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：

$$\mathrm{H(x)} = \mathrm{sign}\left(\sum_{i=1}^{d} x_i w_i - threshold\right)$$

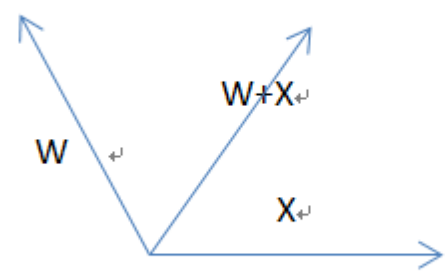
$$\mathrm{sign}(\mathrm{x}) = \begin{cases} 1 & x > 0 \\ -1 & x \le 0 \end{cases}$$

进一步可化简成：

$$H(x)=\mathrm{sign}\left(\sum_{i=0}^{d} x_i w_i\right)=\mathrm{sign}(XW)$$

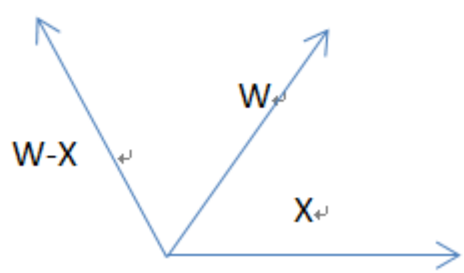
$$y=\begin{cases}1\\-1\end{cases}$$

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 ：

$$y = {W_f}^T X$$

所以：

$$y(t){W_f}^T X(t) \geq \min y\, {W_f}^T X > 0$$

t表示经过第t次调整。

$${W_f}^T W(t+1) = {W_f}^T (W(t) + X(t)y(t)) = {W_f}^T W(t) + y(t){W_f}^T X(t) > {W_f}^T W(t)$$

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：

$$||W(t+1)||^2 = ||(W(t) + X(t)y(t))||^2 = ||W(t)||^2 + ||X(t)y(t)||^2 + 2y(t)W(t)\,X(t)$$

因为第t次发现不合格才会调整，所以得到：

$$2y(t)W(t)X(t) < 0$$

可以得到如下公式：

$$||W(t+1)||^2 < ||W(t)||^2 + ||X(t)y(t)||^2 = ||W(t)||^2 + ||X(t)||^2 \leq ||W(t)||^2 + \max ||X||^2$$

这说明每次调整后，向量的长度增加有限。不妨设：

$$R^2 = \max ||X||^2$$

$$p = \min y \frac{{W_f}^T}{||W_f||} X$$

带入上一公式得到：

$$\frac{||W(t+1)||^2}{||W(t)||^2} \le 1 + \frac{R^2}{||W(t)||^2}$$

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：

$$\begin{aligned} W_f{}^T \mathrm{W}_T = W_f{}^T\left(\mathrm{W}_{T-1} + y_{(T-1)}X_{(T-1)}\right) &= W_f{}^T\mathrm{W}_{T-1} + y_{(T-1)}W_f{}^T X_{(T-1)} \\ &\ge W_f{}^T\mathrm{W}_{T-1} + \min yW_f{}^T X \ge \cdots \ge W_f{}^T\mathrm{W}_0 + \mathrm{T}\min yW_f{}^T X = \mathrm{T}\min yW_f{}^T X \end{aligned}$$

$$||\mathrm{W}_T||^2 \le ||\mathrm{W}_{T-1}||^2 + \max\left||X|\right|^2 \le \cdots \le ||\mathrm{W}_0||^2 + T\max\left||X|\right|^2 = T\max\left||X|\right|^2$$

根据余弦值最大为1，可以得到：

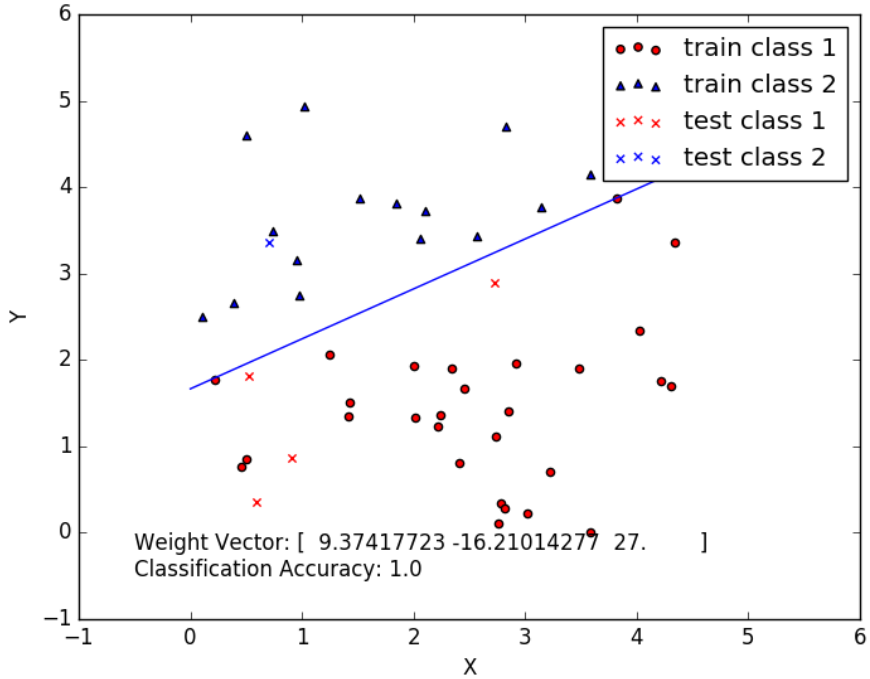
$$\frac{W_f{}^T\mathrm{W}_T}{\left||W_f{}^T|\right|\,||\mathrm{W}_T||} \le 1$$

$$\frac{W_f{}^T\mathrm{W}_T}{||W_f||\,||\mathrm{W}_T||} \ge \frac{\mathrm{T}\min yW_f{}^T X}{\left||W_f{}^T|\right|\sqrt{T\max\left||X|\right|^2}}$$

