 iterations to properly split up the two groups.

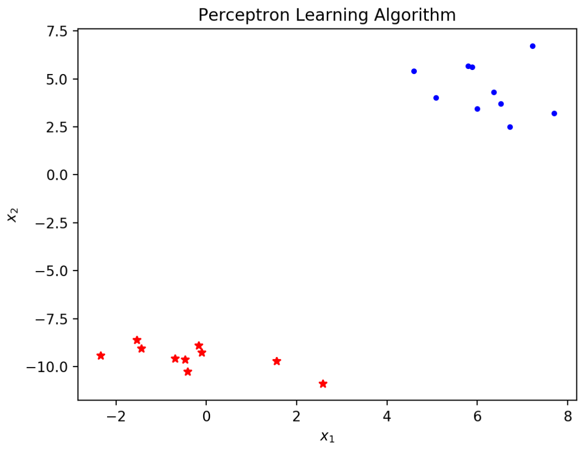
(b): Run the perceptron learning algorithm on the data set above. Report the number of updates that the algorithm takes before converging. Plot the examples $\{(x_n, y_n)\}$, and the final hypothesis g in the same figure. Comment on whether f is close to g.

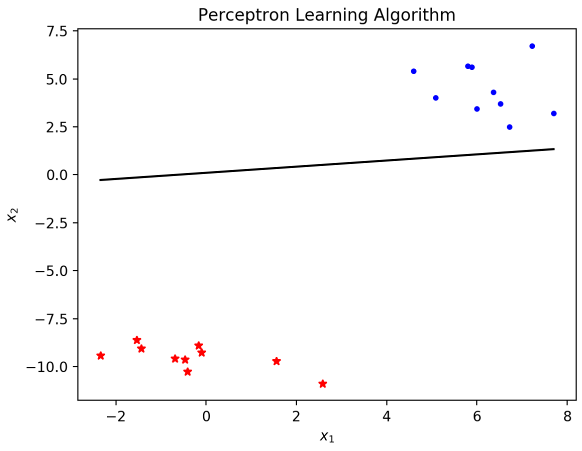




The program took 2 iterations to properly split up the two groups. We are unable to tell if f is close to g because we do not know what f is.

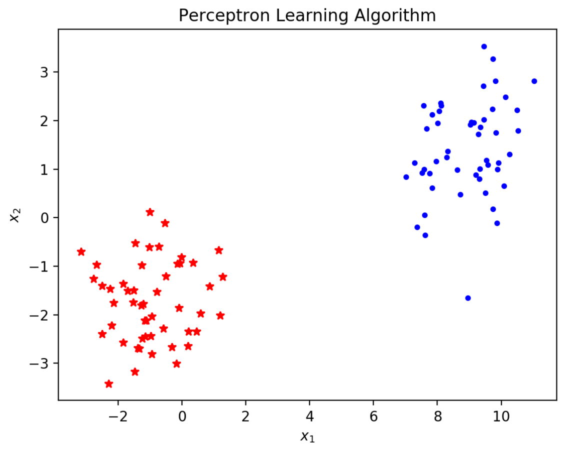
(c): Repeat everything in (b) with another randomly generated data set of size 20. Compare your results with (b).

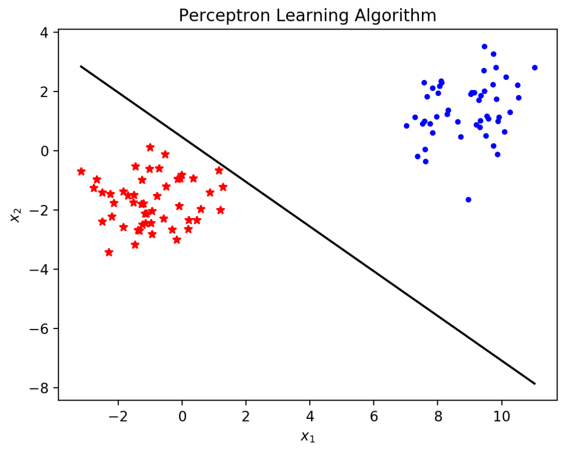




The program took 1 iteration to properly split up the two groups. We are unable to tell if f is close to g because we do not know what f is.

