Jack Barry

9/29/16

AI CMPT 404

Homework 2

Problem 2.1

As we discussed in class it suffices to use the equation $epsilon(M, N, delta) = sqrt\left(\left(\frac{1}{2N}\right)\ln\left(\frac{2M}{delta}\right)\right) \leq epsilon$ and our givens to solve for N. For M = 1, epsilon = 0.05, and delta = 0.03 we note that the equation can be written with these values as $epsilon(1, N, 0.03) = sqrt\left(\left(\frac{1}{2N}\right)\ln\left(\frac{2(1)}{0.03}\right)\right) \leq 0.05$. After squaring both sides and algebraically simplifying the equation we get a result of $840 \leq N\ or\ N \geq 840$. This process and equation proves useful in finding how much input data is needed to reach a particular level of certainty.

Problem 2.11

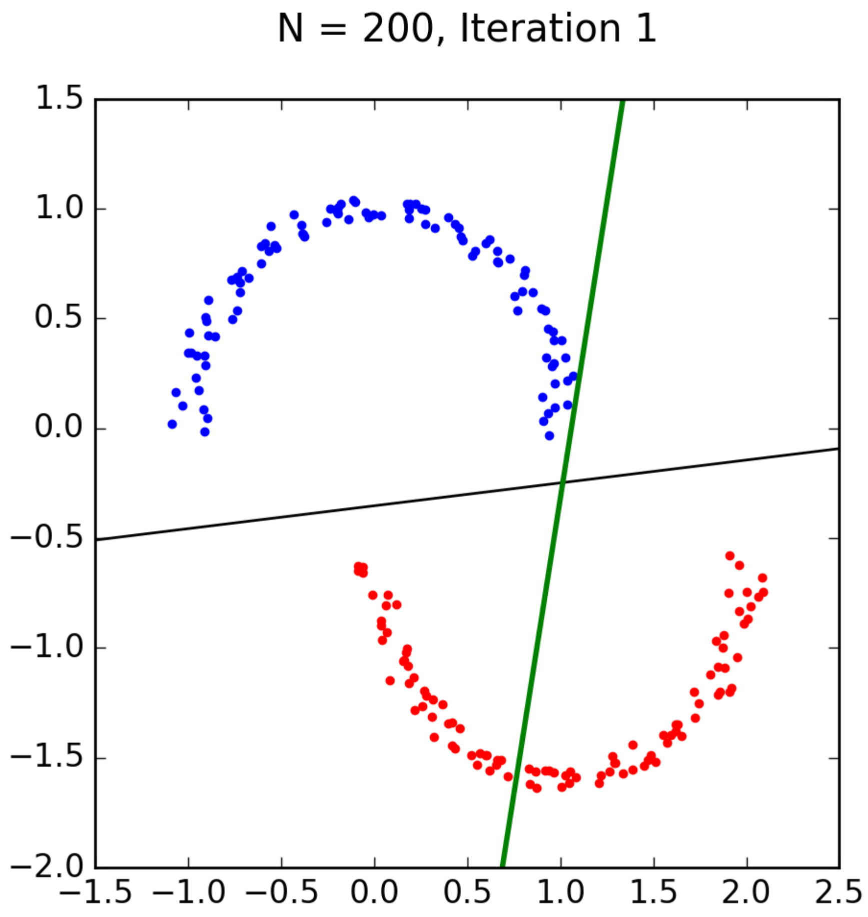
By equation 2.1 we know that $E_{out}(g) \leq\ E_{in}(g) + sqrt(\left(\left(\frac{1}{2N}\right)\ln\left(\frac{2M}{delta}\right)\right)$. For this problem we are given that M = 1, and since we know that we need 90% confidence we have delta = 0.1, and finally we will compute the bound for E_{out} with both N=100 and N = 10,000 and compare out results. For N = 100 we simply plug out given values into the equation and solve maintaining operations on the right side of the equation. Completing operations yields a result of $E_{out}(g) \leq\ E_{in}(g) + 0.1224$. For the case when N = 10,000 out calculation is only changed for the N so we compute as we did before and receive a result of $E_{out}(g) \leq\ E_{in}(g) + 0.01224$. Comparing the two cases the results are as expected and changed by the factor of the input size N. The larger the input size N the smaller the bound for $E_{out}(g)$.

