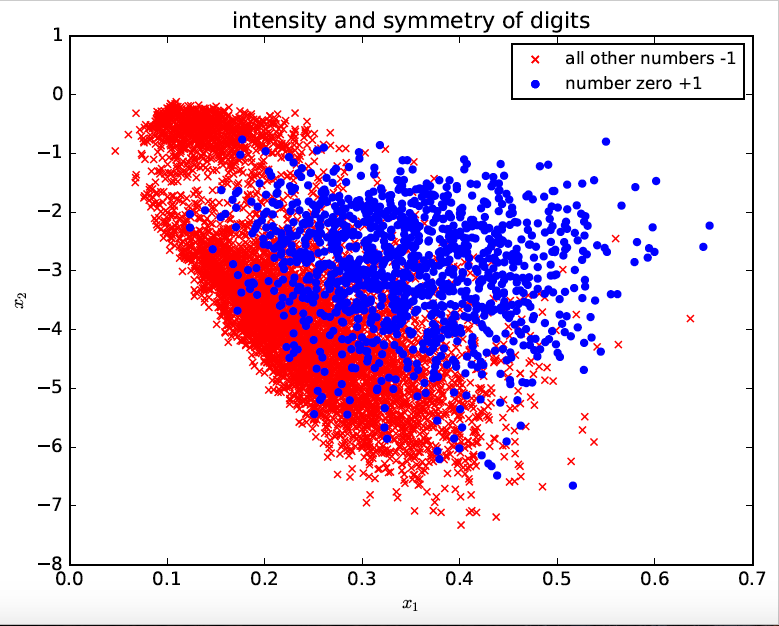
ARTIFICIAL INTELLIGENCE

MIDTERM

**Problem 1:**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from numpy import genfromtxt  dataset = genfromtxt('features.csv', delimiter = ' ')  y = dataset[:, 0]  X = dataset[:, 1:]  y[y<>0] = -1  y[y==0] = +1  #plots data  c0 = plt.scatter(X[y==-1,0], X[y==-1,1], s=20, color='r', marker='x')  c1 = plt.scatter(X[y==1,0], X[y==1,1], s=20, color='b', marker='o')  #displays legend  plt.legend((c0, c1), ('All Other Numbers -1', 'Number Zero +1'), loc='upper right', scatterpoints=1, fontsize=11)  #displays axis legends and title  plt.xlabel(r'$x\_1$')  plt.ylabel(r'$x\_2$')  plt.title(r'Intensity and Symmetry of Digits')  #saves the figure into a .pdf file (desired!)  plt.savefig('midterm.plot.pdf', bbox\_inches='tight')  plt.show()  print X |

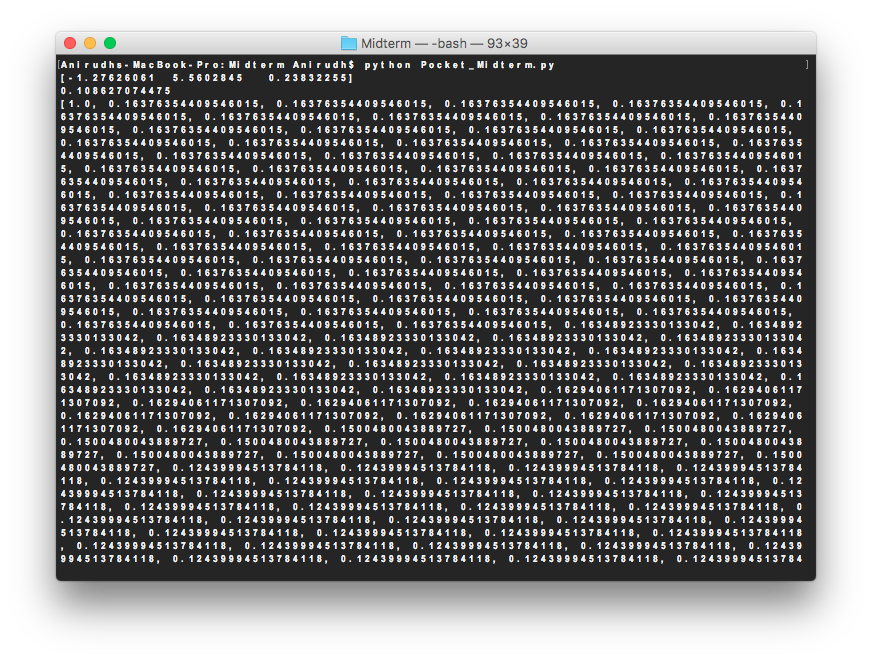
Output:

