Perceptron Learning Algorithm

**一、概论**

对于给定的n维数据，找出一个n-1维的超平面面，能够“尽可能”地按照数据类型分开。

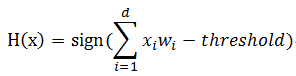
例如对于二维数据，要找一条直线，把这些数据按照不同类型分开。我们要通过PLA**算法**，找到这条直线，然后通过判断预测数据与这条直线的位置关系，划分测试数据类型。

**二、PLA的原理**

对于线性可分的数据，先初始化一条直线，然后通过多次迭代，修改这条直线，通过多次迭代，这条直线会收敛于接近最佳分类直线。

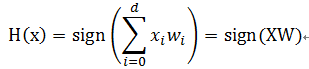
修改直线的标准是，任意找出一个点，判断这个点按照这条直线的划分类型是否跟该点实际类型是否相同。如果相同则开始下次迭代；如果判断错误，则更新直线的参数。

PLA算法即用来求向量W，使得在已知的数据中机器做出的判断与现实完全相同。内积可以表示成：



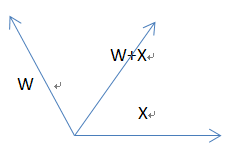


进一步可化简成：



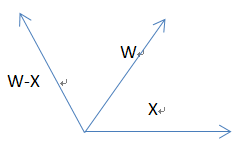


PLA先假定W为0向量，然后找到一个不满足条件的点，调整W的值，依次进行迭代使得最终可以将两部分完全分开。W的调整方案如下：第一种，在给定的已知数据中向该用户发放了数据，但算法给出的结果是不发放，说明两个向量的内积为负，需要调整向量使得二者的值为正，此时y=1。示意图为



则调整后的W’=W+X=W+yX。

第二种情况是原本没有发放但算法显示应该发放，此时y=-1。示意图为：



则调整后的W’=W-X=W+Xy。

对于线性可分的数据集，PLA算法是可收敛的。证明如下：存在完美的Wf使得 ：



所以：



t表示经过第t次调整。



两个向量的内积增大说明两个向量越来越相似或者向量的长度增大。向量W(t+1)的长度可以表示为：



因为第t次发现不合格才会调整，所以得到：

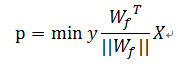


可以得到如下公式：



这说明每次调整后，向量的长度增加有限。不妨设：



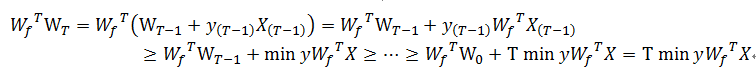


带入上一公式得到：



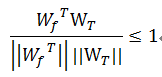
因此，W(t)最终是收敛的。到此已经证明了PLA算法最终可以停止。下面求该算法需要调整多少步才能停止。

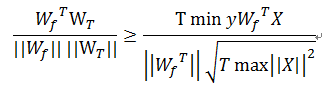
由上述过程可以得到以下两个不等式：

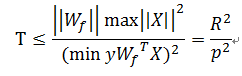




  根据余弦值最大为1，可以得到:

