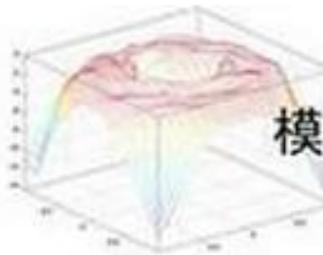


机器学习



模式识别



计算机视觉



数据挖掘



机器学习



统计学习

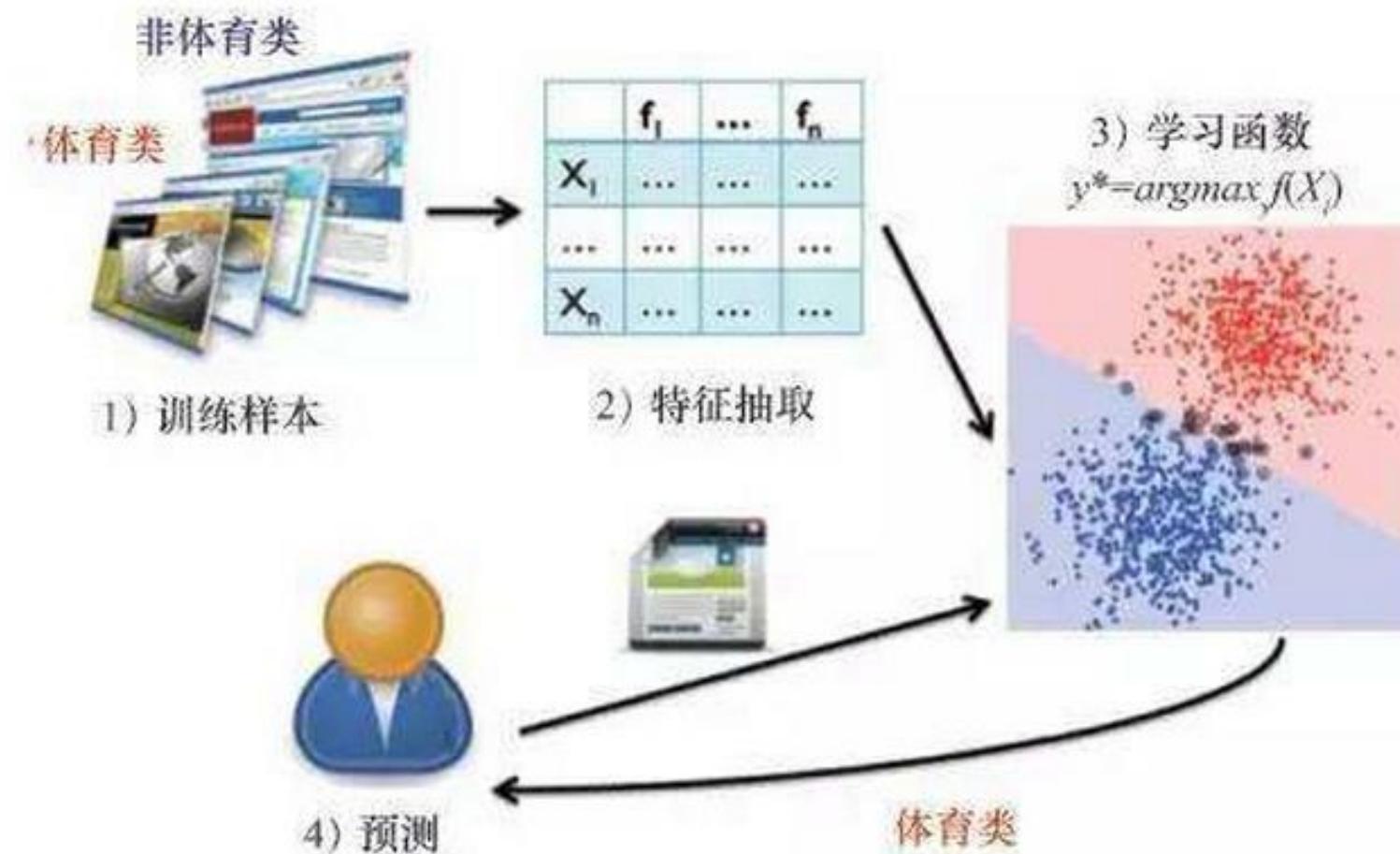


语音识别



自然语言处理

机器学习



Machine Learning

Numpy: 科学计算库

Pandas: 数据分析处理库

Matplotlib: 数据可视化库

Scikit-learn: 机器学习库

线性回归

工资	年龄	额度
4000	25	20000
8000	30	70000
5000	28	35000
7500	33	50000
12000	40	85000

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$h_{\theta}(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x$$

线性回归

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$$

误差 $\varepsilon^{(i)}$ 是独立并且具有相同的分布通常认为服从均值为0方差为 θ^2 的高斯分布

线性回归

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$p(y^{(i)} | x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$\begin{aligned} L(\theta) &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\ &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

线性回归

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2.\end{aligned}$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

线性回归

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2} (X\theta - y)^T (X\theta - y)$$

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \left(\frac{1}{2} (X\theta - y)^T (X\theta - y) \right) = \nabla_{\theta} \left(\frac{1}{2} (\theta^T X^T - y^T)(X\theta - y) \right)$$

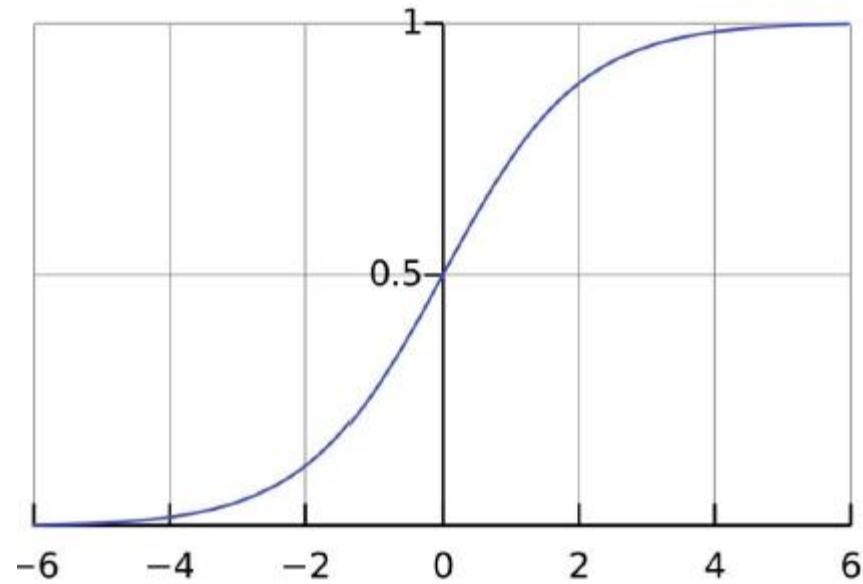
$$= \nabla_{\theta} \left(\frac{1}{2} (\theta^T X^T X \theta - \theta^T X^T y - y^T X \theta + y^T y) \right)$$

$$= \frac{1}{2} (2X^T X \theta - X^T y - (y^T X)^T) = X^T X \theta - X^T y$$

$$\theta = (X^T X)^{-1} X^T y$$

Logistic 回归

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$



Sigmoid 函数

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(x) = \left(\frac{1}{1 + e^{-x}} \right)' = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} = \frac{1}{1 + e^{-x}} \cdot \left(1 - \frac{1}{1 + e^{-x}} \right)$$

$$= g(x) \cdot (1 - g(x))$$

Machine Learning

$$h_{\theta}(x) = \theta_1 x + \theta_0$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i).$$

$$\frac{\partial J(\theta_1)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) * x_i$$

$$\theta_1 := \theta_1 - \alpha * \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} \quad \quad \theta_0 := \theta_0 - \alpha * \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1}$$

Logistic 回归

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

$$L(\theta) = p(\vec{y} \mid X; \theta)$$

$$= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}}$$

Logistic 回归

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

Logistic 回归

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\&= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x)) x_j \\&= (y - h_\theta(x)) x_j\end{aligned}$$

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

Machine Learning

$$h_{\theta}(x) = \theta_1 x + \theta_0$$

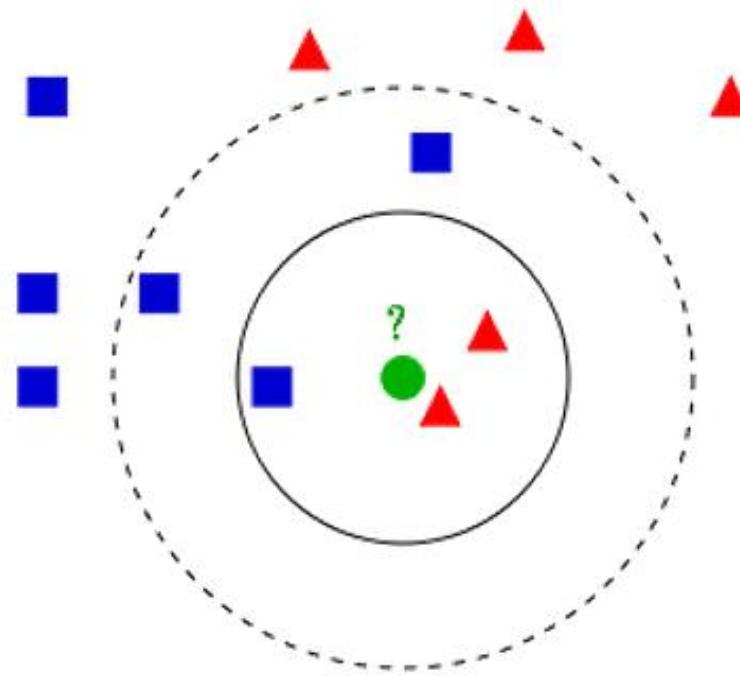
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i).$$

$$\frac{\partial J(\theta_1)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) * x_i$$

$$\theta_1 := \theta_1 - \alpha * \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} \quad \quad \theta_0 := \theta_0 - \alpha * \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1}$$

K-近邻



- 如果 $K=3$, 绿色圆点的最近的3个邻居是2个红色小三角形和1个蓝色小正方形, 少数从属于多数, 基于统计的方法, 判定绿色的这个待分类点属于红色的三角形一类。
- 如果 $K=5$, 绿色圆点的最近的5个邻居是2个红色三角形和3个蓝色的正方形, 还是少数从属于多数, 基于统计的方法, 判定绿色的这个待分类点属于蓝色的正方形一类。

K-近邻

对于未知类别属性数据集中的点：

- 1.计算已知类别数据集中的点与当前点的距离
- 2.按照距离依次排序
- 3.选取与当前点距离最小的K个点
- 4.确定前K个点所在类别的出现概率
- 5.返回前K个点出现频率最高的类别作为当前点预测分类。

