感知机

1 感知机

1.1 线性可分

定义：给定数据集

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$$

其中，$x_i \in R^n, y_i \in \{-1, 1\}, i = 1, 2, ..., N$，如果存在某个超平面 $S : \omega x + b = 0$，使得对 $\forall i = 1, 2, ..., N, (\omega x_i + b)y_i > 0$，则称数据集 T 为线性可分数据集。

感知机需要数据线性可分。

1.2 损失函数

感知机使用**误分类点**到超平面总距离作为学习的损失函数。记 M 是误分类点的集合，则感知机的损失函数为：

$$L(\omega, b) = -\frac{1}{||\omega||}\sum_{x_i \in M} y_i(\omega x_i + b)$$

不妨令 $||\omega|| = 1$(这是因为 $\omega x + b = 0$ 和 $\frac{\omega}{||\omega||}x + \frac{b}{||\omega||} = 0$ 是一个平面)，则感知机的损失函数为

$$L(\omega, b) = -\sum_{x_i \in M} y_i(\omega x_i + b)$$

1.3 优化算法

优化算法的目标是求得 $\omega^*, b^* = argmin_{\omega,b} L(\omega, b)$，使用随机梯度下降法进行优化。方法是：每次选出一个误分类点 (x_i, y_i)，对于该误分类点的损失函数 $L_i(\omega, b) = -y_i(\omega x_i + b)$，有：

$$\begin{cases} \frac{\partial L_i}{\partial \omega} = -y_i x_i \\ \frac{\partial L_i}{\partial b} = -y_i \end{cases}$$

因此，参数的更新过程为：

$$\begin{cases} \omega := \omega + \eta y_i x_i \\ b := b + \eta y_i \end{cases}$$

所以感知机算法如下：

输入：线性可分的训练数据集 $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$。其中，$x_i \in R^n, y \in \{-1, 1\}, i = 1, 2, ..., N$ 和学习率 $\eta \in (0, 1]$。

1. 随机选取初值 w_0, b_0
2. 在训练集中选取数据 (x_i, y_i)
3. 若 $y_i(\omega x_i + b) \leq 0,$
$$\omega := \omega + \eta y_i x_i$$
$$b := b + \eta y_i$$
4. 回到 2，直至训练集中没有误分类点

输出：参数 ω, b 和对应的感知机模型 $f(x) = sign(\omega x + b)$。

1.4 收敛性

引理：对于将数据集 T 完全分开的超平面 $\omega_{opt} x + b_{opt} = 0$，不妨令其满足条件 $||\omega_{opt}|| = 1$。则存在 $\gamma > 0$，对所有 $i = 1, 2, ..., N$ 有：

$$y_i(\hat{\omega}_{opt} \hat{x}_i) := y_i(\omega_{opt} x + b_{opt}) \geq \gamma$$

Proof：显然。

定理：感知机算法必收敛。且其收敛次数 $k \leq (\frac{R}{\gamma})^2$，其中 $R = \max_{1 \leq i \leq N} ||\hat{x}_i||$。

Proof:

设 $\hat{\omega}_k$ 是误分点 (x_i, y_i) 在 $\hat{\omega}_{k-1}$ 上更新而来的。因此有 $y_i \hat{\omega}_{k-1} \hat{x}_i < 0$。

$$\begin{aligned} \hat{\omega}_k \hat{\omega}_{opt} &= (\hat{\omega}_{k-1} + \eta y_i x_i) \hat{\omega}_{opt} \\ &= \hat{\omega}_{k-1} \hat{\omega}_{opt} + \eta y_i x_i \hat{\omega}_{opt} \\ &\geq \hat{\omega}_{k-1} \hat{\omega}_{opt} + \eta \gamma \end{aligned}$$

假设 $\hat{\omega}_0 = 0$, 将此式不断递推即有：

$$\hat{\omega}_k \hat{\omega}_{opt} \geq \hat{\omega}_{k-1} \hat{\omega}_{opt} + \eta\gamma \geq \hat{\omega}_{k-2} \hat{\omega}_{opt} + 2\eta\gamma \geq ... \geq k\eta\gamma$$