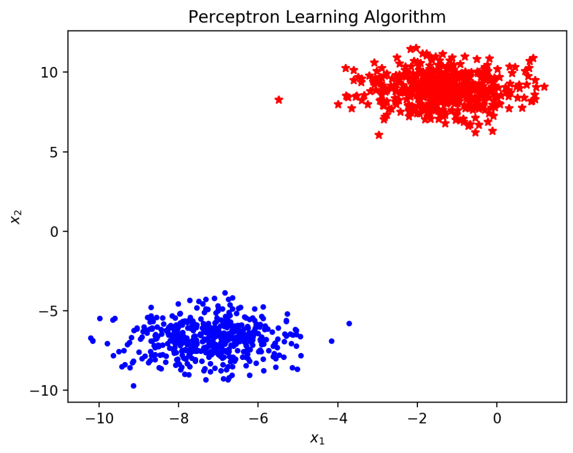
(d): Repeat everything in (b) with another randomly generated data set of size 100. Compare your results with (b).

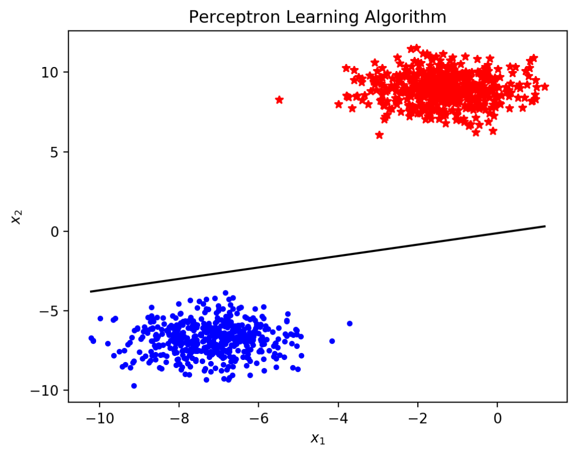




The program took 5 iteration to properly split up the two groups.

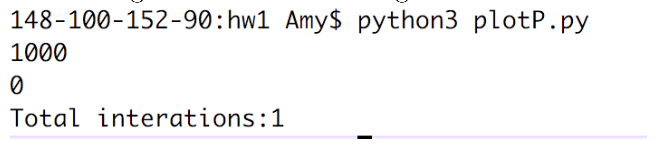
(e): Repeat everything in (b) with another randomly generated data set of size 1,000. Compare your results with (b).





The program took 1 iteration to properly split up the two groups. I thought it was going to take more iterations but the data generator is making perfectly separated groups making it super easy to separate. Also, after running the program a couple of times to see what happens I found that I either get stuck in a loop of never being able to find the perfect line or the groups are easily separated and it take 1 to 2 iterations.