K-近邻

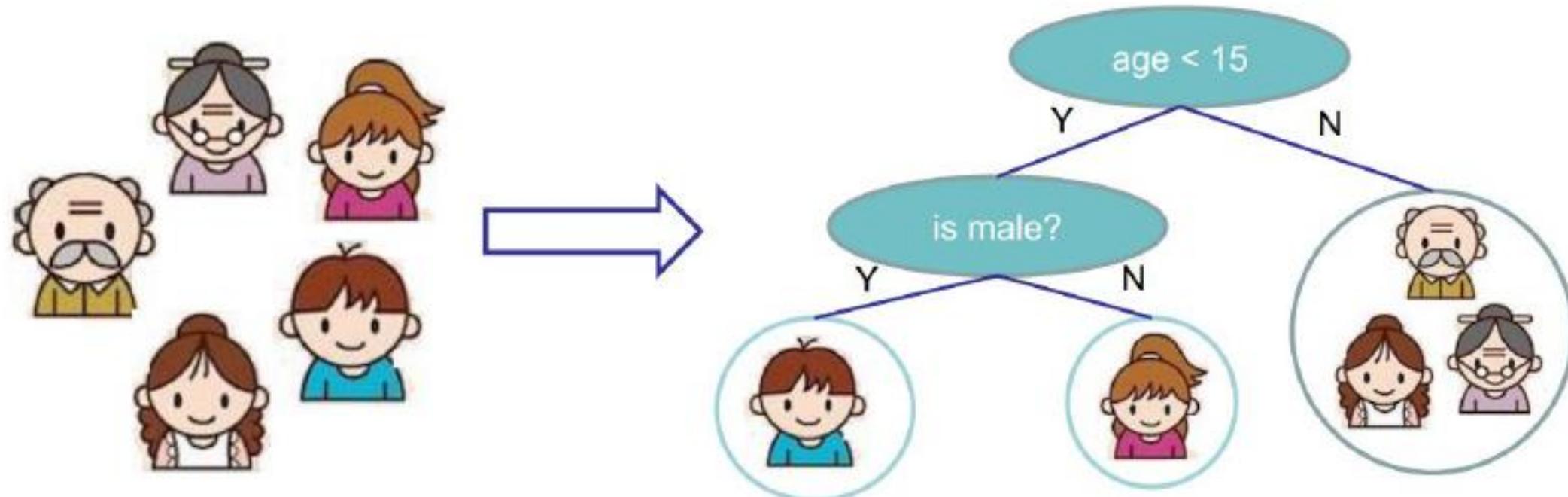
概述：

KNN 算法本身简单有效，它是一种 **lazy-learning** 算法。

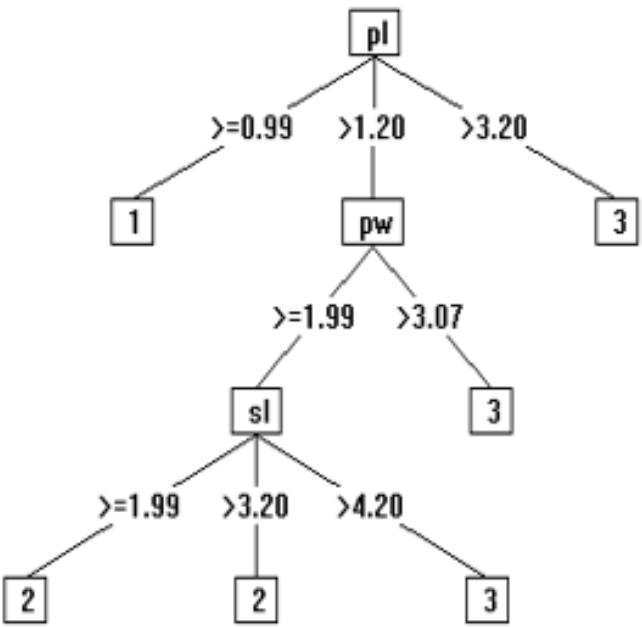
分类器不需要使用训练集进行训练，训练时间复杂度为0。

KNN 分类的计算复杂度和训练集中的文档数目成正比，也就是说，如果训练集中文档总数为 n ，那么 KNN 的分类时间复杂度为 $O(n)$ 。

决策树



决策树算法以树状结构表示数据分类的结果。每个决策点实现一个具有离散输出的测试函数，记为分支。

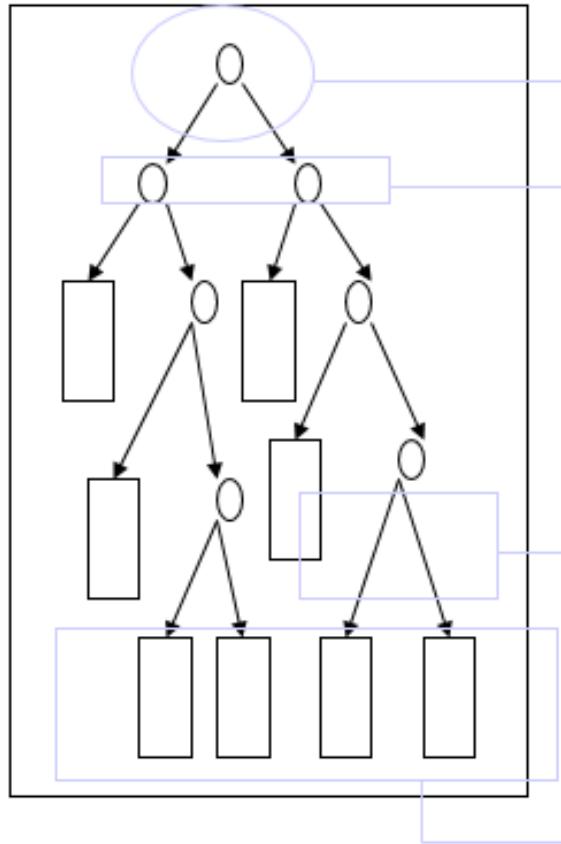


根节点

非叶子节点（决策点）

叶子节点

分支



根部节点 (root node)

非叶子节点 (non-leaf node)

(代表测试的条件, 对数据属性的测试)

分支 (branches) (代表测试的结果)

叶节点 (leaf node)

(代表分类后所获得的分类标记)

1. 训练阶段

从给定的训练数据集 DB ，构造出一棵决策树

$$class = DecisionTree(DB)$$

2. 分类阶段

从根开始，按照决策树的分类属性逐层往下划分，直到叶节点，获得概念（决策、分类）结果。

$$y = DecisionTree(x)$$

决策树-熵

$P(X,Y) = P(X)*P(Y)$ X 和 Y 两个事件相互独立 $\text{Log}(XY) = \text{Log}(X)+\text{Log}(Y)$

$H(X), H(Y)$ 当成它们发生的不确定性

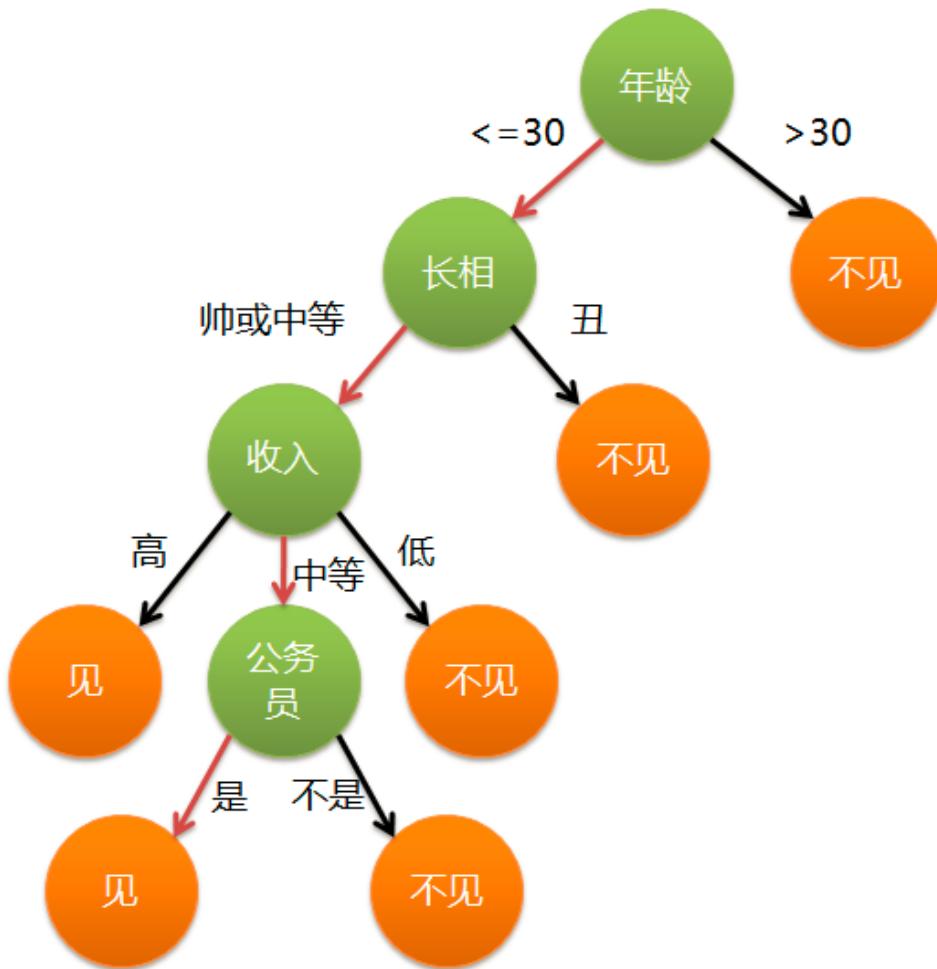
$P(\text{几率越大}) \rightarrow H(X)$ 值越小 如: 今天正常上课

$P(\text{几率越小}) \rightarrow H(X)$ 值越大 如: 今天没翻车

$$\text{熵} = -\sum_{i=1}^n P_i \ln(P_i)$$

$$\text{Gini系数} = Gini(p) = \sum_{k=1}^K p_k(1-p_k) = 1 - \sum_{k=1}^K p_k^2$$

决策树



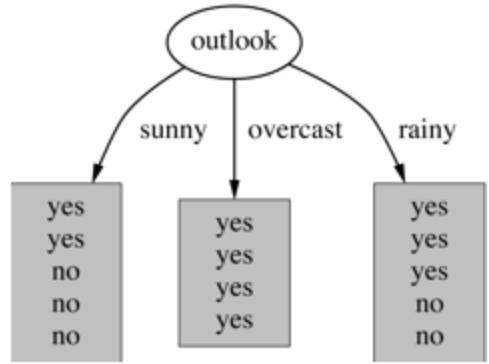
谁当根节点呢？

决策树

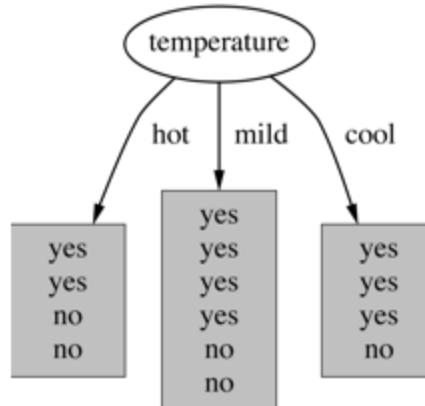
outlook	temperature	humidity	windy	play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

(14行数据，每个数据4个特征
outlook, temperature, humidity, windy)

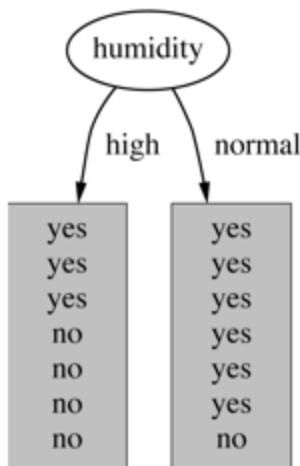
1. 基于天气的划分



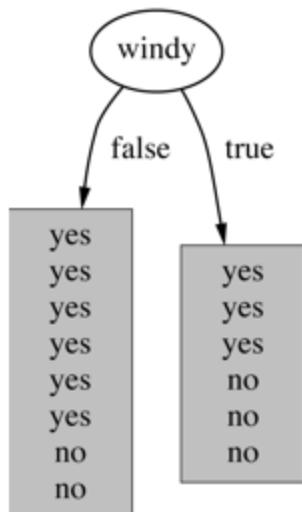
2. 基于温度的划分



3. 基于湿度的划分



4. 基于有风的划分



决策树

熵是无序性（或不确定性）的度量指标。假如事件A的全概率划分是 (A₁,A₂,...,A_n) , 每部分发生的概率是 (p₁,p₂,...,p_n) , 那信息熵定义为：

$$entropy(p_1, p_2, \dots, p_n) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \dots - p_n \log_2 p_n$$

决策树

构造树的基本想法是随着树深度的增加，节点的熵迅速地降低。熵降低的速度越快越好，这样我们有望得到一棵高度最矮的决策树。

在没有给定任何天气信息时，根据历史数据，我们只知道新的一天打球的概率是 $9/14$ ，不打的概率是 $5/14$ 。此时的熵为：

$$-\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.940$$

决策树

属性有4个：outlook , temperature , humidity , windy。我们首先要决定哪个属性作树的根节点。

对每项指标分别统计：在不同的取值下打球和不打球的次数。

下面我们计算当已知变量outlook的值时，信息熵为多少。

outlook=sunny时， $2/5$ 的概率打球， $3/5$ 的概率不打球。entropy=0.971

outlook=overcast时，entropy=0

outlook=rainy时，entropy=0.971

决策树

而根据历史统计数据，outlook取值为sunny、overcast、rainy的概率分别是 $5/14$ 、 $4/14$ 、 $5/14$ ，所以当已知变量outlook的值时，信息熵为： $5/14 \times 0.971 + 4/14 \times 0 + 5/14 \times 0.971 = 0.693$

这样的话系统熵就从 0.940 下降到了 0.693 ，信息增益gain(outlook)为 $0.940 - 0.693 = 0.247$

同样可以计算出gain(temperature)=0.029，gain(humidity)=0.152，gain(windy)=0.048。

gain(outlook)最大（即outlook在第一步使系统的信息熵下降得最快），所以决策树的根节点就取outlook。

决策树

接下来要确定N1取temperature、humidity还是windy?在已知outlook=sunny的情况下，根据历史数据，我们作出类似table 2的一张表，分别计算gain(temperature)、gain(humidity)和gain(windy)，选最大者为N1。

依此类推，构造决策树。当系统的信息熵降为0时，就没有必要再往下构造决策树了，此时叶子节点都是纯的--这是理想情况。最坏的情况下，决策树的高度为属性（决策变量）的个数，叶子节点不纯（这意味着我们要以一定的概率来作出决策）。

决策树

ID3: 信息增益

C4.5: 信息增益率

CART: Gini系数

评价函数: $C(T) = \sum_{t \in leaf} N_t \cdot H(t)$ (希望它越小越好,类似损失函数了)

-
- C4.5算法是ID3算法的扩展
 - 能够处理连续型的属性。首先将连续型属性离散化，把连续型属性的值分成不同的区间，依据是比较各个分裂点Gain值的大小。
 - 缺失数据的考虑：在构建决策树时，可以简单地忽略缺失数据，即在计算增益时，仅考虑具有属性值的记录。

选取(连续值的)哪个分界点?