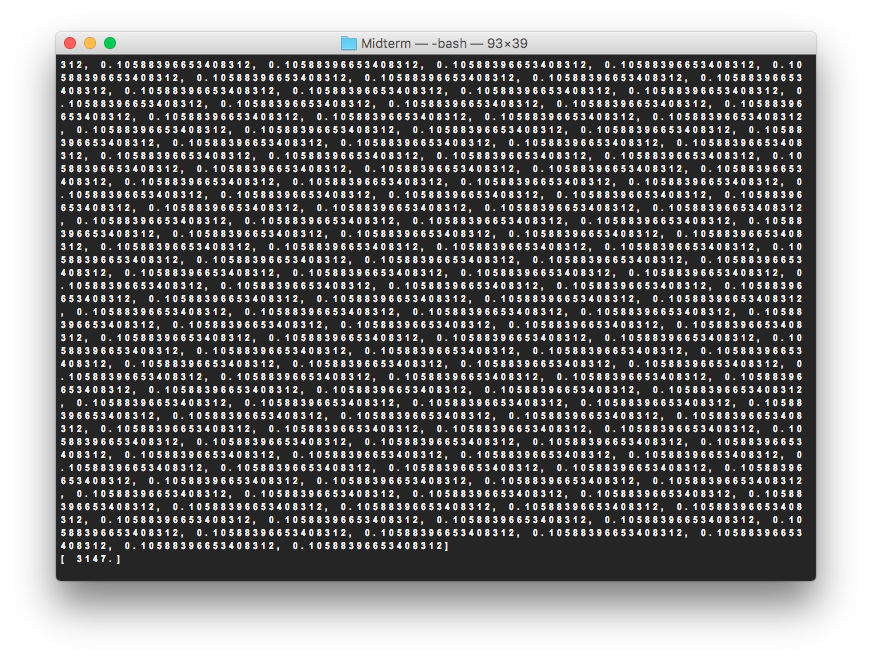


1. The Pocket Algorithm, starting from w = 0

|  |
| --- |
| import numpy as np  import random  import os, subprocess  import matplotlib.pyplot as plt  import copy  from numpy import genfromtxt  class pocket:  def \_\_init\_\_(self, N):  # Random linearly separated data  self.X = self.generate\_points(N)    def generate\_points(self, N):  dataset=genfromtxt('features.csv',delimiter=' ')  y=dataset[0:N,0]  X=dataset[0:N, 1:]  y[y<>0]=-1 #rest of numbers are negative class  y[y==0]=+1 #number zero is the positive class  bX = []  for k in range(0,N) :  bX.append((np.concatenate(([1], X[k,:])), y[k]))  # this will calculate linear regression at this point  X = np.concatenate((np.ones((N,1)), X),axis=1); # adds the 1 constant  self.linRegW = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y) # lin reg  print self.linRegW  return bX    def plot(self, mispts=None, vec=None, save=False):  fig = plt.figure(figsize=(5,5))  plt.xlim(-1.5,2.5)  plt.ylim(-2.0,1.5)  l = np.linspace(-1.5,2.5)  V = self.linRegW  a, b = -V[1]/V[2], -V[0]/V[2]  plt.plot(l, a\*l+b, 'k-')  V = self.bestW # for Pocket  a, b = -V[1]/V[2], -V[0]/V[2]  plt.plot(l, a\*l+b, 'r-')  cols = {1: 'r', -1: 'b'}  for x,s in self.X:  plt.plot(x[1], x[2], cols[s]+'.')  if mispts:  for x,s in mispts:  plt.plot(x[1], x[2], cols[s]+'x')  if vec.size:  aa, bb = -vec[1]/vec[2], -vec[0]/vec[2]  plt.plot(l, aa\*l+bb, 'g-', lw=2)  if save:  it 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 - int(np.sign(vec.T.dot(x))))  if int(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, 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)    #w = np.array([3.56224041, -14.46122032, -1.18142326])  self.bestW = copy.deepcopy(w); # for Pocket  self.plaError = []  self.pocketError = [] # for Pocket  X, N = self.X, len(self.X)  it = 0  lastchange=0  # Iterate until all points are correctly classified  self.plaError.append(self.classification\_error(w))  self.pocketError.append(self.plaError[it]) # for Pocket  while self.plaError[it] != 0 and it - lastchange<1000:  += 1  # Pick random misclassified point  x, s = self.choose\_miscl\_point(w)  # Update weights  w += s\*x  self.plaError.append(self.classification\_error(w))  if (self.pocketError[it-1] > self.plaError[it]): # for Pocket  self.pocketError.append(self.plaError[it])  self.bestW = copy.deepcopy(w);  lastchange=it  else:  self.pocketError.append(self.pocketError[it-1])  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')  plt.close()  #print it  self.w = w  print self.classification\_error(self.linRegW)  #print self.plaError  print self.pocketError  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 = pocket(7291)  it[x] = p.pla(save=False)  print it  main() |

