HomeWork 02

Deep Dand

May 2, 2018

Abstract

Problems 2.1, 2.11, 2.12, 3.1 and 3.2

1 Problem 2.1

1.1 a

With $m = 1, \epsilon = 0.05$ and $\delta = 0.03$.
Equation,

$$\epsilon(M, N, \delta) = \sqrt{\frac{1}{2N} ln \frac{2M}{\delta}}$$

with values,

$$0.05 = \sqrt{\frac{1}{2N} ln \frac{2*1}{0.03}}$$

taking squares on both sides,

$$0.0025 = \frac{1}{2N} ln \frac{2*1}{0.03}$$

$$0.0025 = \frac{1}{2N} * 4.1997$$

$$N = 839.94 \approx 840$$

840 samples are required to make $\epsilon \leq 0.05$ with given variable values.

1.2 b

With $m = 100, \epsilon = 0.05$ and $\delta = 0.03$.
Equation,

$$\epsilon(M, N, \delta) = \sqrt{\frac{1}{2N} ln \frac{2M}{\delta}}$$

with values,

$$0.05 = \sqrt{\frac{1}{2N} ln \frac{2*100}{0.03}}$$

taking squares on both sides,

$$0.0025 = \frac{1}{2N} ln \frac{200}{0.03}$$

$$0.0025 = \frac{1}{2N} * 8.8048$$

$$N = 1760.98 \approx 1761$$

1761 samples are required to make $\epsilon \leq 0.05$ with given variable values.

1.3 c

With $m = 10000, \epsilon = 0.05$ and $\delta = 0.03$.
Equation,

$$\epsilon(M, N, \delta) = \sqrt{\frac{1}{2N} ln \frac{2M}{\delta}}$$

with values,

$$0.05 = \sqrt{\frac{1}{2N} ln \frac{2 * 10000}{0.03}}$$

taking squares on both sides,

$$0.0025 = \frac{1}{2N} ln \frac{20000}{0.03}$$

$$0.0025 = \frac{1}{2N} * 13.41$$

$$N = 2682$$

2682 samples are required to make $\epsilon \leq 0.05$ with given variable values.

2 Problem 2.11

2.1 N=100

$m_H(N) = N + 1$ and $d_{vc} = 1, N = 100\delta = 0.9$
The equation,

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} ln\left(\frac{4((2N)^{d_{vc}} + 1)}{\delta}\right)}$$

substituting values,

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{100} ln\left(\frac{4((2 * 100)^1 + 1)}{0.9}\right)}$$

$$E_{out}(g) \leq E_{in}(g) + \sqrt{0.08 ln\left(\frac{801}{0.9}\right)}$$

$$E_{out}(g) \leq E_{in}(g) + \sqrt{0.5432}$$

$$E_{out}(g) \leq E_{in}(g) + 0.737$$

2.2 N=10000

$m_H(N) = N + 1$ and $d_{vc} = 1, N = 10000\delta = 0.9$
The equation,

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} ln\left(\frac{4((2N)^{d_{vc}} + 1)}{\delta}\right)}$$

substituting values,

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{10000} ln\left(\frac{4((2 * 10000)^1 + 1)}{0.9}\right)}$$

$$E_{out}(g) \leq E_{in}(g) + \sqrt{0.0008 ln\left(\frac{80001}{0.9}\right)}$$

$$E_{out}(g) \leq E_{in}(g) + \sqrt{0.0091}$$

$$E_{out}(g) \leq E_{in}(g) + 0.095$$

3 2.12

$$N \geq \frac{8}{\epsilon^2} ln\left(\frac{4((2N)^{d_{vc}}+1)}{\delta}\right)$$

Code Begins

```
import math
dvc=10.0
d=0.95
e=0.05
e2 = math.pow(e,2) #calculating e^2
#print "e power 2:",e2
ede=8/e2 #calculating 8/e^2
#print "ede",ede
n=100.0
for n in range(100,1000000):
  tn = 2*n #calculating 2*N
  eqt = ede * math.log((4*(math.pow(tn,dvc)+1))/d) #the main equation
  eqt = int(eqt)
  #print "eqt value",eqt,"n = ",n
  if( int(n) >= eqt ): #comparing equation with N
    break
  else:
    n = n+50
#print "eqt value: ",eqt," n value: ",n
print "We need ",n," samples to get error < 0.05"
```

Code Ends

The output - We need 442809 samples to get error ¡ 0.05

4 3.1

Double Semi-Circles.