■ 贪婪算法!

I. 排序

60 70 75 85 90 95 100 120 125 220

若进行“**二分**”，则可能有9个分界点。

例子：

60 70 75 85 90 95 100 120 125 220

↑ 分割成 $\text{TaxIn} \leq 80$ 和 $\text{TaxIn} > 80$

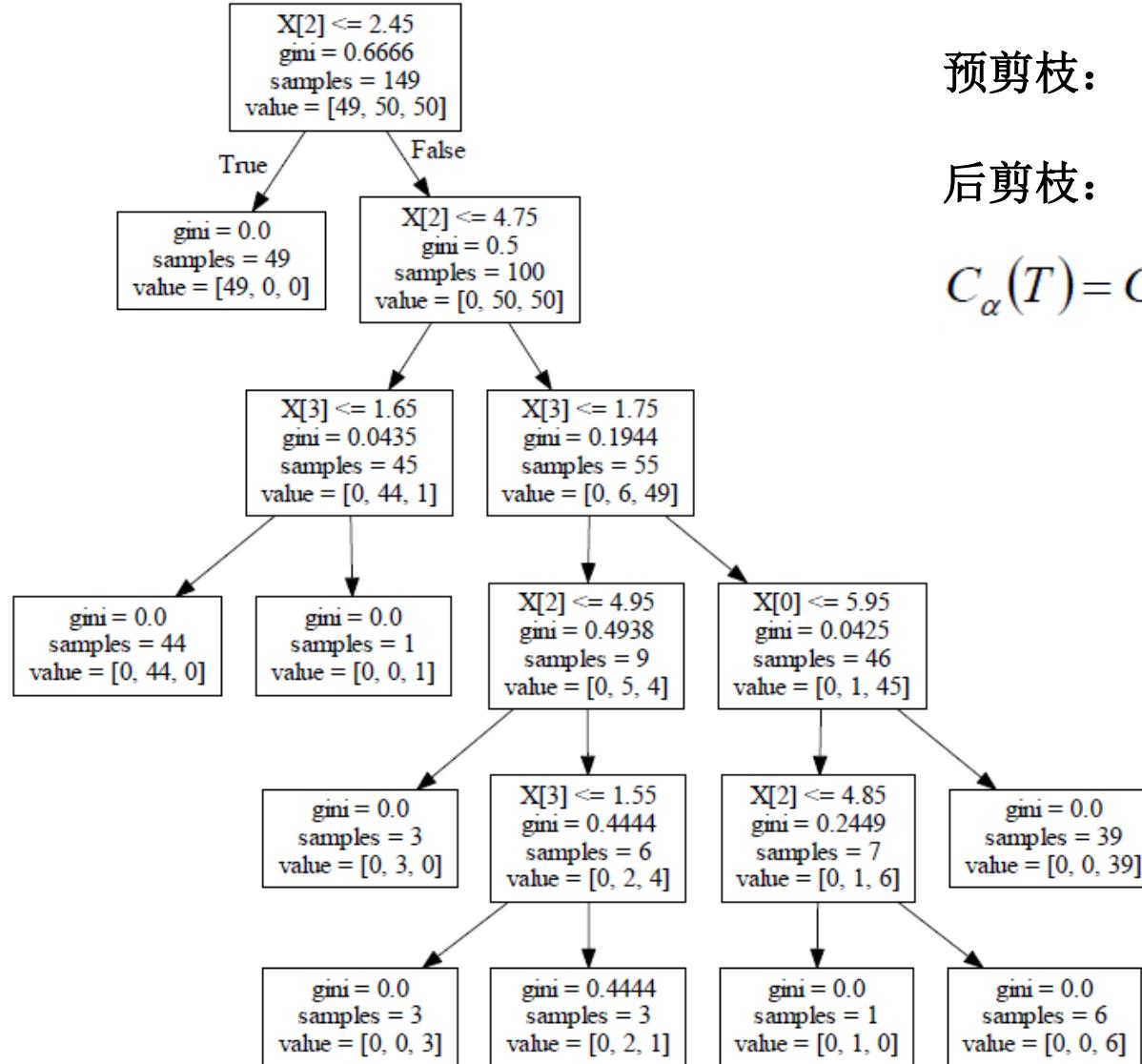
60 70 75 85 90 95 100 120 125 220

↑ 分割成 $\text{TaxIn} \leq 97.5$ 和 $\text{TaxIn} > 97.5$

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

⊕ 实际上，这就是“离散化”过程

决策树



预剪枝： 在构建决策树的过程时， 提前停止。

后剪枝： 决策树构建好后， 然后才开始裁剪。

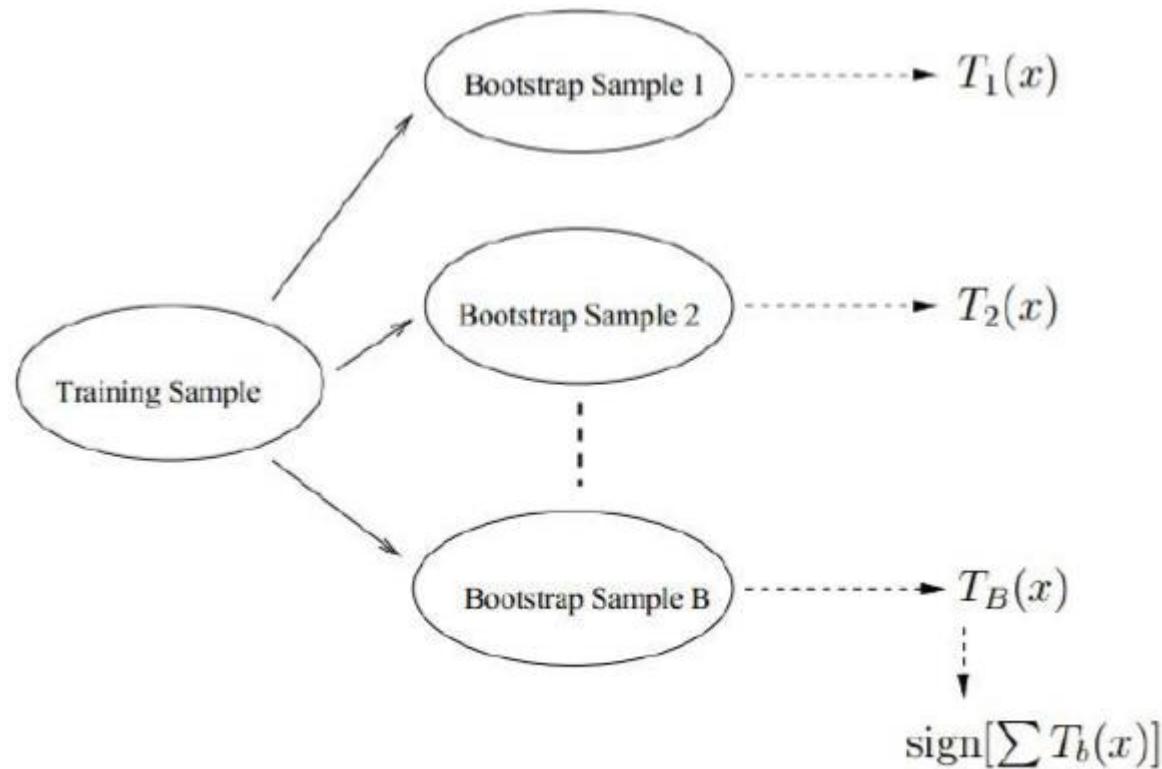
$$C_{\alpha}(T) = C(T) + \alpha \cdot |T_{leaf}|$$

叶子节点个数越多， 损失越大

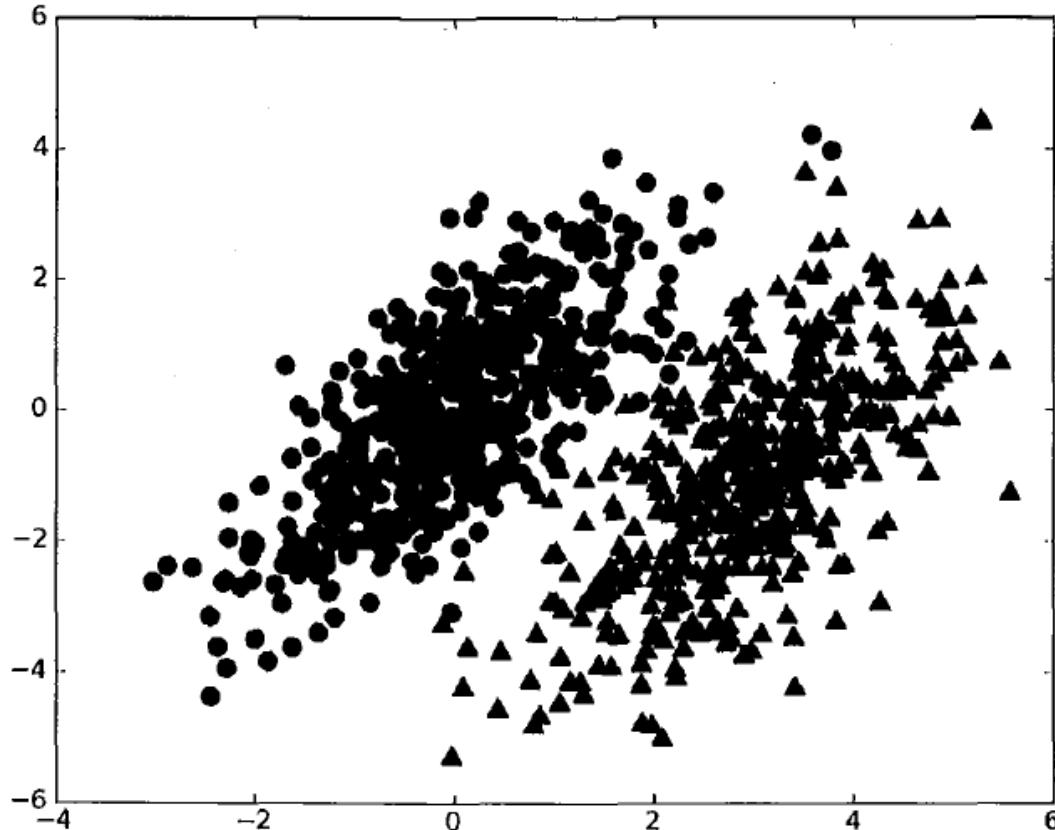
决策树

Bootstraping: 有放回采样

Bagging: 有放回采样n个样本一共建立分类器

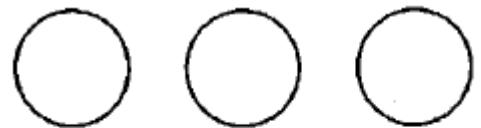


贝叶斯

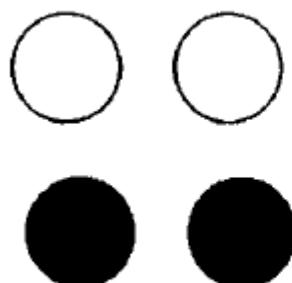


如果 $p_1(x, y) > p_2(x, y)$, 那么类别为1
如果 $p_2(x, y) > p_1(x, y)$, 那么类别为2

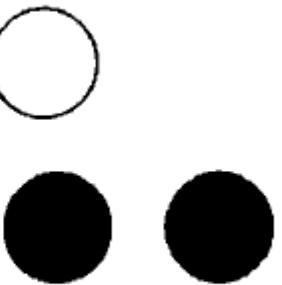
贝叶斯



$$P(\text{gray} | \text{bucketB}) = P(\text{gray and bucketB}) / P(\text{bucketB})$$