Problem 2.12

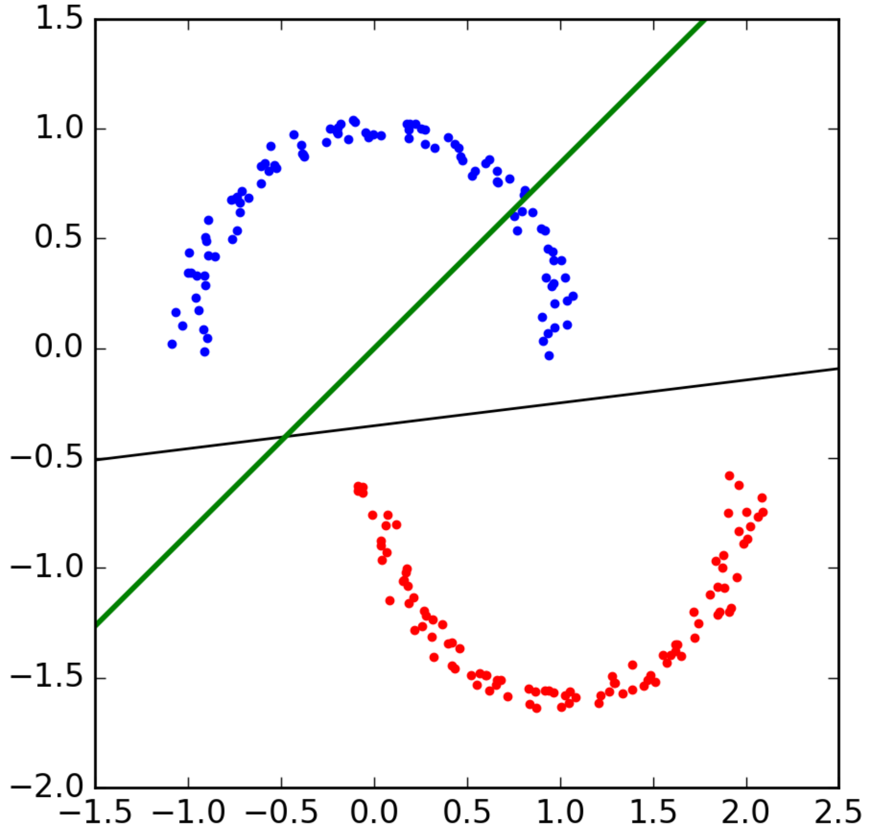
Problem 2.12 is similar to problem 2.1 in the sense that we are finding a solution for an acceptable sample size that will result in a certain level of confidence. However, this problem has increased the complexity of the VC dimension to 10 instead of 1. So to solve this problem we will say it suffices to make $epsilon(M, N, delta) = sqrt\left(\left(\frac{1}{2N}\right)\ln\left(\frac{2M}{delta}\right)\right) \leq epsilon.$ And using our given information we derive our input values to get the following $epsilon(10, N, 0.05) = sqrt\left(\left(\frac{1}{2N}\right)\ln\left(\frac{2(10)}{0.05}\right)\right) \leq 0.05$. After solving for N we get a result of $1{,}198 \leq N\ which\ means\ N \geq 1{,}198$. So therefore to achieve 95% confidence in this example we need a sample size of 1,198.