4.1 a

Code Begins

```
#pocket perceptron with semi-circles dataset and weight = 0

import time
import numpy as np
import random
import os, subprocess
import matplotlib.pyplot as plt
from makeSemiCircles import make_semi_circles

class Perceptron:

    def __init__(self, N):
        # Random linearly separated data
        # # # # # # # # # random.seed(0)
        #xA,yA,xB,yB = [random.uniform(-1, 1) for i in range(4)]
        #x1A,x2A,x3A,...,x1B,x2B, = [random(1,11).uniform(-1, 1) for i in range(4)]
        #self.V = np.array([xB*yA-xA*yB, yB-yA, xA-xB])
        self.V = (np.random.rand(3)*2)-1
        # self.V = np.array([0.25, 0.5, 0.75])
```

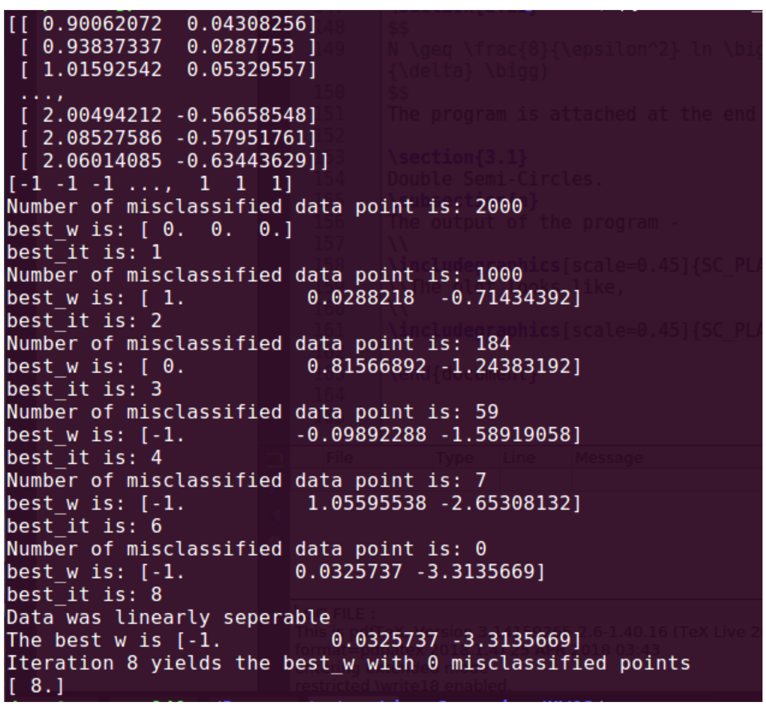
```
        self.X= self.generate_points(N)




    def generate_points(self, N):
        X = []
        x, s = make_semi_circles(n_samples=N, thk=5, rad=10, sep=5, plot=False)
        x = np.insert(x,0,1,axis=1)# added bias of 1
        X = [[x[i], s[i]] for i in range(len(x))]
        return X
        #print(X)
# [
# ([b, x1, x2], s),
# ([b, x1.2, x2.2], s2),
# ]


    def plot(self, mispts=None, vec=None, save=False):
        fig = plt.figure(figsize=(8,8))
        plt.xlim(-2.1,2.1)
        plt.ylim(-2.1,2.1)
        #V = self.V
        #a, b = -V[1]/V[2], -V[0]/V[2]
        l = np.linspace(-3.1,3.1)
        #plt.plot(l, a*l+b, 'k-')
        cols = {1: 'g', -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], 'rx')
        if vec.any() != None:
            aa, bb = -vec[1]/vec[2], -vec[0]/vec[2] #idk what aa and bb are
            plt.plot(l, aa*l+bb, 'k-', 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
        myErr = 0
        for x,s in pts:
            myErr += abs(s - np.sign(vec.T.dot(x)))
            if np.sign(vec.T.dot(x)) != s:
                n_mispts += 1
        error = n_mispts / float(M)
        # print(error)
        # print(myErr)
```

```
        return error

    def choose_miscl_point(self, mispts):
        # Choose a random point among the misclassified
        if not mispts:
            return None,None
        return mispts[random.randrange(0,len(mispts))]


    def miscl_points_calc(self, vec):
        pts = self.X
        mispts = []
        for x,s in pts:
            if np.sign(vec.T.dot(x)) != s:
                mispts.append((x, s))

        return mispts

    def pla(self, save=False):
        # Initialize the weigths to zeros
        w = np.zeros(3)
        #w = self._W
        best_w = None
        best_mispts = None
        best_it = 0
        X, N = self.X, len(self.X)
        it = 0
        # Iterate until all points are correctly classified
        for i in range(50):
            it += 1
            mispts = self.miscl_points_calc(vec=w)

            # Pick random misclassified point
            x, s = self.choose_miscl_point(mispts=mispts)

            #if i % 5 == 0 and save:
            if save:
                self.plot(mispts=mispts, vec=w)
                plt.title('N = %s, Iteration %s, mispts %s\n' % (str(N),str(it), str(len(mispts))
                    ↪ ))
                plt.savefig('p_N%s_it%s' % (str(N),str(it)), dpi=200, bbox_inches='tight')

            #pocket parts
            if best_mispts is None or len(mispts)<best_mispts:
                best_mispts = len(mispts)
                best_w = w
                best_it = it
                print("Number of misclassified data point is: {}".format(best_mispts))
                print("best_w is: {}".format(best_w))
                print("best_it is: {}".format(best_it))

            if x is None:
                print("Data was linearly seperable")
                break

            # Update weights
            w += s*x
```

```
        self.w = w
        self.best_w = best_w
        self.best_it = best_it
        print("The best w is {}".format(best_w))
        print("Iteration {} yields the best_w with {} misclassified points".format(best_it, best_mispts))

        self.plot(mispts=mispts, vec=w)
        plt.title('N = %s, Iteration %s, misclassified points %s\n' % (str(N),str(it), str(len(mispts))))
        plt.savefig('SC_PLA%s_it%s' % (str(N),str(it)), dpi=200, bbox_inches='tight')

        return it

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

def main():
    it = np.zeros(1)
    for x in range(0, 1):
        p = Perceptron(2000)
        it[x] = p.pla(save=False)
        print(it)

main()
```