Outputs:

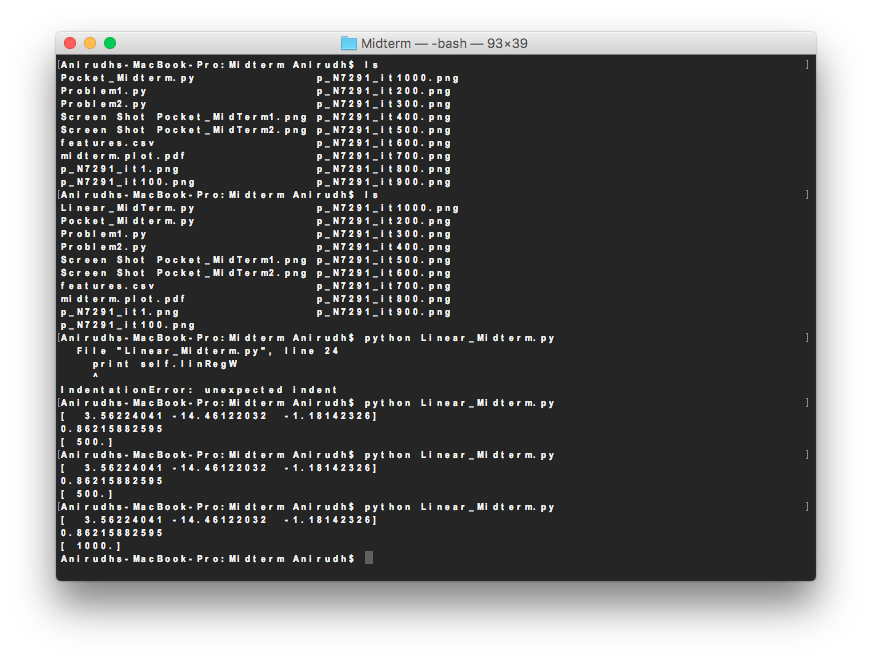




(b) Linear Regression (applied as a classification method)

|  |
| --- |
| import numpy as np  import random  import os, subprocess  import matplotlib.pyplot as plt  from numpy import genfromtxt  from sklearn.datasets.samples\_generator import make\_blobs    class Perceptron:  def \_\_init\_\_(self, N):  # Random linearly separated data  self.X = self.generate\_points(N)    def generate\_points(self, N):  dataset=genfromtxt('features.csv',delimiter=' ')  y=dataset[0:N,0]  X=dataset[0:N, 1:]  bX = []  for k in range(0,N):  bX.append((np.concatenate(([1], X[k,:])), y[k]))  # this will calculate linear regression at this point  X = np.concatenate((np.ones((N,1)), X),axis=1); # adds the 1 constant  self.linRegW = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y) # lin reg  print self.linRegW  return bX    def plot(self, mispts=None, vec=None, save=False):  fig = plt.figure(figsize=(5,5))  plt.xlim(-1.5,2.5)  plt.ylim(-2.0,1.5)  l = np.linspace(-1.5,2.5)  V = self.linRegW  a, b = -V[1]/V[2], -V[0]/V[2]  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]+'.')  if mispts:  for x,s in mispts:  plt.plot(x[1], x[2], cols[s]+'x')  if vec.size:  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  #myErr = 0  for x,s in pts:  #myErr += abs(s - int(np.sign(vec.T.dot(x))))  if int(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, 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 and it <1000:  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')  plt.close()  self.w = w  print self.classification\_error(self.linRegW)  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(7291)  it[x] = p.pla(save=False)  print it  main() |