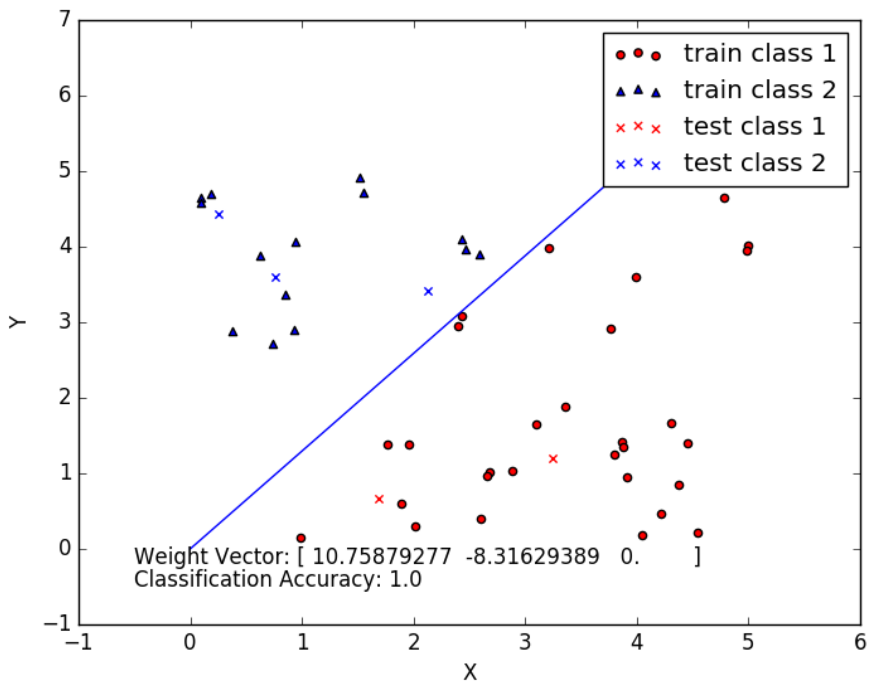
$$\mathrm{T} \le \frac{\left||W_f|\right|\max\left||X|\right|^2}{(\min yW_f{}^T X)^2} = \frac{R^2}{p^2}$$

三、PLA的实现

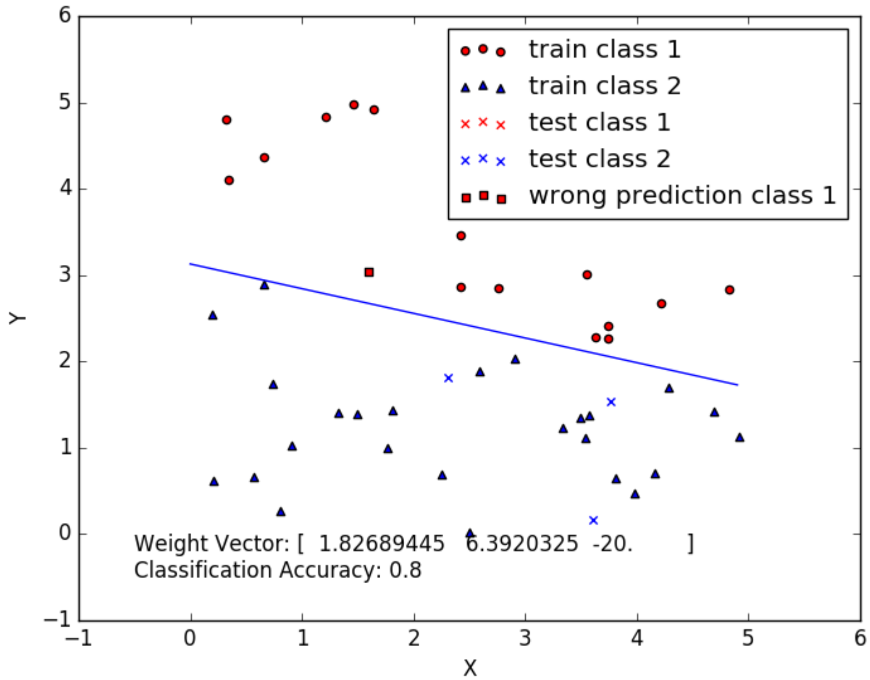
使用python实现的PLA，采用随机函数生成数据集，随机将数据集分成训练集和测试集，将训练集数据用于算法的训练，最后收敛得到CLF直线，将其用于测试集的分类。计算得到分类精度。三次实验截图如下所示：



train class 1
train class 2
test class 1
test class 2
Weight Vector: [9.37417723 -16.21014277 27.   ]
Classification Accuracy: 1.0
X
Y



train class 1
train class 2
test class 1
test class 2
Weight Vector: [10.75879277 -8.31629389   0.   ]
Classification Accuracy: 1.0
X
Y



四、代码

以下代码可以在github（[osgee]）上找到。

```python
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)
```