此外，

$$\begin{aligned} ||\hat{\omega}_k||^2 &= ||\hat{\omega}_{k-1} + \eta y_i x_i||^2 \\ &= ||\omega_{k-1}||^2 + 2\eta y_i \hat{\omega}_{k-1} \hat{x}_i + \eta^2 x_i^2 \\ &\leq ||\omega_{k-1}||^2 + 0 + \eta^2 R^2 \\ &= ||\omega_{k-1}||^2 + \eta^2 R^2 \\ &\leq ||\omega_{k-2}||^2 + 2\eta^2 R^2 \leq ... \\ &\leq k\eta^2 R^2 \end{aligned}$$

联立以上两式有：

$$k\eta\gamma \leq \hat{\omega}_k \hat{\omega}_{opt} \leq ||\hat{\omega}_k|| \cdot ||\hat{\omega}_{opt}|| = ||\hat{\omega}_k|| \leq \sqrt{k}\eta R$$

由 $k\eta\gamma \leq \sqrt{k}\eta R$ 得 $k \leq (\frac{R}{\gamma})^2$。

1.5 对偶形式

原问题的参数更新形式是：

$$\omega := \omega + \eta y_i x_i$$

$$b := b + \eta y_i$$

因此，参数 ω 和 b 可以表示为如下线性组合：

$$\omega = \sum_{i=1}^{N} \alpha_i y_i x_i$$

$$b = \sum_{i=1}^{N} \alpha_i y_i$$

对照原问题的优化步骤，对偶问题的优化步骤为：

输入：线性可分的训练数据集 $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$。其中，$x_i \in R^n, y \in \{-1, 1\}, i = 1, 2, ..., N$ **和**学习率 $\eta \in (0, 1]$。

1. 随机选取初值 $\alpha = (\alpha_1, \alpha_2, ..., \alpha_N)^T$ 和 b
2. 在训练集中选取数据 (x_i, y_i)
3. 若 $y_i[(\sum_{j=1}^{N} \alpha_j y_j x_j) x_i + b] \leq 0$,

$$\alpha_i := \alpha_i + \eta$$

$$b := b + \eta y_i$$

4. 回到 2，直至训练集中没有误分类点

输出：参数 α, b 和对应的模型 $f(x) = sign(\sum_{j=1}^{N} \alpha_j y_j x_j \cdot x + b)$。

在使用对偶问题求解时，由于不断使用 x_i 和 x_j 的积，所以常常使用 Gram 矩阵进行存储内积，即 $G = [x_i \cdot x_j]_{N \times N}$。

2 代码实现

考虑以下数据:

x	y
$(3,3)^T$	1
$(4,3)^T$	1
$(1,1)^T$	-1

2.1 sklearn 实现

准备数据：

```
import numpy as np
X = np.array([[3,3],[4,3],[1,1]])
Y = np.array([1,1,-1]).reshape(-1,1)
```

感知机的 sklearn 接口位于 sklearn.linear_model.Perceptron, 其文档可见**此处**。其中较为重要的参数有：

- eta0：感知机的学习率

```
from sklearn.linear_model import Perceptron
clf = Perceptron(eta0=1)
clf.fit(X,Y)
```

```
print(clf.coef_,clf.intercept_,clf.n_iter_)
```

```
## [[1. 0.]] [-2.] 9
```

2.2 原问题实现

```
def Perceptron(X,Y,eta=1):
    N,dim = X.shape
```

```
    w,b = np.zeros(dim),0
    it = 0
    point = 0
    while it<N:
        if (np.sum(X[point,:]*w)+b)*Y[point,:]<=0:
            w = w+eta*Y[point,:]*X[point,:]
            b = b+eta*Y[point,:]
            it = 0
        else:
            it += 1
        point = (point+1)%N
    return w,b
Perceptron(X,Y,eta=1)
```

```
## (array([1., 1.]), array([-3]))
```

2.3 对偶问题实现

```
def Perceptron_dual(X,Y,eta=1):
    N,_ = X.shape
    alpha,b = np.zeros(N),0
    G = X.dot(X.T)
    it = 0
    point = 0
    while it<N:
        if (np.sum(alpha*G[point,:]*Y.reshape(-1))+b)*Y[point,:]<=0:
            alpha[point] = alpha[point]+eta
            b = b+eta*Y[point,:]
            it = 0
        else:
            it += 1
        point = (point+1)%N
    return alpha,b
Perceptron_dual(X,Y,eta=1)
```

```
## (array([2., 0., 5.]), array([-3]))
```