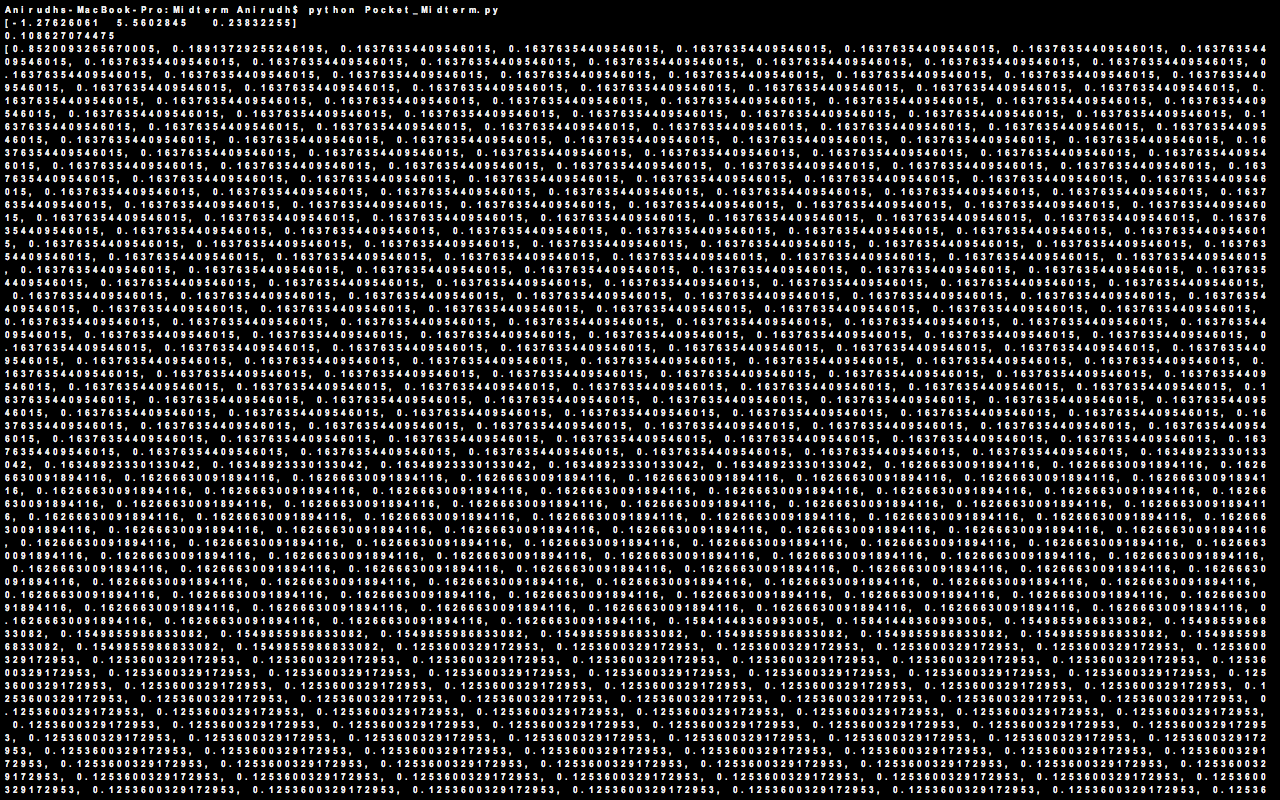
Output:

