HomeWork 03

Deep Dand

May 15, 2018

Abstract

Pocket algorithm, Linear Regression(applied as classification method) on Blob dataset

1 Problem

1.1 a

This is the output with PLA

$$w = 0$$

Code Begins

```
#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
from sklearn.datasets.samples_generator import make_blobs
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 = []

    ctrs = 3*np.random.normal(0,1,(2,2))
    x, s = make_blobs(n_samples=100, centers=ctrs, n_features
        =2, cluster_std=1.0, shuffle=False, random_state=0)
    #change targets that are 0 to -1
    s[s==0] = -1
    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


def plot(self, mispts=None, vec=None, save=False):
    fig = plt.figure(figsize=(8,8))
    #plt.xlim(-5.1,1.1)
    #plt.ylim(-1.1,3.1)
    #V = self.V
    #a, b = -V[1]/V[2], -V[0]/V[2]
    l = np.linspace(-5.1,5.1)
    #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('Blob_PLA_N%s' % (str(len(self.X))), dpi
            =200, bbox_inches='tight')

#is this actually used?
def classification_error(self, vec, pts=None):
```

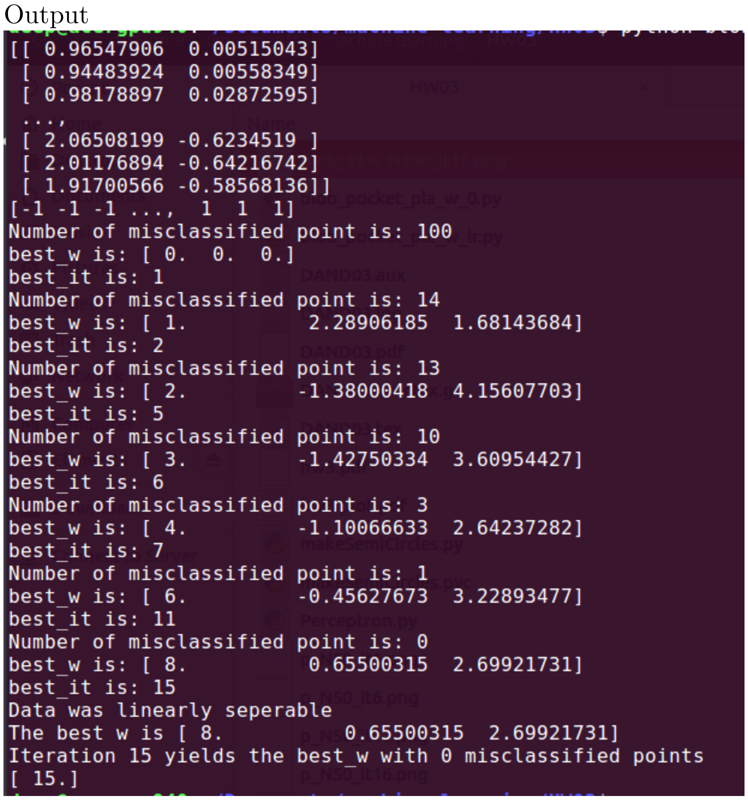
```
        # Error defined as fraction of misclassified points
        if not pts:
            pts = self.X
        M = len(pts)
        n_mispts = 0
        myErr = 0
        for x,s in pts:
            myErr += abs(s - np.sign(vec.T.dot(x)))
            if np.sign(vec.T.dot(x)) != s:
                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 = [0.12435462, 0.23836865, 0.07567514]
        #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(300):
            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('Blob_PLA_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_vec = 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=best_mispts_vec, vec=best_w)
        plt.title('N = %s, Iteration %s, misclassified points %s\n
            ↪ ' % (str(N),str(best_it), str(best_mispts)))
        plt.savefig('Blob_PLA_N%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()
```