Code Ends This is the output with PLA

$$w = 0$$



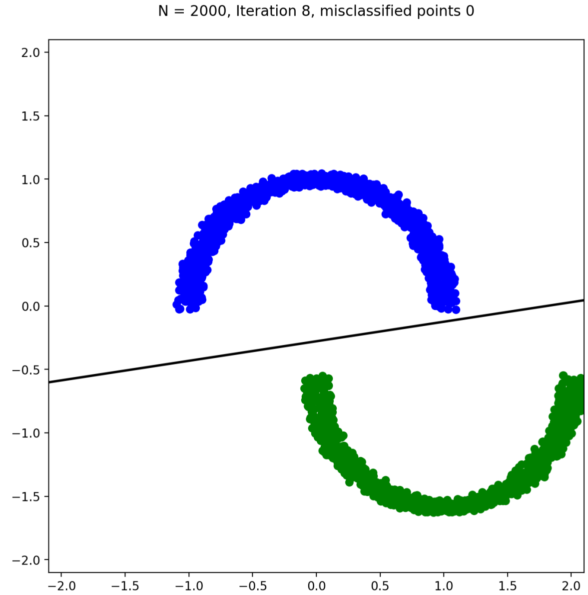
```
[[ 0.90062072  0.04308256]
 [ 0.93837337  0.0287753 ]
 [ 1.01592542  0.05329557]
 ...,
 [ 2.00494212 -0.56658548]
 [ 2.08527586 -0.57951761]
 [ 2.06014085 -0.63443629]]
[-1 -1 -1 ...,  1  1  1]
Number of misclassified data point is: 2000
best_w is: [ 0.  0.  0.]
best_it is: 1
Number of misclassified data point is: 1000
best_w is: [ 1.          0.0288218  -0.71434392]
best_it is: 2
Number of misclassified data point is: 184
best_w is: [ 0.          0.81566892 -1.24383192]
best_it is: 3
Number of misclassified data point is: 59
best_w is: [-1.         -0.09892288 -1.58919058]
best_it is: 4
Number of misclassified data point is: 7
best_w is: [-1.          1.05595538 -2.65308132]
best_it is: 6
Number of misclassified data point is: 0
best_w is: [-1.         0.0325737 -3.3135669]
best_it is: 8
Data was linearly seperable
The best w is [-1.         0.0325737 -3.3135669]
Iteration 8 yields the best_w with 0 misclassified points
[ 8.]
```

The output here shows two vectors at the beginning which are -X and Y In the first iteration the weights are initialized to 0 with np.zero function and the program displays only those iteration details which has better weights than the previous weight results. In the next iterations, 1 though 8, the weights are adjusted and number of misclassified data points are constantly decreasing from 2000 to 0.