A桶



B桶

贝叶斯

我简单的介绍一下它的工作原理. 给定一个单词, 我们的任务是选择和它最相似的拼写正确的单词. (如果这个单词本身拼写就是正确的, 那么最近的就是它自己啦). 当然, 不可能绝对的找到相近的单词, 比如说给定 `lates` 这个单词, 它应该更正为 `late` 呢还是 `latest` 呢? 这些困难指示我们, 需要使用概率论, 而不是基于规则的判断. 我们说, 给定一个词 w , 在所有正确的拼写词中, 我们想要找一个正确的词 c , 使得对于 w 的条件概率最大, 也就是说:

$$\operatorname{argmax}_c P(c|w)$$

贝叶斯

$$\operatorname{argmax}_c P(c|w)$$

$$\operatorname{argmax}_c P(w|c) P(c) / P(w)$$

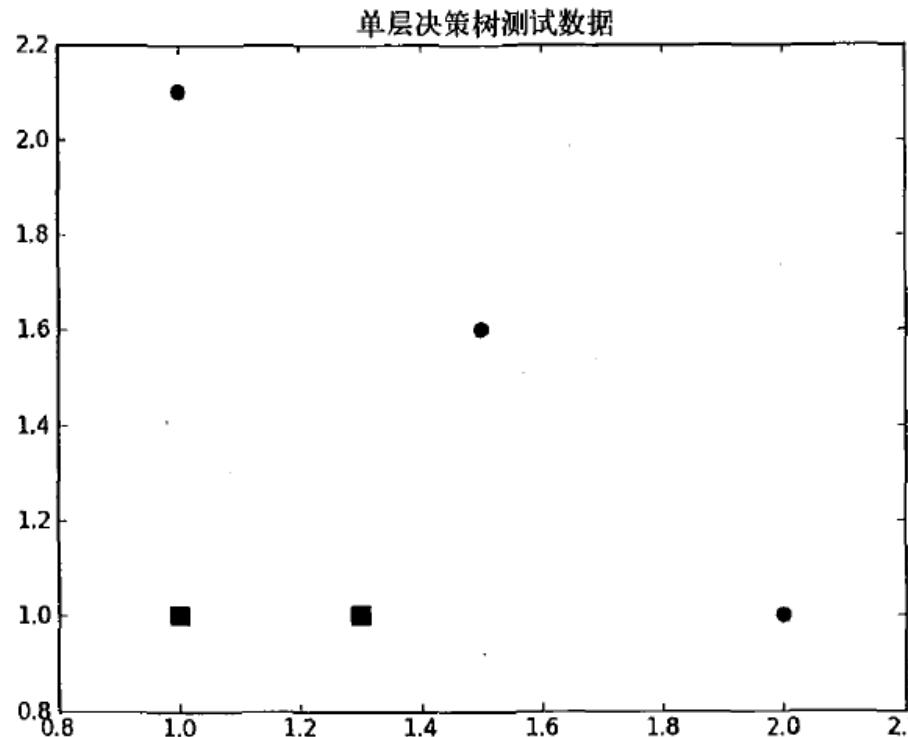
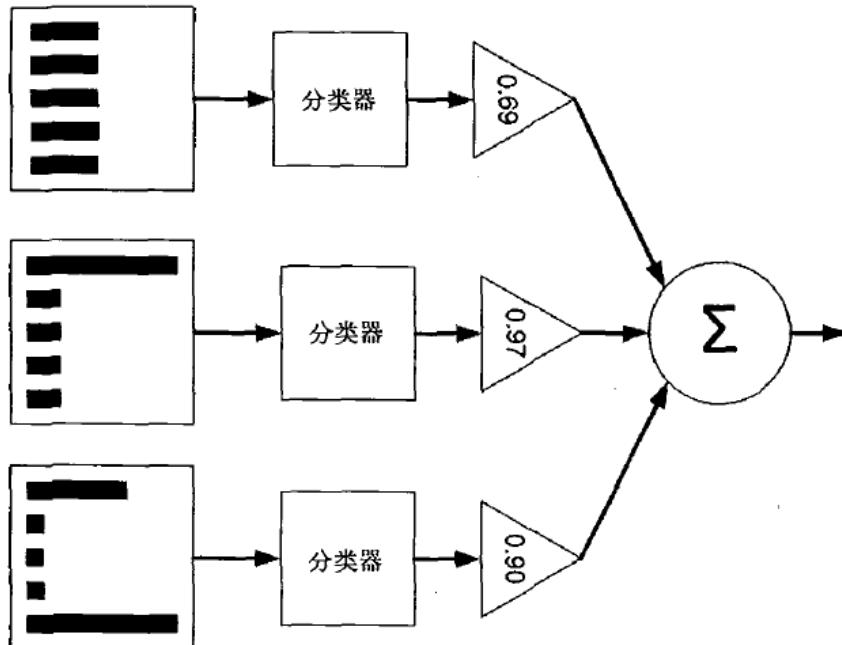
因为用户可以输错任何词，因此对于任何 c 来讲，出现 w 的概率 $P(w)$ 都是一样的，从而我们在上式中忽略它，写成：

$$\operatorname{argmax}_c P(w|c) P(c)$$

1. $P(c)$, 文章中出现一个正确拼写词 c 的概率, 也就是说, 在英语文章中, c 出现的概率有多大呢? 因为这个概率完全由英语这种语言决定, 我们称之为做语言模型. 好比说, 英语中出现 the 的概率 $P(\text{'the'})$ 就相对高, 而出现 $P(\text{'zxzxzxzyy'})$ 的概率接近0(假设后者也是一个词的话).
2. $P(w|c)$, 在用户想键入 c 的情况下敲成 w 的概率. 因为这个是代表用户会以多大的概率把 c 敲错成 w , 因此这个被称为**误差模型**.
3. argmax_c 用来枚举所有可能的 c 并且选取概率最大的, 因为我们有理由相信, 一个(正确的)单词出现的频率高, 用户又容易把它敲成另一个错误的单词, 那么, 那个敲错的单词应该被更正为这个正确的.

Adaboost

AdaBoost，是英文"Adaptive Boosting"（自适应增强）的缩写，由Yoav Freund和Robert Schapire在1995年提出。它的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。



Adaboost

1. 初始化训练数据的权值分布。如果有 N 个样本，则每一个训练样本最开始时都被赋予相同的权重： $1/N$ 。
2. 训练弱分类器。具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权重就被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到提高。然后，权重更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。
3. 将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

Adaboost

给定一个训练数据集 $T = \{(x_1, y_1), (x_2, y_2) \dots (x_N, y_N)\}$ ，其中实例 $x \in \mathcal{X}$ ，而实例空间 $\mathcal{X} \subset \mathbb{R}^n$ ， y_i 属于标记集合 $\{-1, +1\}$ ，Adaboost 的目的就是从训练数据中学习一系列弱分类器或基本分类器，然后将这些弱分类器组合成一个强分类器。

Adaboost 的算法流程如下：

步骤 1. 首先，初始化训练数据的权值分布。每一个训练样本最开始时都被赋予相同的权重： $1/N$ 。

$$D_1 = (w_{11}, w_{12} \dots w_{1i} \dots, w_{1N}), \quad w_{1i} = \frac{1}{N}, \quad i = 1, 2, \dots, N$$

Adaboost

步骤2. 进行多轮迭代，用 $m = 1, 2, \dots, M$ 表示迭代的第多少轮

a. 使用具有权值分布 D_m 的训练数据集学习，得到基本分类器：

$$G_m(x) : \chi \rightarrow \{-1, +1\}$$

b. 计算 $G_m(x)$ 在训练数据集上的分类误差率

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^N w_{mi} I(G_m(x_i) \neq y_i)$$

由上述式子可知， $G_m(x)$ 在训练数据集上的误差率 e_m 就是被 $G_m(x)$ 误分类样本的权值之和。

Adaboost

c. 计算 $G_m(x)$ 的系数， α_m 表示 $G_m(x)$ 在最终分类器中的重要程度（目的：得到基本分类器在最终分类器中所占的权重）：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m}$$

由上述式子可知， $e_m \leq 1/2$ 时， $\alpha_m \geq 0$ ，且 α_m 随着 e_m 的减小而增大，意味着分类误差率越小的基本分类器在最终分类器中的作用越大。

d. 更新训练数据集的权值分布（目的：得到样本的新的权值分布），用于下一轮迭代

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,N}),$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

使得被基本分类器 $G_m(x)$ 误分类样本的权值增大，而被正确分类样本的权值减小。就这样，通过这样的方式，AdaBoost方法能“聚焦于”那些较难分的样本上。

其中， Z_m 是规范化因子，使得 D_{m+1} 成为一个概率分布：

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

Adaboost

步骤3. 组合各个弱分类器

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x)$$

从而得到最终分类器，如下：

$$G(x) = sign(f(x)) = sign\left(\sum_{m=1}^M \alpha_m G_m(x)\right)$$

Adaboost

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1

迭代过程1

对于m=1，在权值分布为**D1**（10个数据，每个数据的权值皆初始化为0.1）的训练数据上，经过计算可得：

1. 阈值v取2.5时误差率为0.3 ($x < 2.5$ 时取1, $x > 2.5$ 时取-1，则6 7 8分错，误差率为0.3) ,
2. 阈值v取5.5时误差率最低为0.4 ($x < 5.5$ 时取1, $x > 5.5$ 时取-1，则3 4 5 6 7 8皆分错，误差率0.6大于0.5，不可取。故令 $x > 5.5$ 时取1, $x < 5.5$ 时取-1，则0 1 2 9分错，误差率为0.4) ,
3. 阈值v取8.5时误差率为0.3 ($x < 8.5$ 时取1, $x > 8.5$ 时取-1，则3 4 5分错，误差率为0.3)。

所以无论阈值v取2.5，还是8.5，总得分错3个样本，故可任取其中任意一个如2.5，弄成第一个基本分类器为：

Adaboost

$$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$

上面说阈值v取2.5时则6 7 8分错，所以误差率为0.3

从而得到 $G_1(x)$ 在训练数据集上的误差率（被 $G_1(x)$ 误分类样本“6 7 8”的权值之和） $e_1 = P(G_1(x_i) \neq y_i) = 3 * 0.1 = 0.3$ 。

然后根据误差率 e_1 计算 G_1 的系数：

$$\alpha_1 = \frac{1}{2} \log \frac{1 - e_1}{e_1} = 0.4236$$

这个 α_1 代表 $G_1(x)$ 在最终的分类函数中所占的权重，为0.4236。

Adaboost

接着更新训练数据的权值分布，用于下一轮迭代：

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,N})$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

第一轮迭代后，最后得到各个数据新的权值分布 $\mathbf{D2} = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715)$ 。由此可以看出，因为样本中是数据“6 7 8”被 $G_1(x)$ 分错了，所以它们的权值由之前的0.1增大到0.1666，反之，其它数据皆被分正确，所以它们的权值皆由之前的0.1减小到0.0715。

分类函数 $f_1(x) = a_1 * G_1(x) = 0.4236G_1(x)$ 。

此时，得到的第一个基本分类器 $\text{sign}(f_1(x))$ 在训练数据集上有3个误分类点（即6 7 8）。

从上述第一轮的整个迭代过程可以看出：被误分类样本的权值之和影响误差率，误差率影响基本分类器在最终分类器中所占的权重。

Adaboost

迭代过程2

对于 $m=2$ ，在权值分布为**D2 = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715)**的训练数据上，经过计算可得：

1. 阈值v取2.5时误差率为 $0.1666*3$ ($x < 2.5$ 时取1, $x > 2.5$ 时取-1，则6 7 8分错，误差率为 $0.1666*3$)，
2. 阈值v取5.5时误差率最低为 $0.0715*4$ ($x > 5.5$ 时取1, $x < 5.5$ 时取-1，则0 1 2 9分错，误差率为 $0.0715*3 + 0.0715$)，
3. 阈值v取8.5时误差率为 $0.0715*3$ ($x < 8.5$ 时取1, $x > 8.5$ 时取-1，则3 4 5分错，误差率为 $0.0715*3$)。

所以，阈值v取8.5时误差率最低，故第二个基本分类器为：

$$G_2(x) = \begin{cases} 1, & x < 8.5 \\ -1, & x > 8.5 \end{cases}$$

Adaboost

计算G2的系数：

$$\alpha_2 = \frac{1}{2} \log \frac{1 - e_2}{e_2} = 0.6496$$

更新训练数据的权值分布：

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,N})$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

D3 = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.01667, 0.1060, 0.1060, 0.1060, 0.0455),
.....

