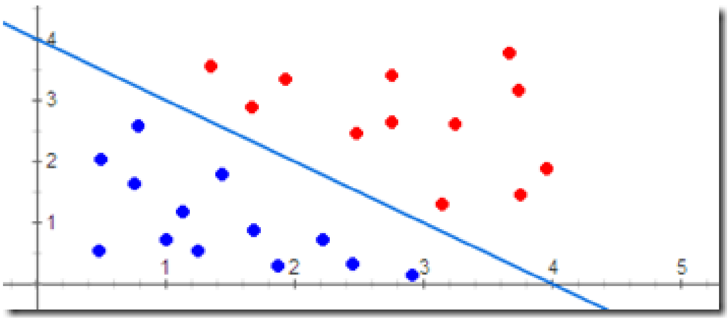
感知机

1.模型：

感知机是典型的二分类线性模型。其思想是在空间中找到一个分离超平面，将所有实例点按照其正确标记进行分割，输入为样本的特征向量，输出为样本的类别（+1，-1），这就需要一个映射函数来实现：

$$f(x) = sign(w \cdot x + b)$$

其中线性方程w·x + b = 0对应于问题空间中的一个超平面S，该平面将输入点分为两类



2.学习策略

感知机要求样本点是线性可分的。感知机的目标是确定w和b，能将所有点正确分类，需要定义一个经验损失函数。直观来看，可以选择误分类点的个数作为损失函数，但是这样的函数不是参数w，b的连续可到函数，数学性质不好，于是我们想到用误分类点到分离超平面的距离作为损失函数，总距离约小，模型越好。

输入空间中任一点到超平面S的距离公式为：

$$\frac{1}{\|w\|}|w \cdot x_0 + b|$$

这里||w||是w的范数。对所有的误分类点来说，

$$-y_i(w \cdot x_i + b) > 0$$

因此误分类点到超平面S的距离可以写作（保证了计算距离时的非负性）：

$$-\frac{1}{\|w\|}y_i(w \cdot x_i + b)$$

对误分类点集合M来说，总的就离为：

$$-\frac{1}{\|w\|}\sum_{x_i \in M} y_i(w \cdot x_i + b)$$

，因此，我们的目标函数为：

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

具体可以使用梯度下降算法求解w和b。

3.感知机学习算法：

3.1原始形式：

原始形式就是通过梯度下降不断更新w和b的值，在感知机算法中选用随机梯度下降法

算法1（感知机学习算法的原始形式）

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)\}$，其中$x_i\in\chi=R^n$，$y_i\in\Upsilon=\{-1,+1\}$，i = 1,2,...,N；学习率$\eta(0<\eta\le1)$

输出：w，b；感知机模型$f(x)=sign(w\cdot x+b)$

（1）选取初始值w_0，b_0

（2）在训练集中选取数据(x_i,y_i)

（3）如果$y_i(w\cdot x_i+b)\le0$（从公式（3）变换而来）

$$w\leftarrow w+\eta y_i x_i$$

$$b\leftarrow b+\eta y_i$$

（4）转至（2），直至训练集中没有误分类点

算法的收敛性证明没看

3.2对偶形式：

对偶形式的本质就是把w和b通过x和y线性表示出来。

从原始形式可以看出，不论如何选择样本点对w和b进行更新，肯定会使用到所有的样本点，而且在w和b的更新过程中，不同的样本点被使用的次数不同，离超平面越近的点更新的次数越多，这种点对学习结果影响越大。

因此，初始的w(b)和最终的w(b)之间的差值就等于所有点更新的增量的总和，每一次更新的增量为步长乘以偏导数，更新的总增量为：$\Delta w=\sum n_i\eta y_i x_i$，其中$n_i$代表第i个点的更新次数。如果w的初始值为0，则最终的w就等于Δw。因此最终学习到的w和b为：

$w=\sum n_i\eta y_i x_i$;

$b=\sum n_i\eta y_i$

算法2（感知机学习算法的对偶形式）

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)\}$，其中$x_i\in\chi=R^n$，$y_i\in\Upsilon=\{-1,+1\}$，i = 1,2,...,N；学习率$\eta(0<\eta\le1)$

输出：α，b；感知机模型$f(x)=sign(\sum_{j=1}^{N}\alpha_j y_j x_j\cdot x+b)$

其中$\alpha=(\alpha_1,\alpha_2,\dots,\alpha_N)^T$

（1）$\alpha\leftarrow0$，$b\leftarrow0$

（2）在训练集中选取样本(x_i,y_i)

（3）如果$y_i(\sum_{j=1}^{N}\alpha_j y_j x_j\cdot x_i+b)\le0$

$$\alpha_i\leftarrow\alpha_i+\eta$$

$$b\leftarrow b+\eta y_i$$

（4）转至（2）直到没有误分类样本出现

如果（3）条件满足，那就说明当前这个点的更新次数还没有达到最终的更新次数，需要把次数加1，也就是再走一个步长。

训练实例仅以内积的形式出现，可以提前将实例间的内积计算出来并以矩阵形式存储，着就是Gram矩阵

$$G=[x_i\cdot x_j]_{N\times N}$$