感知机建模及对偶形式

线性可分感知机

数据建模，每个样本$\mathrm{x}=[x_1,x_2,,\ldots x_n]$，二分类问题，对应标签$y\in\{-1,1\}$，构造线性分类器：

$$y=\operatorname{sign}\left(\mathrm{w}\cdot\mathrm{x}^T+b\right)$$

对任意一样本对(x_i,y_i)，则分类正确时满足：

$$y_i\left(\mathrm{w}\cdot\mathrm{x}^T+b\right)>0$$

对应分类错误时候时满足：

$$f(\mathrm{w},b)=y_i\left(\mathrm{w}\cdot\mathrm{x}^T+b\right)\leq 0$$

更新参数(w,b)使其大于0，即需要朝梯度是上升方向更新参数:

$$\frac{\partial f(\mathrm{w},b)}{\partial\mathrm{w}}=\mathrm{x}_i\cdot y_i$$
$$\frac{\partial f(\mathrm{w},b)}{\partial b}=y_i$$

对应更新方式为:

$$\mathbf{w}\leftarrow\mathbf{w}+\eta\cdot\mathrm{x}_i\cdot y_i$$
$$b\leftarrow\eta\cdot y_i$$

可以采用SGD进行优化即可

扩充权重向量

可以通过对x补1，即$\hat{\mathrm{x}}=[\mathrm{x},1]$ 从而把b并入w，此时

$$f(\hat{\mathrm{x}})=\operatorname{sign}(\mathrm{w}\cdot\hat{\mathrm{x}})$$

对应更新方式为

$$\mathbf{w}\leftarrow\mathbf{w}+\eta\cdot\hat{\mathbf{x}}_{\mathbf{i}}\cdot y_i$$

对偶形式

与《统计学习方法》里面略有不同，这里讨论扩充权重向量时的对偶形式，首先分析更新方式：

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \cdot \hat{\mathbf{x}}_{\mathbf{i}} \cdot y_i$$

初始化$\mathbf{w} = \vec{0}$，可以发现，最终的结果$\mathbf{w}$只与$(\hat{\mathbf{x}}_i, y_i)$被记为负压样本的次数$n_i$有关，那么可以通过如下形式表示$\mathbf{w}$：

$$\mathbf{w} = \sum_{i=0}^{k} \eta \cdot n_i \cdot \hat{\mathbf{x}}_i \cdot y_i = \sum_{i=0}^{k} \alpha_i \cdot y_i \cdot \hat{\mathbf{x}}_i$$

其中k 为样本数量，对应推理方程可以变为：

$$f(\hat{\mathbf{x}}) = \operatorname{sign}\left(\sum_{i=0}^{k} \alpha_i \cdot y_i \cdot \hat{\mathbf{x}}_i \times \mathbf{x}^T\right)$$

分类错误时：

$$\sum_{i=0}^{k} \alpha_i \cdot y_i \cdot \hat{\mathbf{x}}_i \times \hat{\mathbf{x}}_{\mathbf{j}}{}^T \cdot y_j \leq 0$$

此时第j个样本出了问题，类似原始问题，只需要让$n_j + 1$即可

$$\begin{aligned} & n_j \leftarrow n_j + 1 \\ & \alpha_j \leftarrow \eta(n_j + 1) = \alpha + \eta \\ & \alpha_j \leftarrow \alpha_j + \eta \end{aligned}$$

为什么使用对偶形式：

对偶形式训练的时候使用了$\hat{\mathbf{x}}_i \times \hat{\mathbf{x}}_j$ ，可以预先计算他们的值加速计算，Gram矩阵。

```
#[k, n + 1]
Extend_X = np.hstack([X, np.ones([X.shape[0], 1])])
# [k, k]
Gram = Extend_X.dot(Extend_X.T)
```

下面是扩充向量对偶形式的Python代码

```
import numpy as np
import random

```

```python

class Perceptron(object):
    def __init__(self,
                 max_iter=5000,
                 eta=1,
                 ):
        self.eta = eta
        self.max_iter_ = max_iter
        self.w = 0

    def fit(self, X, y):
        """
        X: [k, n]
        y: [k, ]
        compute w:[n + 1,]
        """
        # [1, k]
        self.alpha = np.zeros([1, X.shape[0]])
        n_iter_ = 0
        # [k, n + 1]
        Extend_X = np.hstack([X, np.ones([X.shape[0], 1])])
        # [k, k]
        self.Gram = Extend_X.dot(Extend_X.T)
        while n_iter_ < self.max_iter_:
            index = random.randint(0, y.shape[0] - 1)
            # \sum(\alpha x y_i x x_i x x_j)
            pred = self.alpha.dot(np.multiply(y, self.Gram[index, :]))
            # y_j x pred
            if y[index] * pred <= 0:
                self.alpha[0, index] += self.eta
            n_iter_ += 1
        # 恢复扩充权重向量
        self.w = self.alpha.dot(np.multiply(y, Extend_X.T).T)

    def predict(self, X):
        X = np.hstack([X, np.ones(X.shape[0]).reshape((-1, 1))])
        rst = np.array([1 if rst else -1 for rst in np.dot(X, self.w.T) > 0])
        return rst
```