Adaboost

迭代过程3

$$G_3(x) = \begin{cases} 1, & x < 5.5 \\ -1, & x > 5.5 \end{cases}$$

$$\alpha_3 = \frac{1}{2} \log \frac{1-e_3}{e_3} = 0.7514$$

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,N})$$

$$w_{m+i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

$$\mathbf{D4} = (\underline{0.125}, \underline{0.125}, \underline{0.125}, 0.102, 0.102, 0.102, 0.065, 0.065, 0.065, \underline{0.125})$$

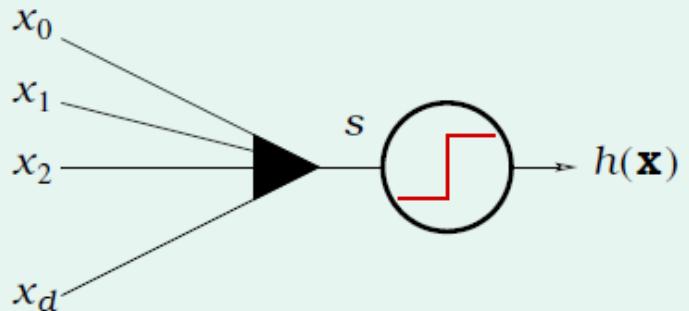
Adaboost

$$G(x) = \text{sign}[f_3(x)] = \text{sign}[a_1 * G_1(x) + a_2 * G_2(x) + a_3 * G_3(x)]$$

$$G(x) = \text{sign}[0.4236G_1(x) + 0.6496G_2(x) + 0.7514G_3(x)].$$

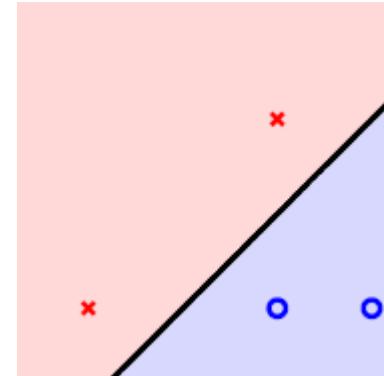
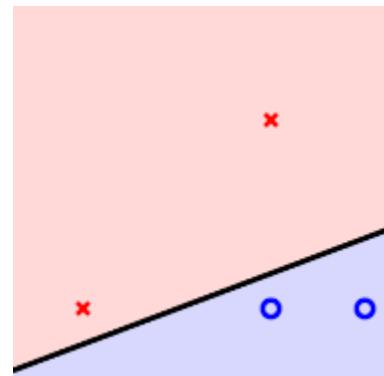
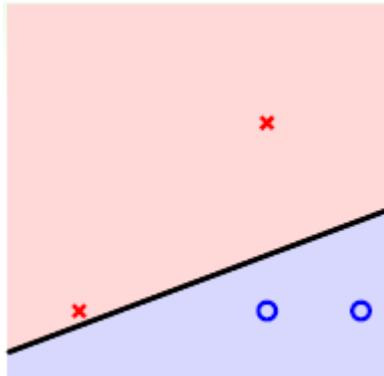
如何选择一条线呢？

$$h(\mathbf{x}) = \text{sign}(s)$$

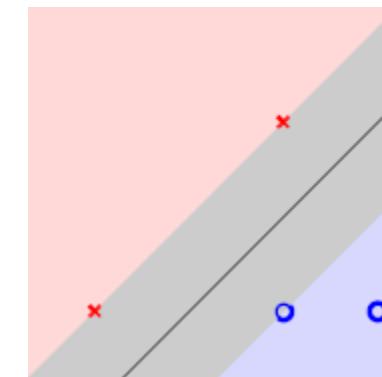
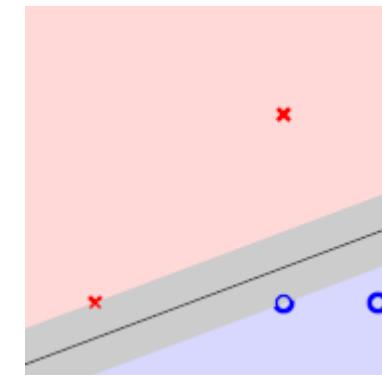
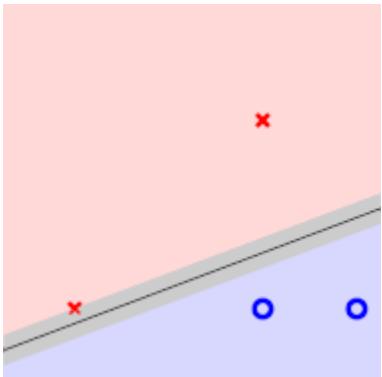
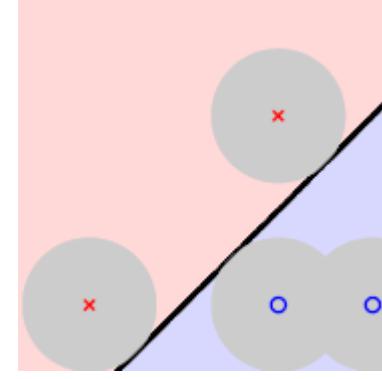
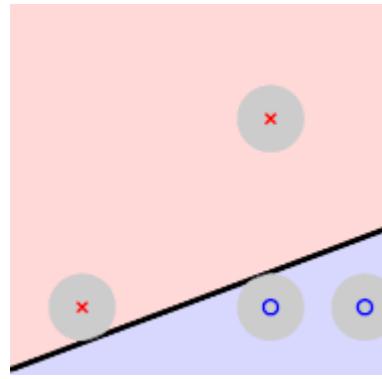
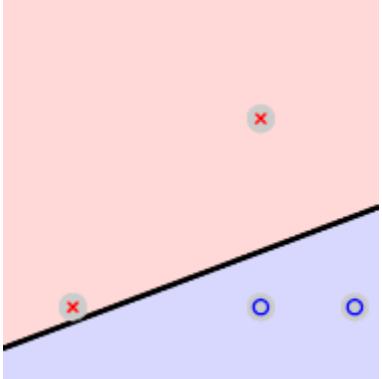


plausible err = 0/1
(small flipping noise)
minimize **specially**

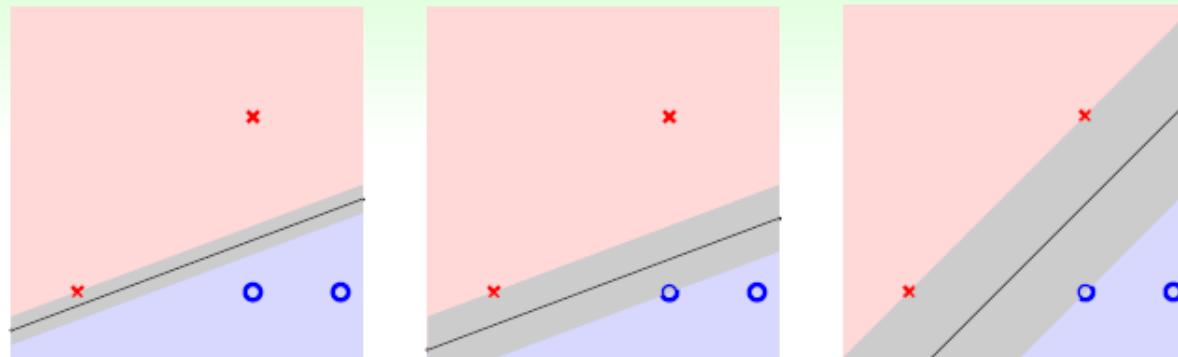
linear (hyperplane) classifiers:
 $h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$



越胖越好



目的



$$\max_{\mathbf{w}} \text{fatness}(\mathbf{w})$$

subject to \mathbf{w} classifies every (\mathbf{x}_n, y_n) correctly

$$\text{fatness}(\mathbf{w}) = \min_{n=1, \dots, N} \text{distance}(\mathbf{x}_n, \mathbf{w})$$

- fatness: formally called **margin**
- correctness: $y_n = \text{sign}(\mathbf{w}^T \mathbf{x}_n)$

goal: find **largest-margin separating** hyperplane

want: distance(\mathbf{x} , b , \mathbf{w}), with hyperplane $\mathbf{w}^T \mathbf{x}' + b = 0$

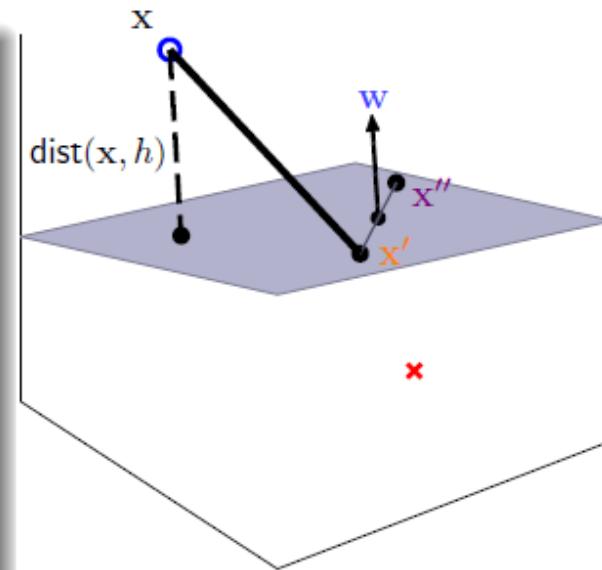
consider \mathbf{x}' , \mathbf{x}'' on hyperplane

① $\mathbf{w}^T \mathbf{x}' = -b$, $\mathbf{w}^T \mathbf{x}'' = -b$

② $\mathbf{w} \perp$ hyperplane:

$$\begin{pmatrix} \mathbf{w}^T & (\underbrace{\mathbf{x}'' - \mathbf{x}'}_{\text{vector on hyperplane}}) \end{pmatrix} = 0$$

③ distance = project $(\mathbf{x} - \mathbf{x}')$ to \perp hyperplane



$$\text{distance}(\mathbf{x}, b, \mathbf{w}) = \left| \frac{\mathbf{w}^T}{\|\mathbf{w}\|} (\mathbf{x} - \mathbf{x}') \right| \stackrel{①}{=} \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T \mathbf{x} + b|$$

数据集 $(x_1, y_1), (x_2, y_2)$ 到 (x_n, y_n)

y 为样本的类别：

当 x 为正例时候 $y = +1$

当 x 为负例时候 $y = -1$

找到一个条线 (w 和 b)，使得离该线最近的点能够最远
 $\text{argmax}(w, b)$ 使得 \min (最近的点到该线的距离)

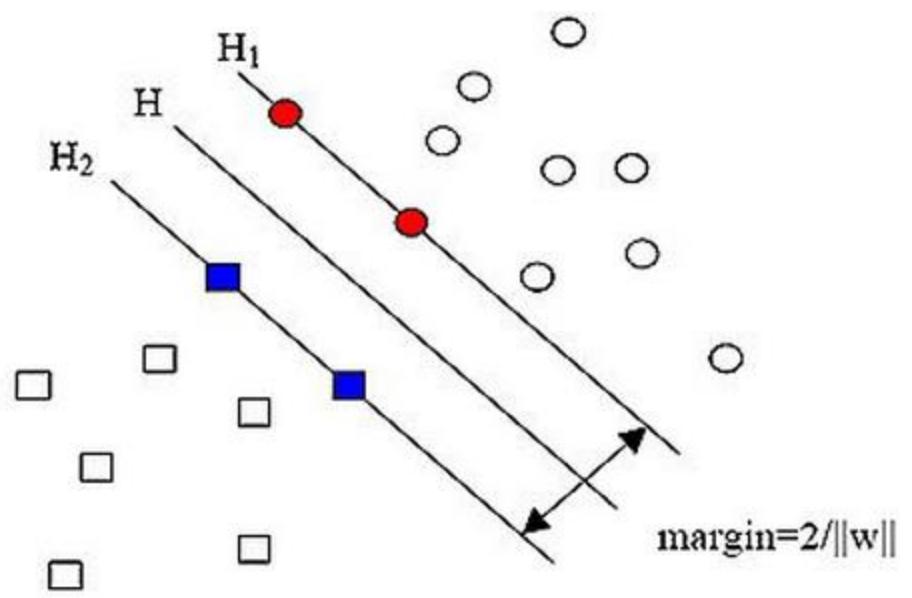
$$y(x) = w^T \Phi(x) + b$$

$$y(x_i) > 0 \Leftrightarrow y_i = +1$$

$$y(x_i) < 0 \Leftrightarrow y_i = -1$$

可推出 $y_i \cdot y(x_i) > 0$

$$\frac{y_i \cdot (w^T \cdot \Phi(x_i) + b)}{\|w\|}$$



对于线 (w, b) 可以通过放缩使得其结果值 $|Y| \geq 1$

$$y_i \cdot (w^T \cdot \Phi(x_i) + b) \geq 1$$

$$\arg \max_{w,b} \left\{ \frac{1}{\|w\|} \min_i [y_i \cdot (w^T \cdot \Phi(x_i) + b)] \right\}$$

$$\arg \max_{w,b} \frac{1}{\|w\|} \quad (\text{搞定目标函数})$$

目标函数: $\max_{w,b} \frac{1}{\|w\|}$ 且 $y_i(w^T \cdot \Phi(x_i) + b) \geq 1$

转换成求最小值 $\min_{w,b} \frac{1}{2} w^2$ 且 $y_i(w^T \cdot \Phi(x_i) + b) \geq 1$

拉格朗日乘子法标准格式:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

拉格朗日乘子法

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T \cdot \Phi(x_i) + b) - 1)$$