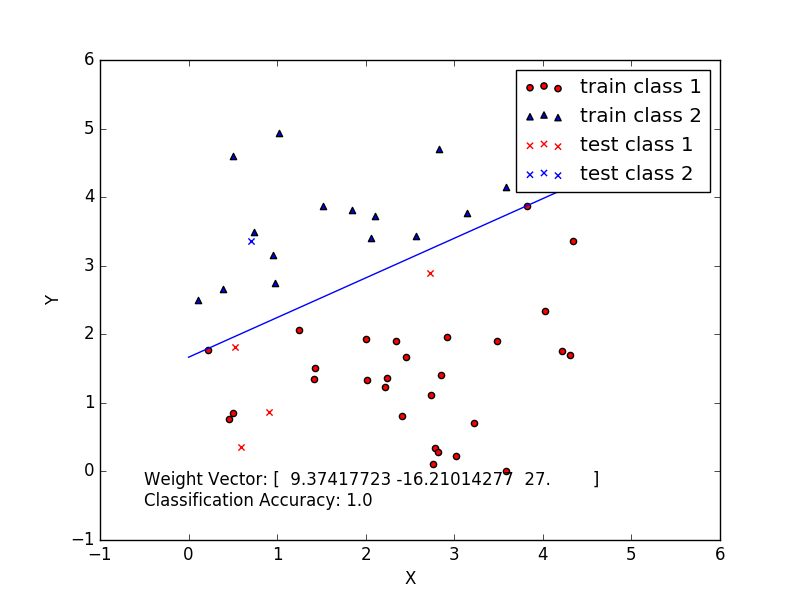


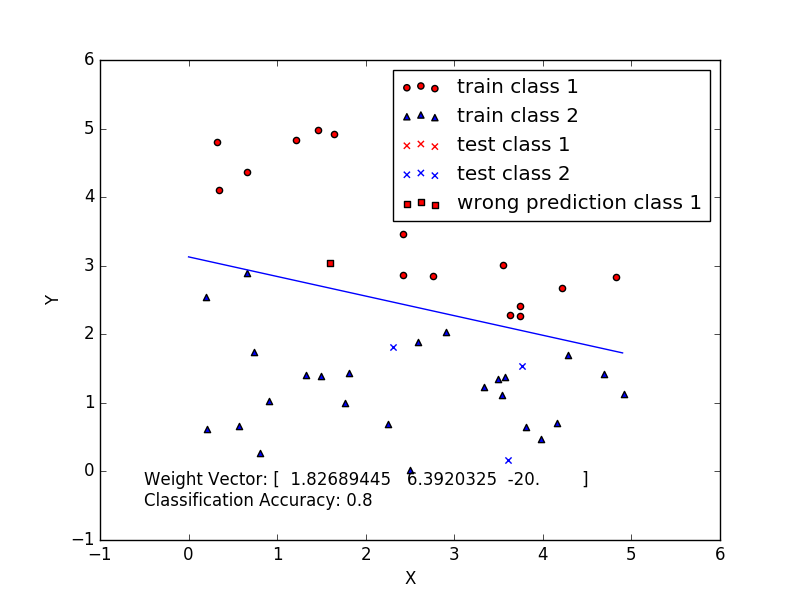
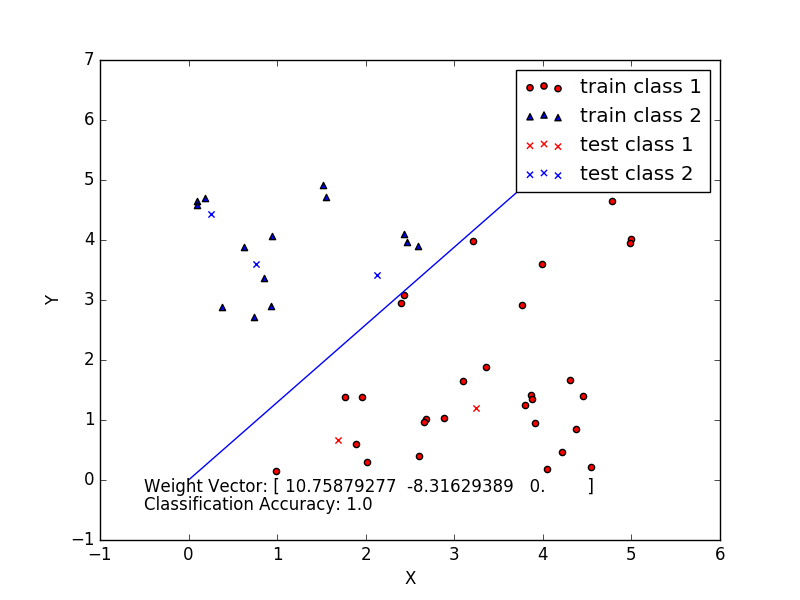




**三、PLA的实现**

使用python实现的PLA，采用随机函数生成数据集，随机将数据集分成训练集和测试集，将训练集数据用于算法的训练，最后收敛得到CLF直线，将其用于测试集的分类。计算得到分类精度。三次实验截图如下所示：





**四、代码**

**以下代码可以在github（**[**osgee**](https://github.com/osgee/Machine_Learning)**）上找到。**

**import** random  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
data\_file = **'Dataset\_PLA.csv'**Max\_Iteration = 1000  
  
  
**def** generate\_data(w, border, size):  
 **with** open(data\_file, **'w+'**) **as** data\_set:  
 **for** i **in** range(size):  
 x = random.random() \* border  
 y = random.random() \* border  
 z = w[0] \* x + w[1] \* y + w[2] \* 1  
 **if** z > 0:  
 s = 1  
 **else**:  
 s = -1  
 data\_set.write(str(x) + **','** + str(y) + **','** + **'1'** + **','** + str(s) + **'\n'**)  
  
  
**def** load\_data(test\_ratio):  
 **with** open(data\_file, **'r'**) **as** data\_set:  
 lines = data\_set.readlines()  
 data\_size = len(lines)  
 test\_size = int(data\_size \* test\_ratio)  
 test\_index = random.sample(range(data\_size), test\_size)  
 train\_array = [[float(c) **for** c **in** lines[i].strip().split(**','**)] **for** i **in** range(data\_size) **if** i **not in** test\_index]  
 test\_array = [[float(c) **for** c **in** lines[i].strip().split(**','**)] **for** i **in** range(data\_size) **if** i **in** test\_index]  
 train\_mat = np.array(train\_array)  
 test\_mat = np.array(test\_array)  
 **return** train\_mat, test\_mat  
  