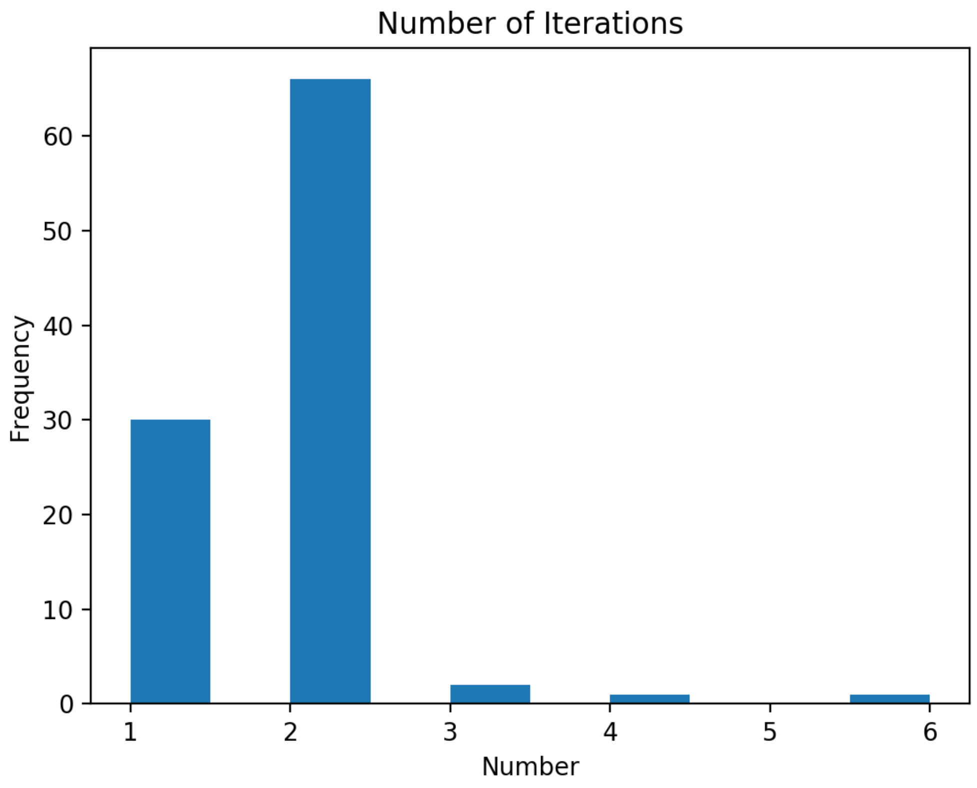
(f): Modify the algorithm such that it takes $x_n \in \mathbb{R}^{10}$ instead of $\mathbb{R}^2$. Randomly generate a linearly separable data set of size 1,000 with $x_n \in \mathbb{R}^{10}$ and feed the data set to the algorithm. How many updates does the algorithm take to converge?



```
148-100-152-90:hw1 Amy$ python3 plotP.py
1000
0
Total interations:1
```

The algorithm took one update to converge.

(g): Repeat the algorithm on the same data set as (f) for 100 experiments. In the iterations of each experiment, pick $x(t)$ randomly instead of deterministically. Plot a histogram for the number of updates that the algorithm take to converge.



```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.datasets.samples_generator import make_blobs

# Some attempt to do the PLA
itlist = []
for x in range(100):
    def main():
        N = 1000

        # data
        X, y = make_blobs(n_samples=N, centers=2, n_features=10)
        y[y==0] = -1 #changing all the 0 to negative one
        X = np.append(np.ones((N,1)), X, 1) #adding a column of ones

        # initialize the weigths to zeros
        w = np.zeros(11)
        it = 0

        # Iterate until all points are correctly classified
        while classification_error(w, X, y) != 0:
            it += 1
            # Pick random misclassified point
            x, s = choose_miscl_point(w, X, y)
            # Update weights
            w += s*x
        itlist.append(it) #saving the iterations

    def classification_error(w, X, y):
        err_cnt = 0
        N = len(X)
        for n in range(N):
            s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
            if y[n] != s:
                err_cnt += 1
        return err_cnt

    def choose_miscl_point(w, X, y):
        mispts = []
        # Choose a random point among the misclassified
        for n in range(len(X)):
            if np.sign(w.T.dot(X[n])) != y[n]:
                mispts.append((X[n], y[n]))
        return mispts[random.randrange(0,len(mispts))]

    main()

plt.hist(itlist) #plotting a histogram
plt.title("Number of Iterations")
plt.xlabel("Number")
plt.ylabel("Frequency")
plt.show()
```

(h): Summarize your conclusions with respect to accuracy and running time as a function of N and d.

I found that the generated data if broken up takes about 1 to 2 iterations to properly split the data. However, when there were only two dimensions there was a more likely chance that a data set was generated that was not able to be split up resulting in an endless loop. Running the 10-dimensional data set there was less of a chance that the data would not be able to be split. Also when N was increased there was a higher chance that a data set was produced that was not able to be split by a line. I wonder what would happen if we used a different data generator.