对偶问题：

$$\min_{w,b} \max_{\alpha} L(w,b,\alpha) \rightarrow \max_{\alpha} \min_{w,b} L(w,b,\alpha)$$

分别对w和b求偏导,分别得到两个条件

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^n \alpha_i y_i \Phi(x_i)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow 0 = \sum_{i=1}^n \alpha_i y_i$$

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T \Phi(x_i) + b) - 1)$$

$$= \frac{1}{2} w^T w - w^T \sum_{i=1}^n \alpha_i y_i \Phi(x_i) - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i$$

$$= \sum_{i=1}^n \alpha_i - \frac{1}{2} \left(\sum_{i=1}^n \alpha_i y_i \Phi(x_i) \right)^T \sum_{i=1}^n \alpha_i y_i \Phi(x_i)$$

$$= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \Phi^T(x_i) \Phi(x_j)$$

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T \cdot \Phi(x_i) + b) - 1) \quad \xrightarrow{\text{完成了第一步求解}} \min_{w, b} L(w, b, \alpha)$$

$$\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i, j=1}^n \alpha_i \alpha_j y_i y_j \Phi^T(x_i) \Phi(x_j)$$

继续对 α 求极大值 $\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j))$

条件: $\sum_{i=1}^n \alpha_i y_i = 0$

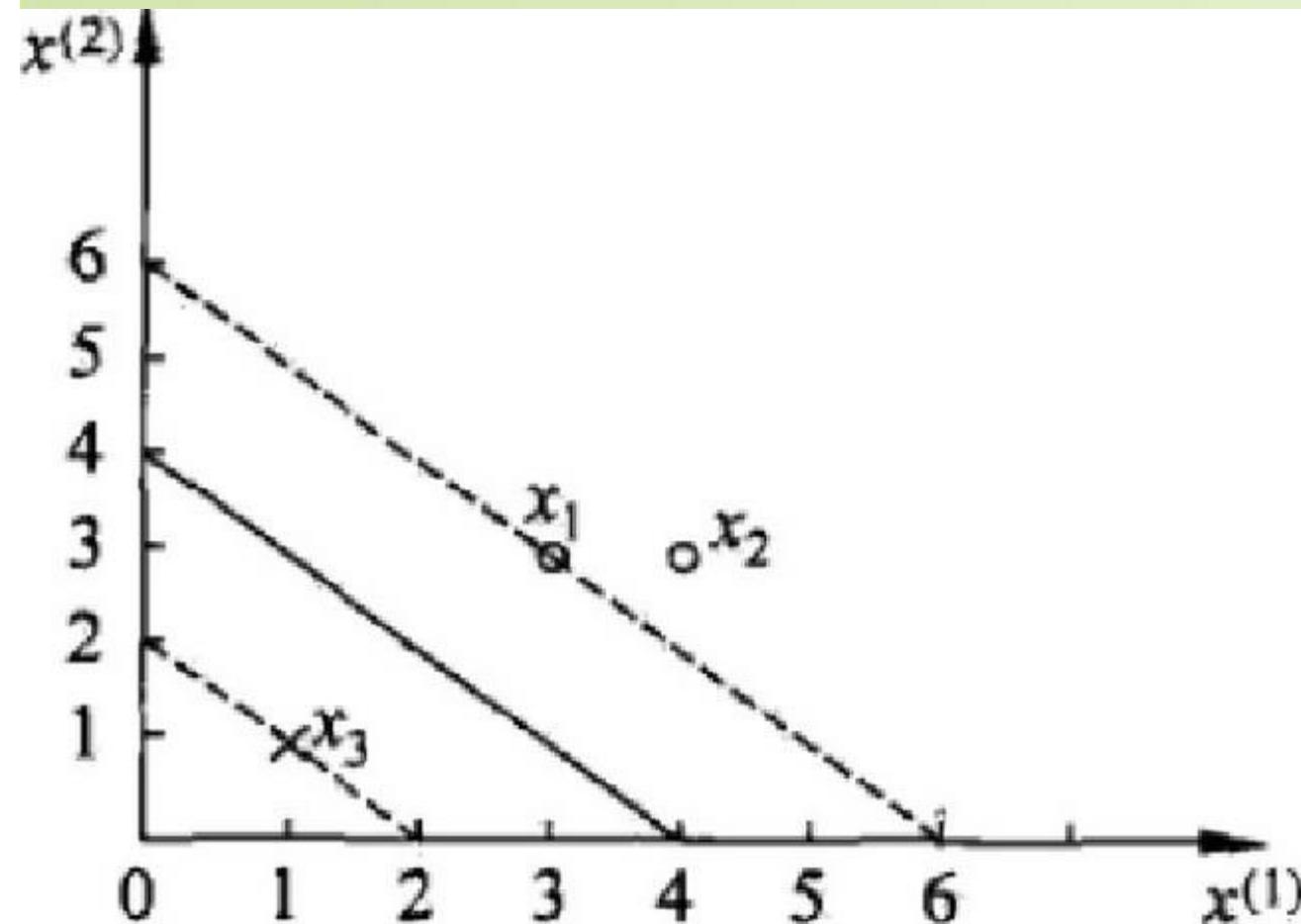
$$\alpha_i \geq 0$$

极大值转换成求极小值 $\min_{\alpha} \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j)) - \sum_{i=1}^n \alpha_i$

条件: $\sum_{i=1}^n \alpha_i y_i = 0$

$$\alpha_i \geq 0$$

例1. 已知一个如图所示的训练数据集，其正例点是 $x_1 = (3, 3)^T$
 $x_2 = (4, 3)^T$ ，负例点是 $x_3 = (1, 1)^T$ ，试求最大间隔分离超平面。



样本: $X_1(3,3,1)$ $X_2(4,3,1)$ $X_3(1,1,-1)$

求解: $\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^n \alpha_i$

$$\alpha_1 + \alpha_2 - \alpha_3 = 0$$

$$\alpha_i \geq 0, \quad i = 1, 2, 3$$

$$\frac{1}{2} (18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 42\alpha_1\alpha_2 - 12\alpha_1\alpha_3 - 14\alpha_2\alpha_3) - \alpha_1 - \alpha_2 - \alpha_3$$

$$\frac{1}{2}(18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 42\alpha_1\alpha_2 - 12\alpha_1\alpha_3 - 14\alpha_2\alpha_3) - \alpha_1 - \alpha_2 - \alpha_3$$

$$\alpha_1 + \alpha_2 = \alpha_3 \rightarrow 4\alpha_1^2 + \frac{13}{2}\alpha_2^2 + 10\alpha_1\alpha_2 - 2\alpha_1 - 2\alpha_2$$

分别对参数进行求导得： $\alpha_1 = 1.5$ $\alpha_2 = -1$ \rightarrow 不满足条件 $\alpha_i \geq 0, i = 1, 2, 3$

最终的解应该为边界上的点 $\rightarrow \alpha_1 = 0, \alpha_2 = -2/13 \rightarrow$ 带入原式=-0.153

$\alpha_1 = 0.25$ $\alpha_2 = 0 \rightarrow$ 带入原式=-0.25

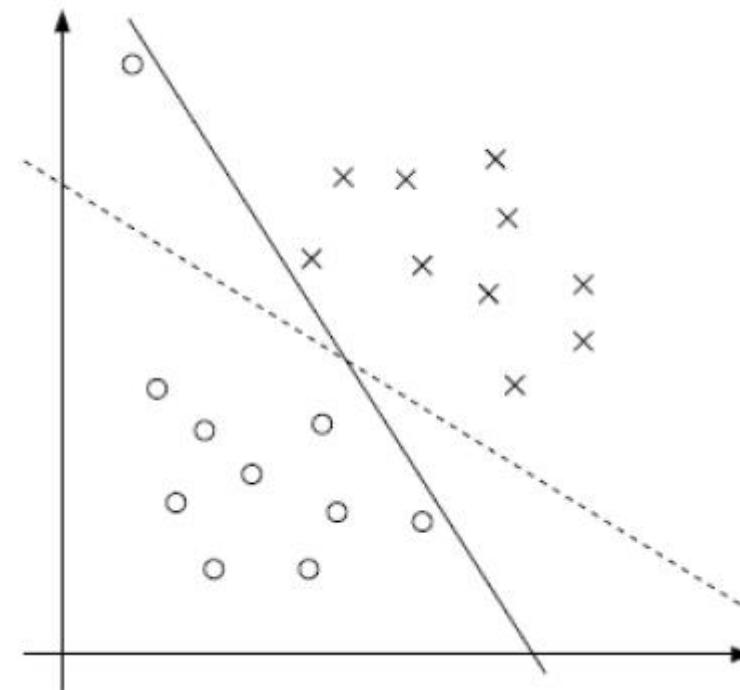
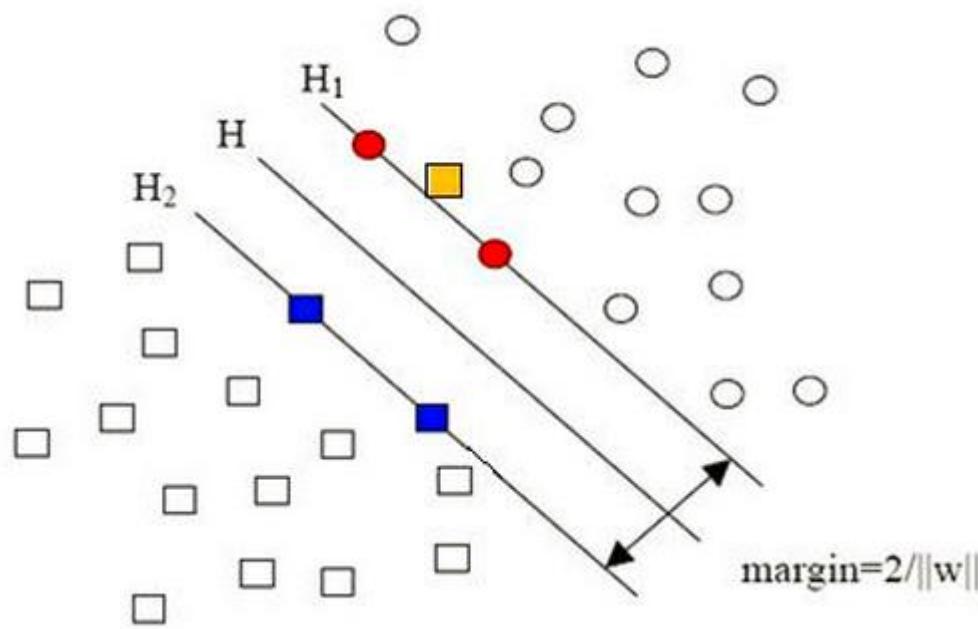
最小值在(0.25,0,0.25)处取得