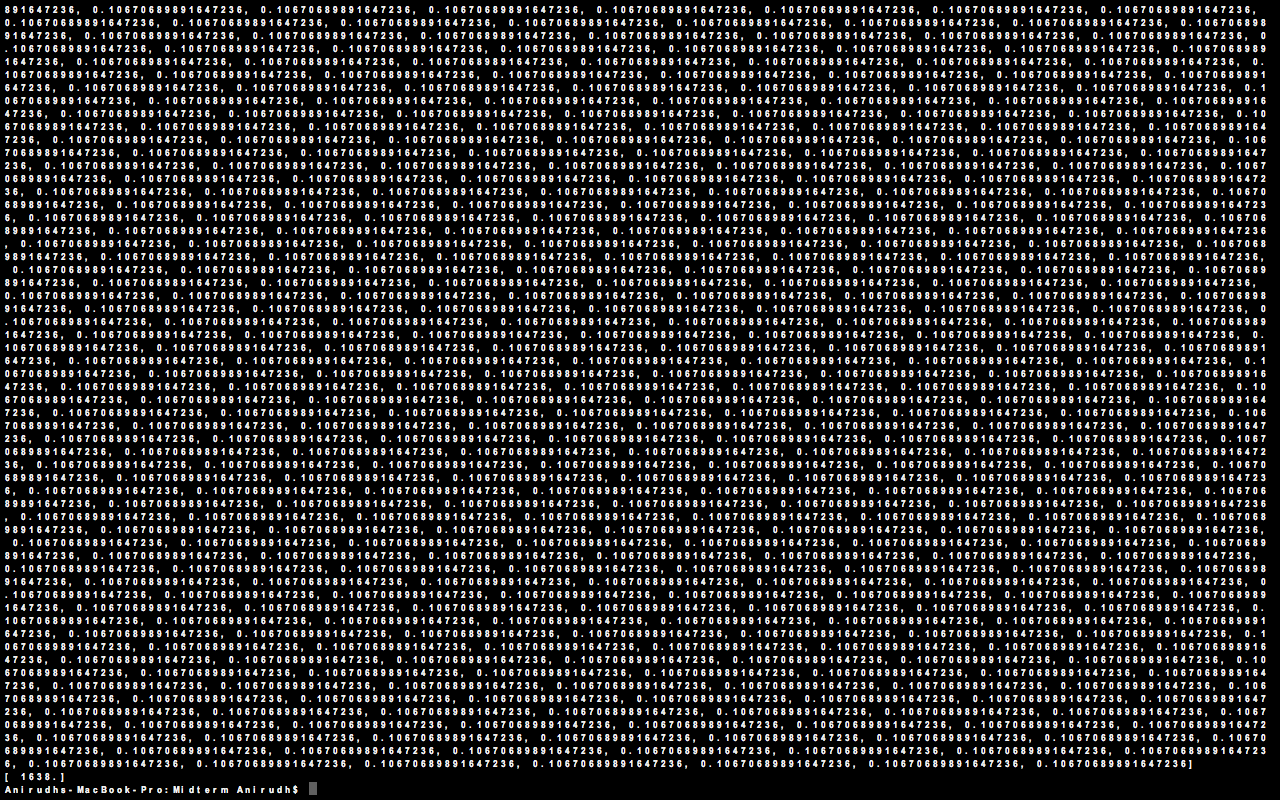


(c) Procket Algorithm, starting from the solution given by linear regression

|  |
| --- |
| import numpy as np  import random  import os, subprocess  import matplotlib.pyplot as plt  import copy  from numpy import genfromtxt  class pocket:  def \_\_init\_\_(self, N):  # Random linearly separated data  self.X = self.generate\_points(N)    def generate\_points(self, N):  dataset=genfromtxt('features.csv',delimiter=' ')  y=dataset[0:N,0]  X=dataset[0:N, 1:]  y[y<>0]=-1 #rest of numbers are negative class  y[y==0]=+1 #number zero is the positive class  bX = []  for k in range(0,N) :  bX.append((np.concatenate(([1], X[k,:])), y[k]))  # this will calculate linear regression at this point  X = np.concatenate((np.ones((N,1)), X),axis=1); # adds the 1 constant  self.linRegW = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y) # lin reg  print self.linRegW  return bX    def plot(self, mispts=None, vec=None, save=False):  fig = plt.figure(figsize=(5,5))  plt.xlim(-1.5,2.5)  plt.ylim(-2.0,1.5)  l = np.linspace(-1.5,2.5)  V = self.linRegW  a, b = -V[1]/V[2], -V[0]/V[2]  plt.plot(l, a\*l+b, 'k-')  V = self.bestW # for Pocket  a, b = -V[1]/V[2], -V[0]/V[2]  plt.plot(l, a\*l+b, 'r-')  cols = {1: 'r', -1: 'b'}  for x,s in self.X:  plt.plot(x[1], x[2], cols[s]+'.')  if mispts:  for x,s in mispts:  plt.plot(x[1], x[2], cols[s]+'x')  if vec.size:  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  #myErr = 0  for x,s in pts:  #myErr += abs(s - int(np.sign(vec.T.dot(x))))  if int(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, 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)    #w = np.array([3.56224041, -14.46122032, -1.18142326])  self.bestW = copy.deepcopy(w); # for Pocket  self.plaError = []  self.pocketError = [] # for Pocket  X, N = self.X, len(self.X)  it = 0  lastchange=0  # Iterate until all points are correctly classified  self.plaError.append(self.classification\_error(w))  self.pocketError.append(self.plaError[it]) # for Pocket  while self.plaError[it] != 0 and it - lastchange<1000:  it += 1  # Pick random misclassified point  x, s = self.choose\_miscl\_point(w)  # Update weights  w += s\*x  self.plaError.append(self.classification\_error(w))  if (self.pocketError[it-1] > self.plaError[it]): # for Pocket  self.pocketError.append(self.plaError[it])  self.bestW = copy.deepcopy(w);  lastchange=it  else:  self.pocketError.append(self.pocketError[it-1])  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')  plt.close()  #print it  self.w = w  print self.classification\_error(self.linRegW)  #print self.plaError  print self.pocketError  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 = pocket(7291)  it[x] = p.pla(save=False)  print it  main() |