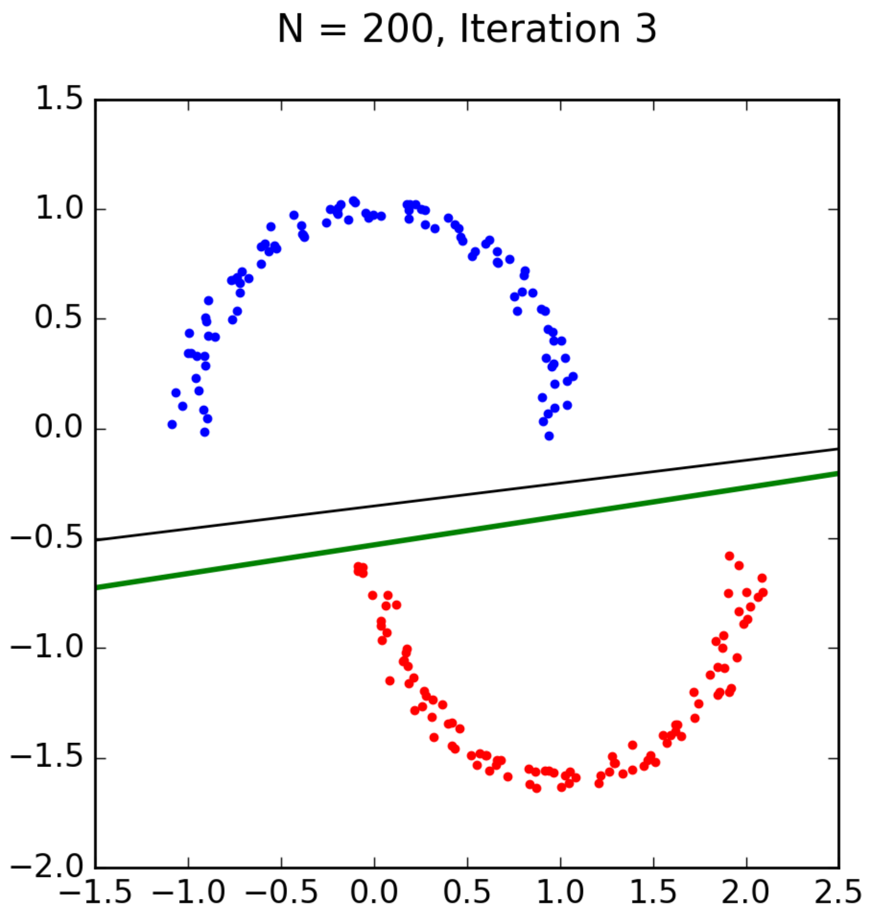
Problem 3.1

This was an interesting problem to apply the PLA to because the data was so specifically linearly separable. Combining the code you provided to make the semi circles and the PLA code from homework and running on N = 2000 caused my program to crash, I tired different input sizes and the eventually N = 200 yielded a result as follows. I am still looking into why the program crashed, my computer does not have the most impressive specs but it should still be able to handle N = 2000.