对于 α 值(0.25,0,0.25)

$$w^* = \sum_{i=1}^N \alpha_i^* y_i \Phi(x_i) \rightarrow w_1=w_2=0.5$$

$$b^* = y_i - \sum_{i=1}^N \alpha_i^* y_i (\Phi(x_i) \cdot \Phi(x_j)) \rightarrow b=-2$$

$$0.5 \times 1 + 0.5 \times 2 - 2 = 0$$



为了解决该问题，引入松弛因子

当C趋于无穷大时：意味着分类严格不能有错误
当C趋于很小的时：意味着可以有更大的错误容忍

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

目标函数： $\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$

$$L(w, b, \xi, \alpha, \mu) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i (y_i(w \cdot x_i + b) - 1 + \xi_i) - \sum_{i=1}^n \mu_i \xi_i$$

$$\begin{aligned} w &= \sum_{i=1}^n \alpha_i y_i \phi(x_i) \\ 0 &= \sum_{i=1}^n \alpha_i y_i \end{aligned}$$

$$C - \alpha_i - \mu_i = 0$$

带入原式：

$$-\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_{i=1}^n \alpha_i \quad \text{仍然求对偶问题}$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

$$C - \alpha_i - \mu_i = 0$$

$$\alpha_i \geq 0$$

$$\mu_i \geq 0$$

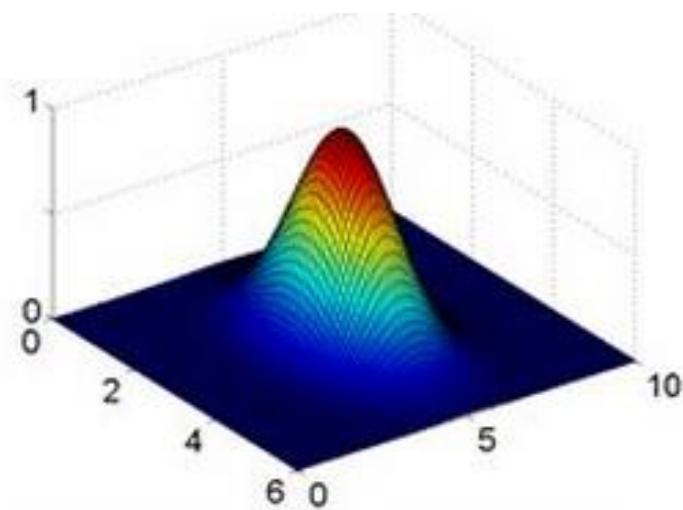
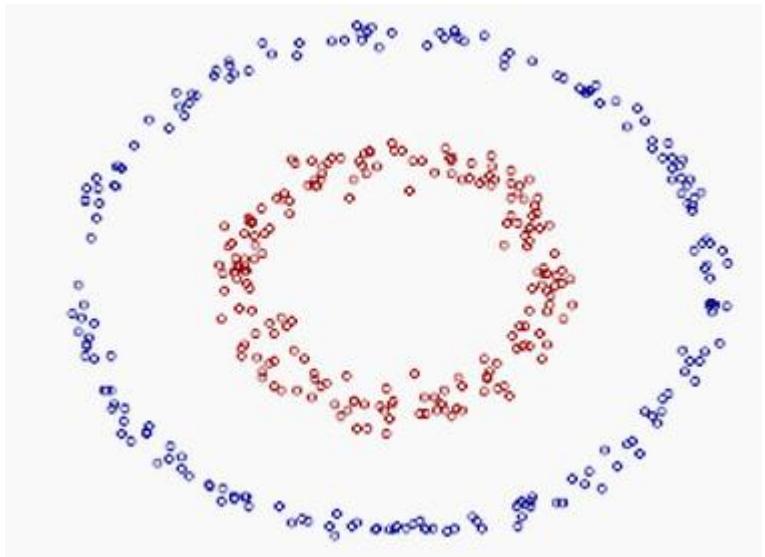
$$0 \leq \alpha_i \leq C$$



$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^n \alpha_i$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C$$



还是先从一个小例子来阐述问题。假设我们有俩个数据， $x = (x_1, x_2, x_3)$; $y = (y_1, y_2, y_3)$ ，此时在3D空间已经不能对其进行线性划分了，那么我们通过一个函数将数据映射到更高维的空间，比如9维的话，那么 $f(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$ ，由于需要计算内积，所以在新的数据在9维空间，需要计算 $\langle f(x), f(y) \rangle$ 的内积，需要花费 $O(n^2)$ 。

在具体点，令 $x = (1, 2, 3)$; $y = (4, 5, 6)$ ，那么 $f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)$, $f(y) = (16, 20, 24, 20, 25, 36, 24, 30, 36)$,

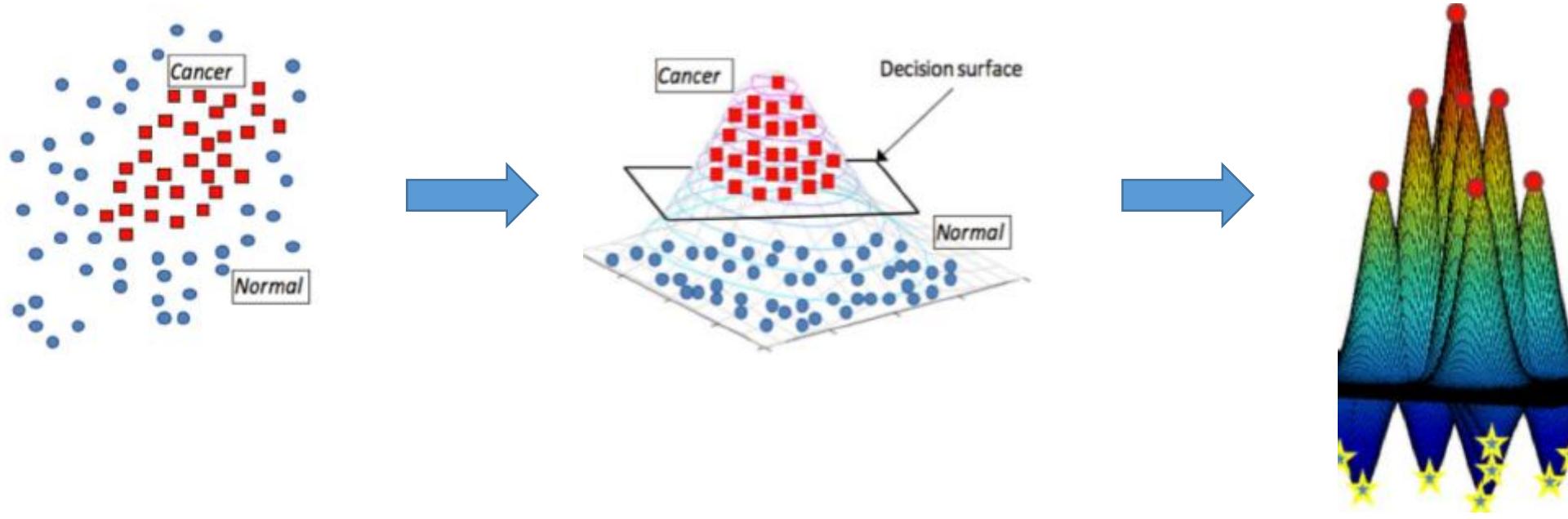
$$\text{此时 } \langle f(x), f(y) \rangle = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324 = 1024$$

似乎还能计算，但是如果将维数扩大到一个非常大数时候，计算起来可就不是一丁点问题了。

但是发现， $K(x, y) = (\langle x, y \rangle)^2$

$$K(x, y) = (4 + 10 + 18)^2 = 32^2 = 1024$$

两者相等， $K(x, y) = (\langle x, y \rangle)^2 = \langle f(x), f(y) \rangle$ ，但是 $K(x, y)$ 计算起来却比 $\langle f(x), f(y) \rangle$ 简单的多，也就是说只要用 $K(x, y)$ 来计算，效果和 $\langle f(x), f(y) \rangle$ 是一样的，但是计算效率却大幅度提高了，如： $K(x, y)$ 是 $O(n)$ ，而 $\langle f(x), f(y) \rangle$ 是 $O(n^2)$ 。所以使用核函数的好处就是，可以在一个低维空间去完成高维度（或者无限维度）样本内积的计算，比如 $K(x, y) = (4 + 10 + 18)^2$ 的3D空间对比 $\langle f(x), f(y) \rangle = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324$ 的9D空间。

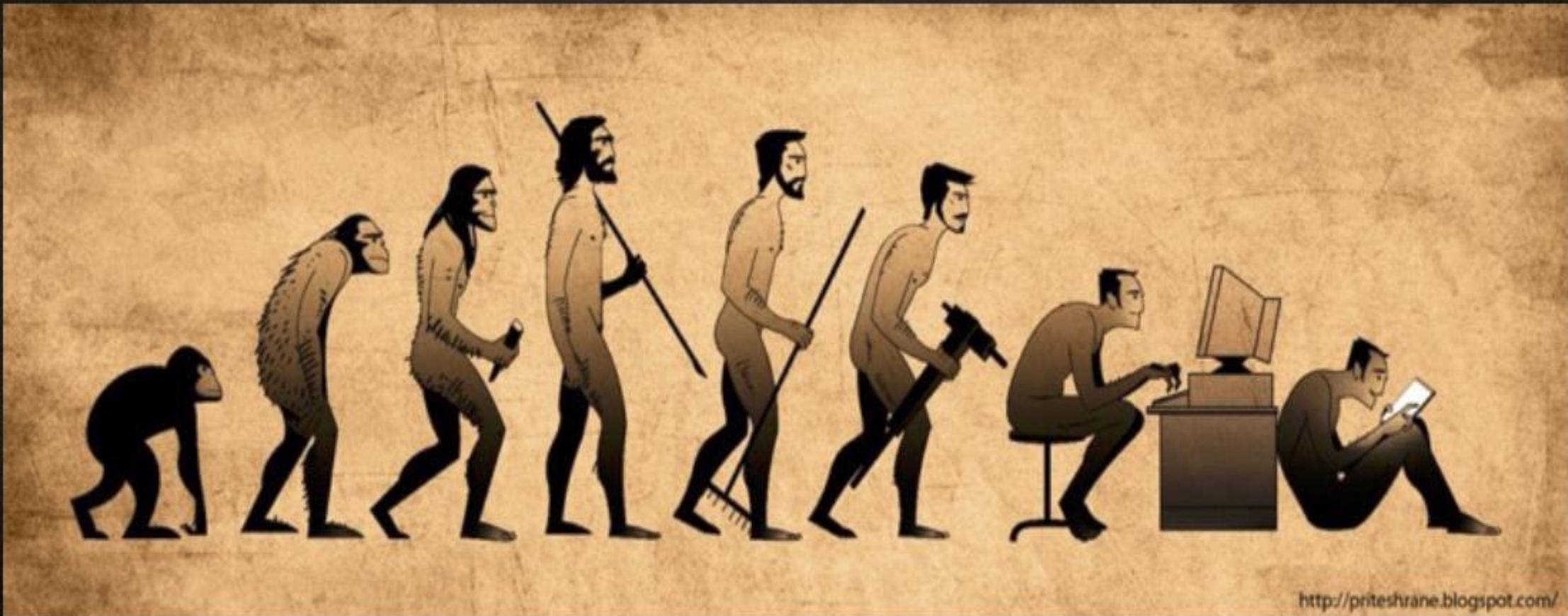


高斯核函数：

$$K(X, Y) = \exp \left\{ -\frac{\|X - Y\|^2}{2\sigma^2} \right\}$$

什么是
人工智能？





<http://priteshrane.blogspot.com/>

学习的能力，是智能的本质！

万物互联

万物智能

大数据时代





www.image-net.org

22K categories and **14M** images

- Animals
 - Bird
 - Fish
 - Mammal
 - Invertebrate
- Plants
 - Tree
 - Flower
 - Food
 - Materials
- Structures
 - Artifact
 - Tools
 - Appliances
 - Structures
- Person
- Scenes
 - Indoor
 - Geological Formation
- Sport Activities