In iteration 8 we get all the datapoints classified well and are linearly seperable.
The best weight is -

$$w = [-1, 0.0325737, -3.3135669]$$

The plot looks like,



The plot here shows data points linearly seperated with a straight line.

4.2 b

```
#linear regressions perceptron with semi-circles dataset

import time
import numpy as np
import random
import os, subprocess
import matplotlib.pyplot as plt
from makeSemiCircles import make_semi_circles

class Perceptron:

    def __init__(self, N):
        #self.V = (np.random.rand(3)*2)-1
        self.X,self._W = self.generate_points(N)





    def generate_points(self, N):
        X = []
        X_inv = []
        Y_inv = []

        x, s = make_semi_circles(n_samples=N, thk=5, rad=10, sep=5, plot=False)
        x = np.insert(x,0,1,axis=1)# added bias of 1
        X = [[x[i], s[i]] for i in range(len(x))]
        X_inv = np.array(x) # used to calculate pseudo inv
```

```
        Y_inv = np.array(s) # used to calculate pseudo inv

        _W = np.linalg.pinv(X_inv.T.dot(X_inv)).dot(X_inv.T).dot(Y_inv)


        return X,_W





    def plot(self, mispts=None, vec=None, save=False):
        fig = plt.figure(figsize=(8,8))
        plt.xlim(-2.1,2.1)
        plt.ylim(-2.1,2.1)
        #V = self.V
        #a, b = -V[1]/V[2], -V[0]/V[2]
        l = np.linspace(-3.1,3.1)
        #plt.plot(l, a*l+b, 'k-')
        cols = {1: 'g', -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], 'rx')
        if vec.any() != None:
            aa, bb = -vec[1]/vec[2], -vec[0]/vec[2] #idk what aa and bb are
            plt.plot(l, aa*l+bb, 'k-', 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')

    #is this actually used?
    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
        myErr = 0
        for x,s in pts:
            myErr += abs(s - np.sign(vec.T.dot(x)))
            if np.sign(vec.T.dot(x)) != s:
                n_mispts += 1
        error = n_mispts / float(M)
        # print(error)
        # print(myErr)
        return error

    def choose_miscl_point(self, mispts):
        # Choose a random point among the misclassified
        if not mispts:
            return None,None
        return mispts[random.randrange(0,len(mispts))]
```

```
def miscl_points_calc(self, vec):
    pts = self.X
    mispts = []
    for x,s in pts:
        if np.sign(vec.T.dot(x)) != s:
            mispts.append((x, s))

    return mispts

def pla(self, save=False):
    # Initialize the weigths to zeros
    #w = np.zeros(3)
    w = self._W
    best_w = None
    best_mispts = None
    best_it = 0
    X, N = self.X, len(self.X)
    it = 0
    # Iterate until all points are correctly classified
    for i in range(50):
        it += 1
        mispts = self.miscl_points_calc(vec=w)

        # Pick random misclassified point
        x, s = self.choose_miscl_point(mispts=mispts)

        #if i % 5 == 0 and save:
        if save:
            self.plot(mispts=mispts, vec=w)
            plt.title('N = %s, Iteration %s, misclassified points %s\n' % (str(N),str(it),
                ↪ str(len(mispts))))
            plt.savefig('p_N%s_it%s' % (str(N),str(it)), dpi=200, bbox_inches='tight')

        #pocket parts
        if best_mispts is None or len(mispts)<best_mispts:
            best_mispts = len(mispts)
            best_w = w
            best_it = it
            print("Number of misclassified point is: {}".format(best_mispts))
            print("best_w is: {}".format(best_w))
            print("best_it is: {}".format(best_it))

        if x is None:
            print("Data was linearly seperable")
            break

        # Update weights
        w += s*x

    self.w = w
    self.best_w = best_w
    self.best_it = best_it
    print("The best w is {}".format(best_w))
    print("Iteration {} yields the best_w with {} misclassified points".format(best_it,
        ↪ best_mispts))
```

```
        self.plot(mispts=mispts, vec=w)
        plt.title('N = %s, Iteration %s, misclassified points %s\n' % (str(N),str(it), str(len(mispts))))
        plt.savefig('SC_LR%s_it%s' % (str(N),str(it)), dpi=200, bbox_inches='tight')

        return it

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

def main():
    it = np.zeros(1)
    for x in range(0, 1):
        p = Perceptron(2000)
        it[x] = p.pla(save=False)
        print(it)

main()
```