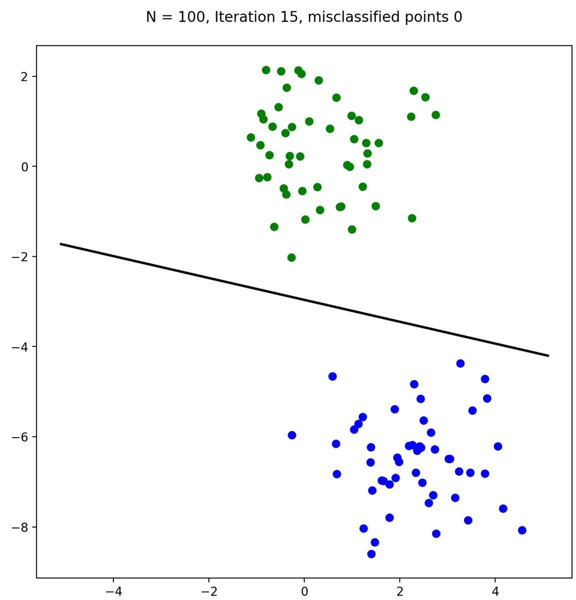
Code ends

Output



```
[[ 0.96547906  0.00515043]
 [ 0.94483924  0.00558349]
 [ 0.98178897  0.02872595]
 ...,
 [ 2.06508199 -0.6234519 ]
 [ 2.01176894 -0.64216742]
 [ 1.91700566 -0.58568136]]
[-1 -1 -1 ...,  1  1  1]
Number of misclassified point is: 100
best_w is: [ 0.  0.  0.]
best_it is: 1
Number of misclassified point is: 14
best_w is: [ 1.          2.28906185  1.68143684]
best_it is: 2
Number of misclassified point is: 13
best_w is: [ 2.         -1.38000418  4.15607703]
best_it is: 5
Number of misclassified point is: 10
best_w is: [ 3.         -1.42750334  3.60954427]
best_it is: 6
Number of misclassified point is: 3
best_w is: [ 4.         -1.10066633  2.64237282]
best_it is: 7
Number of misclassified point is: 1
best_w is: [ 6.         -0.45627673  3.22893477]
best_it is: 11
Number of misclassified point is: 0
best_w is: [ 8.          0.65500315  2.69921731]
best_it is: 15
Data was linearly seperable
The best w is [ 8.          0.65500315  2.69921731]
Iteration 15 yields the best_w with 0 misclassified points
[ 15.]
```

The plot looks like,



The pocket algorithm takes 15 iterations for the execution time and finds the best classification. The best weight is

$$w = [8, 0.655, 2.6699]$$

and the best result came at iteration 15.

1.2 b

The output of Linear regression weights- Code Begins

```
#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
from sklearn.datasets.samples_generator import make_blobs
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 = []

        ctrs = 3*np.random.normal(0,1,(2,2))
        x, s = make_blobs(n_samples=100, centers=ctrs, n_features
            ↪ =2, cluster_std=1.0, shuffle=False, random_state=0)
        #change targets that are 0 to -1
        s[s==0] = -1
        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


    def plot(self, mispts=None, vec=None, save=False):
        fig = plt.figure(figsize=(8,8))
        #plt.xlim(-5.1,1.1)
        #plt.ylim(-1.1,3.1)
        #V = self.V
        #a, b = -V[1]/V[2], -V[0]/V[2]
        l = np.linspace(-5.1,5.1)
        #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('Blob_PLA_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 = [0.12435462, 0.23836865, 0.07567514]
    #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
```

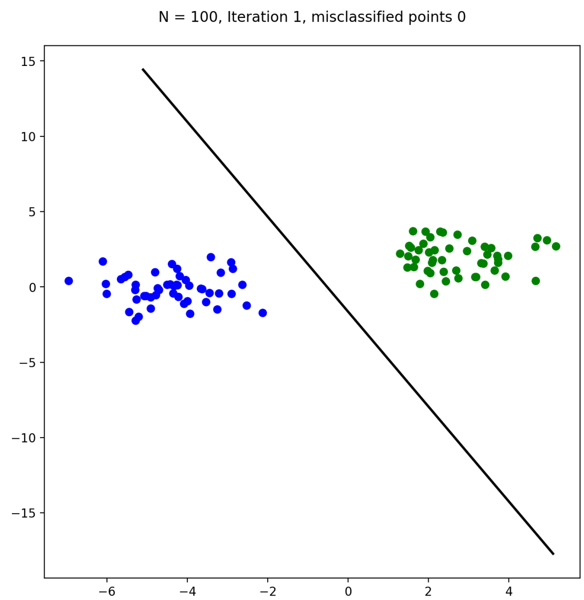
```
for i in range(300):
    it += 1
    mispts = self.miscl_points_calc(vec=w)

    # 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('Blob_PLA_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_vec = 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=best_mispts_vec, vec=best_w)
plt.title('N = %s, Iteration %s, misclassified points %s\n
    ↪ ' % (str(N),str(best_it), str(best_mispts)))
plt.savefig('Blob_PLA_N%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.
Output

```
[[ 0.93438823 -0.04720666]
 [ 1.07841872  0.0051774 ]
 [ 0.95321342  0.05320635]
 ...,
 [ 1.95874136 -0.60179137]
 [ 1.91052338 -0.6067994 ]
 [ 1.96399579 -0.61167434]]
[-1 -1 -1 ...,  1  1  1]
Number of misclassified point is: 0
best_w is: [ 0.12435462  0.23836865  0.07567514]
best_it is: 1
Data was linearly seperable
The best w is [ 0.12435462  0.23836865  0.07567514]
Iteration 1 yields the best_w with 0 misclassified points
[ 1.]
deep@acergpu940:~/Documents/machine_learning/HW03$
```

The plot looks like,



The linear regression takes only 1 iterations for the execution time and finds the best classification. The best weight is

$$w = [0.1243, 0.2383, 0.0757]$$

and the best result came at iteration 1.

2 summary of experiments

From the experiments that we have performed in this document, linear regression weights are a good way to initialize the weights for pocket algorithm/PLA. The Linear regressions itself aren't the best at giving goof accuracy but if the weights from LR is given as initial weight to Pocket Algorithm, the results can be much faster to converge since initializing the results to some potential value helps Pocket to identify the closest possible results in less number of iterations.