Outputs:





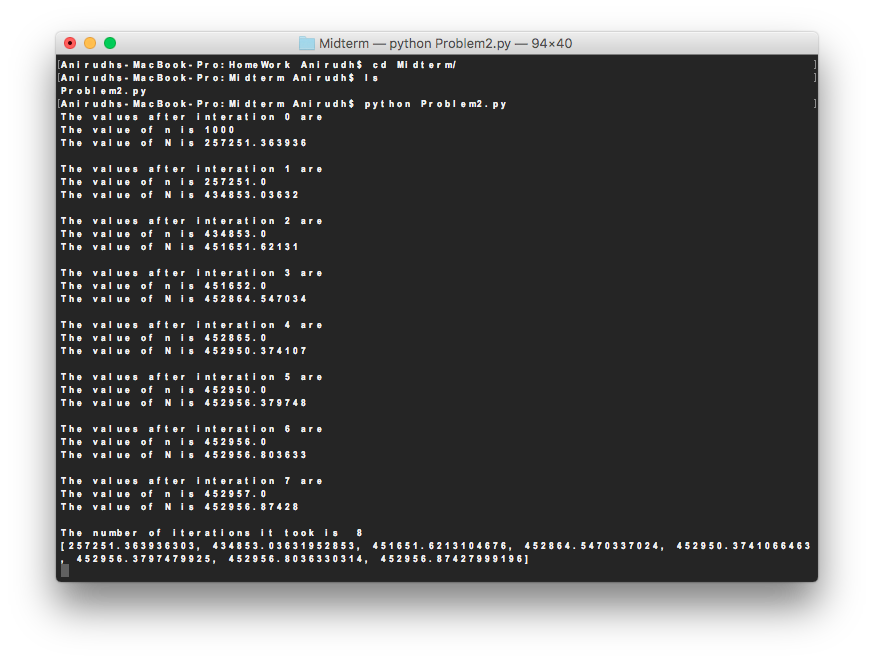
**Problem 2:**

Python problem to solve Problem 2.12

**Code**

|  |
| --- |
| import matplotlib.pyplot as plt  import math  Dvc = 10  d = 0.05  e = 0.05  n = 1000 #Initial sample  iterations = 0 #To count the number of iterations 'N' takes to converge  A = [] #Array to store values of 'N'  for i in range (1,10):  N = (8/(math.pow(e,2))) \* math.log(4\*(math.pow(2\*n,Dvc)+1)/d)  A.append(N)  print "The values after interation", iterations, "are"  print "The value of n is", n  print "The value of N is", N  print  iterations = iterations + 1 #Increment the number of iterations  if round(N) == n: #Compare the value of 'N' to the previous value  print "The number of iterations it took is ", iterations  break  else:  n = round(N)  continue  print A #Prints all values of 'N'  plt.plot(A) #Plots the graph of all values of 'N'  plt.title('Graph of N') #Sets the title  plt.ylabel('N')  plt.xlabel('Iterations')  plt.show(A) |

**Terminal**



**Graph**