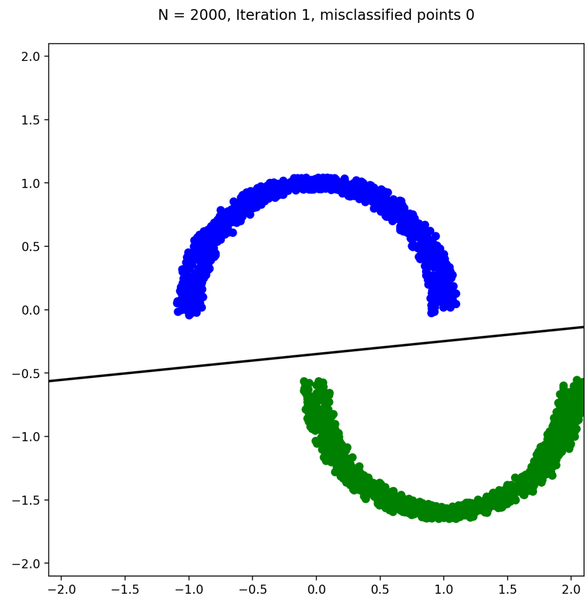
The output of Linear regression weights-

```
[[ 0.99564616  0.01985616]
 [ 0.97966131  0.02677898]
 [ 0.95330793  0.04733137]
 ...,
 [ 1.93806761 -0.6306617 ]
 [ 2.02310356 -0.59713891]
 [ 1.95835764 -0.64482745]]
[-1 -1 -1 ...,  1  1  1]
Number of misclassified point is: 0
best_w is: [-0.32167259  0.09341143 -0.91804317]
best_it is: 1
Data was linearly seperable
The best w is [-0.32167259  0.09341143 -0.91804317]
Iteration 1 yields the best_w with 0 misclassified points
[ 1.]
```

The output here shows the vectors - X and Y With the linear regression weigths, the data is linearly seperated in 1 iteration with the weights being,

$$w = [-0.32167259, 0.09341143, -0.91804317]$$

The plot looks like,



The plot here shows data points linearly seperated with a line.
While in part a, with PLA weight initiated at 0, PLA calculates the weights and converges to solution in 8 iterations while it learns to seperate the semi-circles linearly.
With linear regression wegith, since the weight is calculated analytically, and separation is 5, it makes Linear regression easy to calculate a weight that separates semi-circles and chooses the the first line that separates data.