  
**def** update(w, train\_vector):  
 **if** np.dot(train\_vector[:-1], w) \* train\_vector[-1] > 0:  
 **return** w, **False  
 else**:  
 **return** w + train\_vector[-1] \* np.transpose(train\_vector[:-1]), **True  
  
  
def** train(w, train\_data):  
 iteration = Max\_Iteration  
 **for** i **in** range(iteration):  
 updated = **False  
 for** t **in** range(train\_data.shape[0]):  
 w, up\_out = update(w, train\_data[t])  
 updated = updated **or** up\_out  
 **if not** updated:  
 **break  
 return** w  
  
  
**def** predict(w, train\_data, test\_data):  
 w = train(w, train\_data)  
 print(w)  
 fig = plt.figure()  
 ax = fig.add\_subplot(111)  
 n = train\_data.shape[0]  
 train\_scatter1 = **None** train\_scatter2 = **None  
 for** xs, ys, zs, ts **in** train\_data:  
 **if** ts == 1:  
 c = **'r'** m = **'o'** train\_scatter1 = ax.scatter(xs, ys, c=c, marker=m)  
 **else**:  
 c = **'b'** m = **'^'** train\_scatter2 = ax.scatter(xs, ys, c=c, marker=m)  
 test\_scatter1 = **None** test\_scatter2 = **None  
 for** xs, ys, zs, ts **in** test\_data:  
 **if** ts == 1:  
 c = **'r'** m = **'x'** test\_scatter1 = ax.scatter(xs, ys, c=c, marker=m)  
 **else**:  
 c = **'b'** m = **'x'** test\_scatter2 = ax.scatter(xs, ys, c=c, marker=m)  
  
 wrong\_data = []  
 **for** i **in** range(test\_data.shape[0]):  
 **if** np.dot(test\_data[i, :-1], w) > 0:  
 r = 1  
  
 **else**:  
 r = -1  
 **if** test\_data[i, -1] != r:  
 test\_data[i, -1] = r  
 wrong\_data.append(test\_data[i, :])  
  
 prediction\_acc = 1 - len(wrong\_data) / test\_data.shape[0]  
 plt.annotate(**'Classification Accuracy: '** + str(prediction\_acc), xy=(1, 1), xytext=(-0.5, -0.5))  
 plt.annotate(**'Weight Vector: '** + str(w), xy=(1, 1), xytext=(-0.5, -0.2))  
 wrong\_scatter1 = **None** wrong\_scatter2 = **None  
 for** xs, ys, zs, ts **in** wrong\_data:  
 **if** ts == 1:  
 c = **'r'** m = **'s'** wrong\_scatter1 = ax.scatter(xs, ys, c=c, marker=m)  
 **else**:  
 c = **'b'** m = **'s'** wrong\_scatter2 = ax.scatter(xs, ys, c=c, marker=m)  
  
 ax.set\_xlabel(**'X'**)  
 ax.set\_ylabel(**'Y'**)  
 x = np.arange(0, 5, 0.1)  
 y = (-w[2] - w[0] \* x) / w[1]  
 line\_clf = ax.plot(x, y, label=**'CLF'**)  
 handles, labels = ax.get\_legend\_handles\_labels()  
 **if** wrong\_scatter1 **is not None or** wrong\_scatter2 **is not None**:  
 ax.legend([train\_scatter1, train\_scatter2, test\_scatter1, test\_scatter2, wrong\_scatter1, wrong\_scatter2], \  
 [**'train class 1'**, **'train class 2'**, **'test class 1'**, **'test class 2'**, **'wrong prediction class 1'**,  
 **'wrong prediction class 2'**])  
 **elif** test\_scatter1 **is not None or** test\_scatter2 **is not None**:  
 ax.legend([train\_scatter1, train\_scatter2, test\_scatter1, test\_scatter2], \  
 [**'train class 1'**, **'train class 2'**, **'test class 1'**, **'test class 2'**])  
 **else**:  
 ax.legend([train\_scatter1, train\_scatter2], \  
 [**'train class 1'**, **'train class 2'**])  
 plt.show()  
  
  
generate\_data([1, -1, 1], 5, 50)  
*# generate\_data([0.5, 1, -4], 5, 50)  
# generate\_data([0.5, -1, 2], 5, 50)*train\_data, test\_data = load\_data(0.1)  
w = np.ones((train\_data.shape[1] - 1, 1))  
predict(w, train\_data, test\_data)