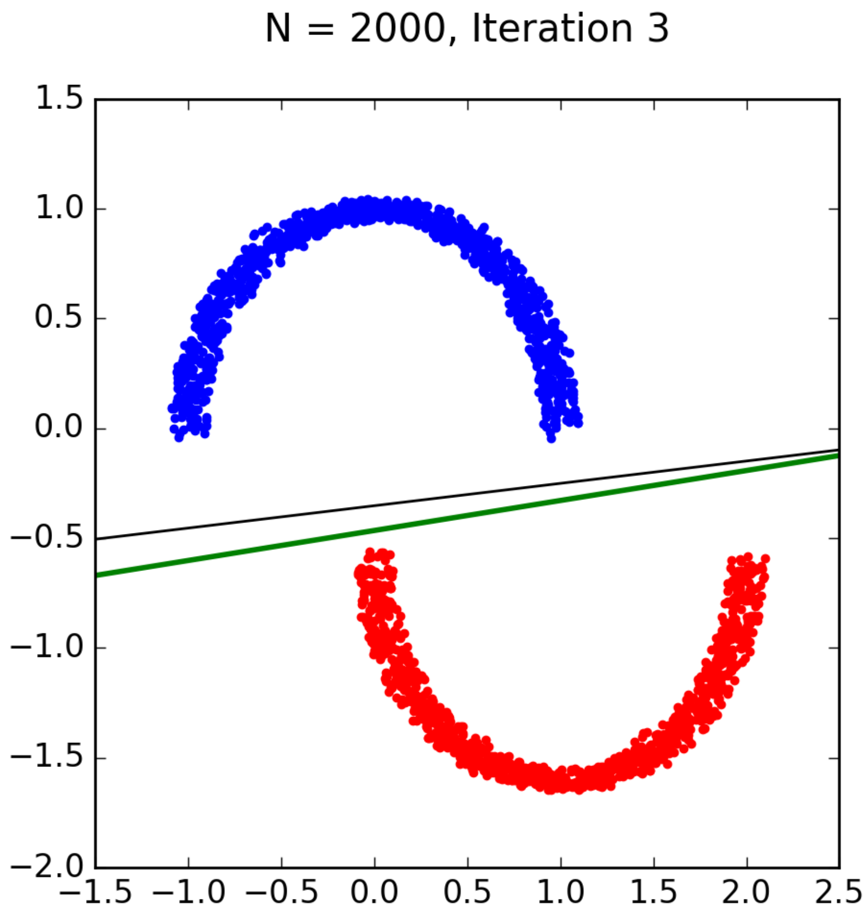


N = 200, Iteration 2

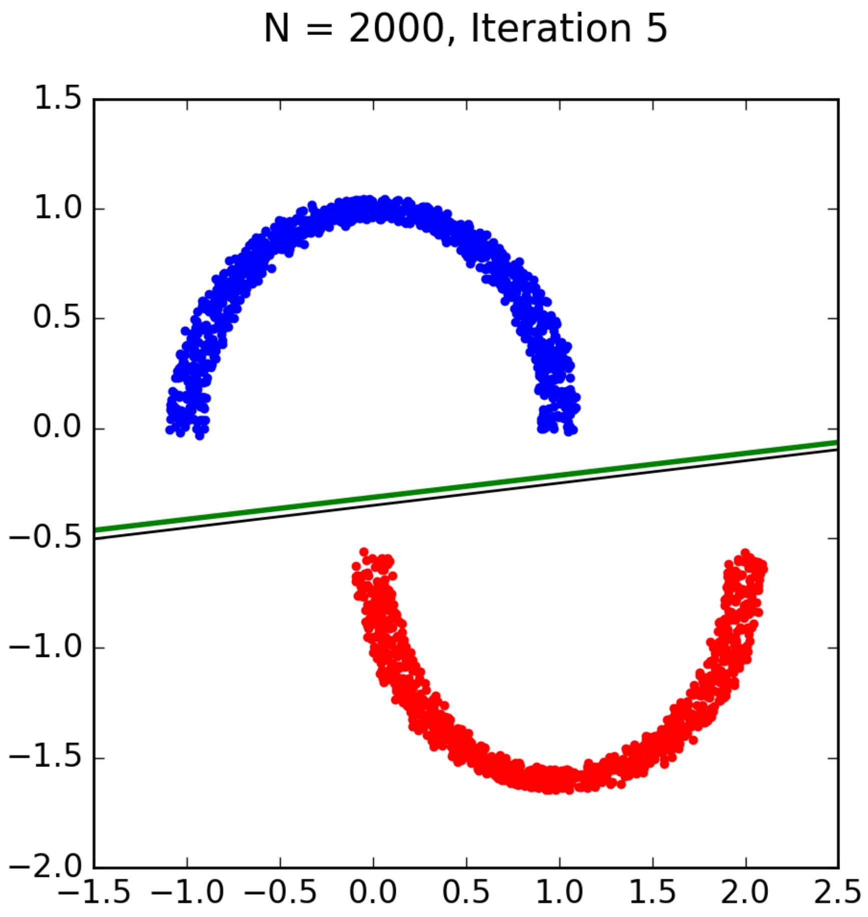




Finding a solution on the 3rd iteration is rather quick compared to some of the data we saw from homework 1. I suspect this is due to the minimizing of points down to 200 and the fact that the data is specifically linearly separable. More interestingly moving on to implement linear regression for classification I saw that I was able to handle N = 2000 which I was happy about. I suspect linear regression is more efficient at finding a solution in a smaller number of iterations because even with N = 2000 I was still able to find a solution in 5 or less iterations which is close to that of the previous PLA implementation. The results of the linear regression classification are as follows.