5 3.2

Running the semi-circle with sep varying =

$$[0.2, 0.4, ...5]$$

The code is as follows.

```
#pocket perceptron with semi-circles dataset

import time
import numpy as np
import random
import os, subprocess
import matplotlib.pyplot as plt
from makeSemiCircles import make_semi_circles

class Perceptron:

    def __init__(self, N,j):
        # Random linearly separated data
        # # # # # # # # # random.seed(0)
        #xA,yA,xB,yB = [random.uniform(-1, 1) for i in range(4)]
        #x1A,x2A,x3A,...,x1B,x2B, = [random(1,11).uniform(-1, 1) for i in range(4)]
        #self.V = np.array([xB*yA-xA*yB, yB-yA, xA-xB])
        self.V = (np.random.rand(3)*2)-1
        # self.V = np.array([0.25, 0.5, 0.75])
        self.X= self.generate_points(N,j)




    def generate_points(self, N,j):
        X = []
        x, s = make_semi_circles(n_samples=N, thk=5, rad=10, sep=j, plot=False)
        x = np.insert(x,0,1,axis=1)# added bias of 1
        X = [[x[i], s[i]] for i in range(len(x))]
        print(j)
        return X
        #print(X)
# [
# ([b, x1, x2], s),
# ([b, x1.2, x2.2], s2),
# ]


    def plot(self, mispts=None, vec=None, save=False):
        fig = plt.figure(figsize=(8,8))
        plt.xlim(-2.1,2.1)
        plt.ylim(-2.1,2.1)
        #V = self.V
        #a, b = -V[1]/V[2], -V[0]/V[2]
        l = np.linspace(-3.1,3.1)
        #plt.plot(l, a*l+b, 'k-')
        cols = {1: 'g', -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], 'rx')
        if vec.any() != None:
            aa, bb = -vec[1]/vec[2], -vec[0]/vec[2] #idk what aa and bb are
            plt.plot(l, aa*l+bb, 'k-', 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
        myErr = 0
        for x,s in pts:
            myErr += abs(s - np.sign(vec.T.dot(x)))
            if np.sign(vec.T.dot(x)) != s:
                n_mispts += 1
        error = n_mispts / float(M)
        # print(error)
        # print(myErr)
        return error

    def choose_miscl_point(self, mispts):
        # Choose a random point among the misclassified
        if not mispts:
            return None,None
        return mispts[random.randrange(0,len(mispts))]


    def miscl_points_calc(self, vec):
        pts = self.X
        mispts = []
        for x,s in pts:
            if np.sign(vec.T.dot(x)) != s:
                mispts.append((x, s))

        return mispts

    def pla(self, save=False):
        # Initialize the weigths to zeros
        w = np.zeros(3)
        #w = self._W
        best_w = None
        best_mispts = None
        best_it = 0
        X, N = self.X, len(self.X)
        it = 0
        # Iterate until all points are correctly classified
        for i in range(50):
            it += 1
            mispts = self.miscl_points_calc(vec=w)

            # Pick random misclassified point
            x, s = self.choose_miscl_point(mispts=mispts)

            #if i % 5 == 0 and save:
```

```
            if save:
                self.plot(mispts=mispts, vec=w)
                plt.title('N = %s, Iteration %s, mispts %s\n' % (str(N),str(it), str(len(mispts))
                    ↪ ))
                plt.savefig('p_N%s_it%s' % (str(N),str(it)), dpi=200, bbox_inches='tight')

            #pocket parts
            if best_mispts is None or len(mispts)<best_mispts:
                best_mispts = len(mispts)
                best_w = w
                best_it = it
                print("Number of misclassified data point is: {}".format(best_mispts))
                print("best_w is: {}".format(best_w))
                print("best_it is: {}".format(best_it))

            if x is None:
                print("Data was linearly seperable")
                break

            # Update weights
            w += s*x

        self.w = w
        self.best_w = best_w
        self.best_it = best_it
        print("The best w is {}".format(best_w))
        print("Iteration {} yields the best_w with {} misclassified points".format(best_it,
            ↪ best_mispts))

        self.plot(mispts=mispts, vec=w)
        plt.title('N = %s, Iteration %s, misclassified points %s\n' % (str(N),str(it), str(len(
            ↪ mispts))))
        plt.savefig('SC_PLA%s_it%s' % (str(N),str(it)), dpi=200, bbox_inches='tight')

        return it

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

def main():
    it = np.zeros(1)
    for x in range(0, 1):
        sepa = np.array([0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8,
            ↪ 3.0, 3.2, 3.4, 3.6, 3.8, 4.0, 4.2, 4.4, 4.6, 4.8, 5.0])
        for j in sepa:
            p = Perceptron(2000,j)
            it[x] = p.pla(save=False)
        print(it)

main()
```

The code varies the sep variable from 0.2 till 5 with 0.2 step and we analyze the result. The table below shows seperation between semi-circles(lower the number, less the distance between semi circles).

sep	iteration number
0.2	49
0.4	44
0.6	34
0.8	38
1	38
1.2	28
1.4	24
1.6	18
1.8	16
2	12
2.2	16
2.4	14
2.6	10
2.8	12
3	12
3.2	8
3.4	8
3.6	8
3.8	14
4	8
4.2	8
4.4	4
4.6	8
4.8	8
5	6

The chart shows the trend of number of iterations against sepration values. As the sep value increases, it takes less iterations for PLA to converge and linearly seperate data.