My first successful run had the PLA find a solution that differs from the regression. The PLA took 3 iterations to converge, the cases in 3 iterations as seen above. I ran it again and the PLA went on to 5 iterations to converge as seen below and reached a result that is very close to the result of the Linear



Regression.

The code for the first implantation of the PLA is below the code for the second run is the code that you provided for us.

```
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 = [random.uniform(-1, 1) for i in range(4)]
    self.V = np.array([xB * yA - xA * yB, yB - yA, xA - xB])
    self.X = self.generate_points(N)

  def generate_points(self, N):
    X, y = self.make_semi_circles(n_samples=N, sep=5)
    bX = []
    for k in range(N):
        x1, x2 = [random.uniform(-1, 1) for k in range(2)]
        x = np.array([1, x1, x2])
        s = int(np.sign(self.V.T.dot(x)))
        bX.append((np.concatenate(([1], X[k, :])), y[k]))
    return bX
def  make_semi_circles(self, n_samples=2000, thk=5, rad=10, sep=5, plot=True):
    """Make two semicircles circles
    A simple toy dataset to visualize classification algorithms.
    Parameters
    ----------
    n_samples : int, optional (default=2000)
        The total number of points generated.
    thk : int, optional (default=5)
        Thickness of the semi circles.
    rad : int, optional (default=10)
        Radious of the circle.
    sep : int, optional (default=5)
        Separation between circles.
    plot : boolean, optional (default=True)
        Whether to plot the data.
    Returns
    -------
    X : array of shape [n_samples, 2]
        The generated samples.
    y : array of shape [n_samples]
        The integer labels (-1 or 1) for class membership of each sample.
    """

    noisey = np.random.uniform(low=-thk / 100.0, high=thk / 100.0, size=(n_samples // 2))

    noisex = np.random.uniform(low=-rad / 100.0, high=rad / 100.0, size=(n_samples // 2))

    separation = np.ones(n_samples // 2) * ((-sep * 0.1) - 0.6)

    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out

    # generator = check_random_state(random_state)

    outer_circ_x = np.cos(np.linspace(0, np.pi, n_samples_out)) + noisex
    outer_circ_y = np.sin(np.linspace(0, np.pi, n_samples_out)) + noisey
    inner_circ_x = (1 - np.cos(np.linspace(0, np.pi, n_samples_in))) + noisex
    inner_circ_y = (1 - np.sin(np.linspace(0, np.pi, n_samples_in)) - .5) + noisey + separation

    X = np.vstack((np.append(outer_circ_x, inner_circ_x),
                   np.append(outer_circ_y, inner_circ_y))).T
    y = np.hstack([np.ones(n_samples_in, dtype=np.intp) * -1,
                   np.ones(n_samples_out, dtype=np.intp)])

    if plot:
        plt.plot(outer_circ_x, outer_circ_y, 'r.')
        plt.plot(inner_circ_x, inner_circ_y, 'b.')
        plt.show()

    return X, y

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)
    if save:
        if not mispts:
            plt.title('N = %s' % (str(len(self.X))))
        else:
            plt.title('N = %s with %s test points' \
                      % (str(len(self.X)), str(len(mispts))))
        plt.savefig('p_N%s' % (str(len(self.X))), \
                    dpi=200, bbox_inches='tight')

def classification_error(self, vec, pts=None):
    # Error defined as fraction of misclassified points
    if not pts:
        pts = self.X
        M = len(pts)
        n_mispts = 0
        for x, s in pts:
            if int(np.sign(vec.T.dot(x))) != s:
                n_mispts += 1
        error = n_mispts / float(M)
        return error

    def choose_miscl_point(self, vec):
        # Choose a random point among the misclassified
        pts = self.X
        mispts = []
        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')
        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():
    p = Perceptron(1000)
    p.pla(save = True)
    p.plot()

main()
```