为什么围棋选手不信阿法狗呢？

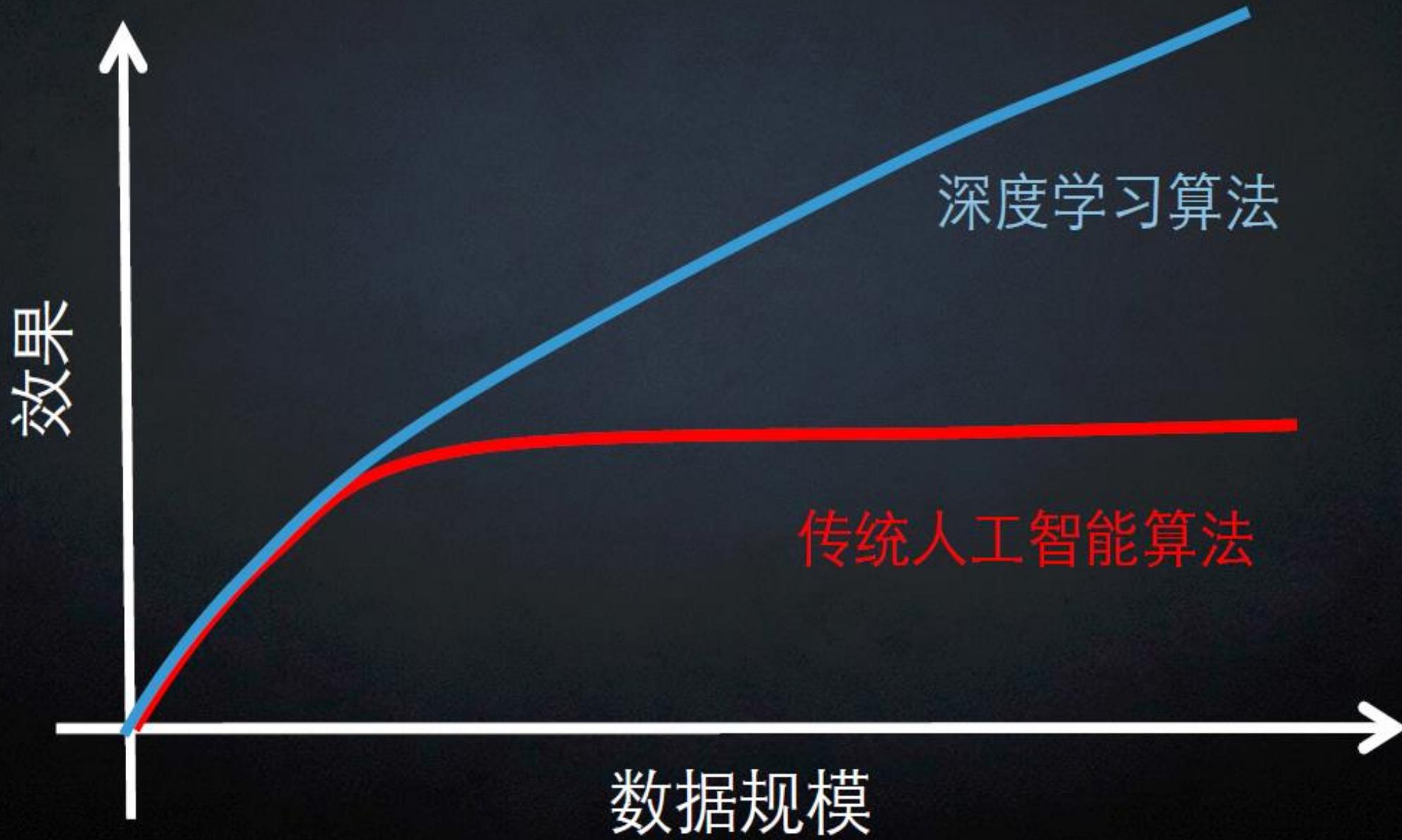
结果呢？

4:1 李世石惨败

人工智能的时代已经来临

为什么人工智能技术这么厉害？





述说图片的故事



A yellow bus driving down a road with green trees and green grass in the background.



Living room with white couch and blue carpeting. The room in the apartment gets some afternoon sun.

这些字幕是深度学习程序写的



无人驾驶汽车：

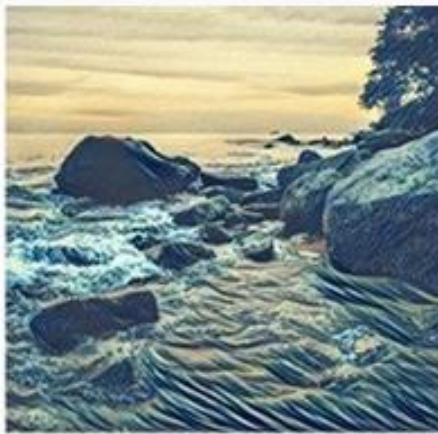
1. 物体检测

2. 行人检测

3. 速度识别

4. 速度识别

。 。 。



01. Take a picture

02. Pick a style

03. Enjoy the result



黑科技：Image Transfer

Content + Style = Interesting thing

图像分类：计算机视觉核心任务



(假设我们有一系列的标签：狗，猫，汽车，
飞机。。。)



猫

图像在计算机中长什么样呢

一张图片被表示成三维数组的形式，每个像素的值从0到255

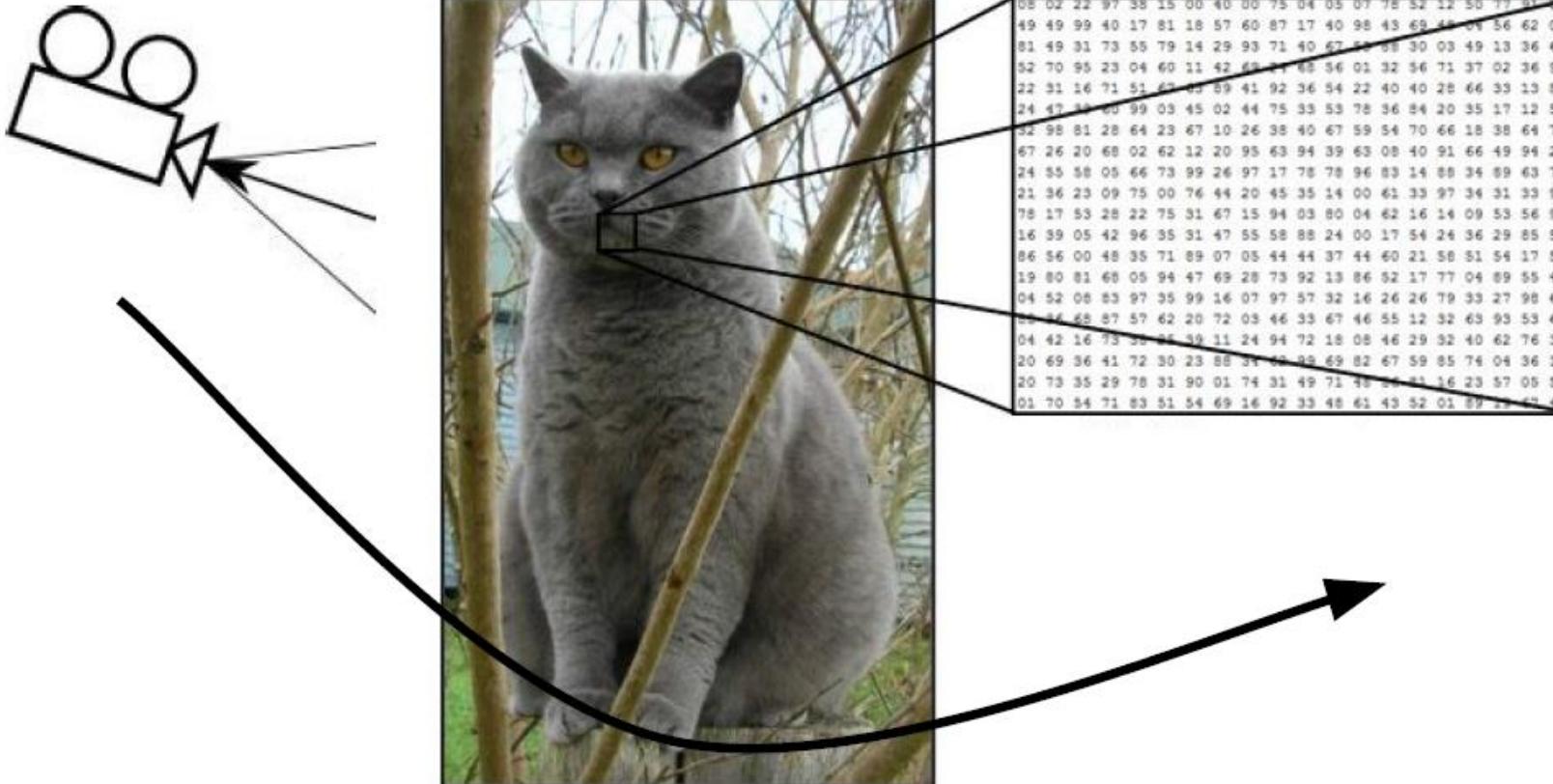
例如：300*100*3



08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 56	49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 06 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 08 30 03 49 13 36 65	52 70 95 23 04 60 11 42 62 21 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 62 03 59 41 92 36 54 22 40 40 28 66 33 13 80	24 47 33 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70	67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 96 21
24 55 58 05 66 73 99 26 97 17 78 78 96 03 14 88 34 89 63 72	21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92	16 39 05 42 96 35 31 47 55 58 88 24 08 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58	19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 63 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66	23 86 48 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 30 95 39 11 24 94 72 18 08 46 29 32 40 62 76 36	20 69 36 41 72 30 23 88 34 68 89 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 61 51 16 23 57 05 54	01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 13 47 46

What the computer sees

挑战：照射角度



挑战：光照强度



挑战：形状改变

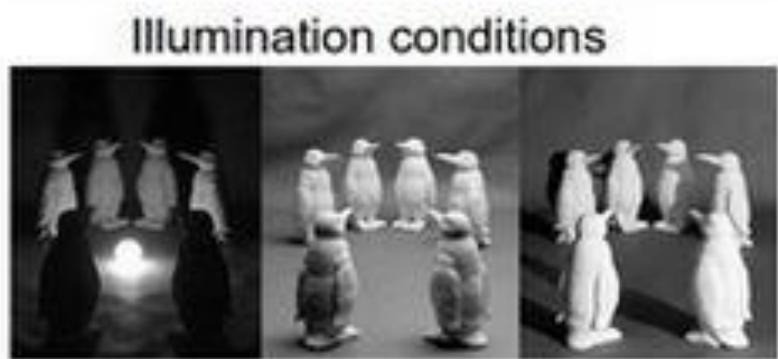


挑战：部分遮蔽



挑战：背景混入





常规套路：

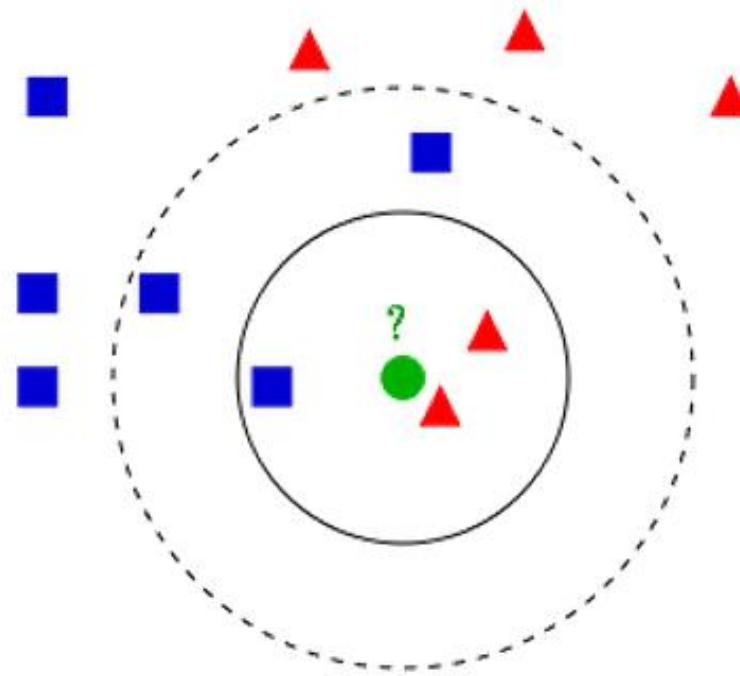
1. 收集数据并给定标签
2. 训练一个分类器
3. 测试，评估

```
def train(train_images, train_labels):  
    # build a model for images -> labels...  
    return model  
  
def predict(model, test_images):  
    # predict test_labels using the model...  
    return test_labels
```

Example training set



K-近邻



- 如果 $K=3$, 绿色圆点的最近的3个邻居是2个红色小三角形和1个蓝色小正方形, 少数从属于多数, 基于统计的方法, 判定绿色的这个待分类点属于红色的三角形一类。
- 如果 $K=5$, 绿色圆点的最近的5个邻居是2个红色三角形和3个蓝色的正方形, 还是少数从属于多数, 基于统计的方法, 判定绿色的这个待分类点属于蓝色的正方形一类。

K-近邻

对于未知类别属性数据集中的点：

- 1.计算已知类别数据集中的点与当前点的距离
- 2.按照距离依次排序
- 3.选取与当前点距离最小的K个点
- 4.确定前K个点所在类别的出现概率
- 5.返回前K个点出现频率最高的类别作为当前点预测分类。

K-近邻

概述：

KNN 算法本身简单有效，它是一种 **lazy-learning** 算法。

分类器不需要使用训练集进行训练，训练时间复杂度为0。

KNN 分类的计算复杂度和训练集中的文档数目成正比，也就是说，如果训练集中文档总数为 n ，那么 KNN 的分类时间复杂度为 $O(n)$ 。

K-近邻

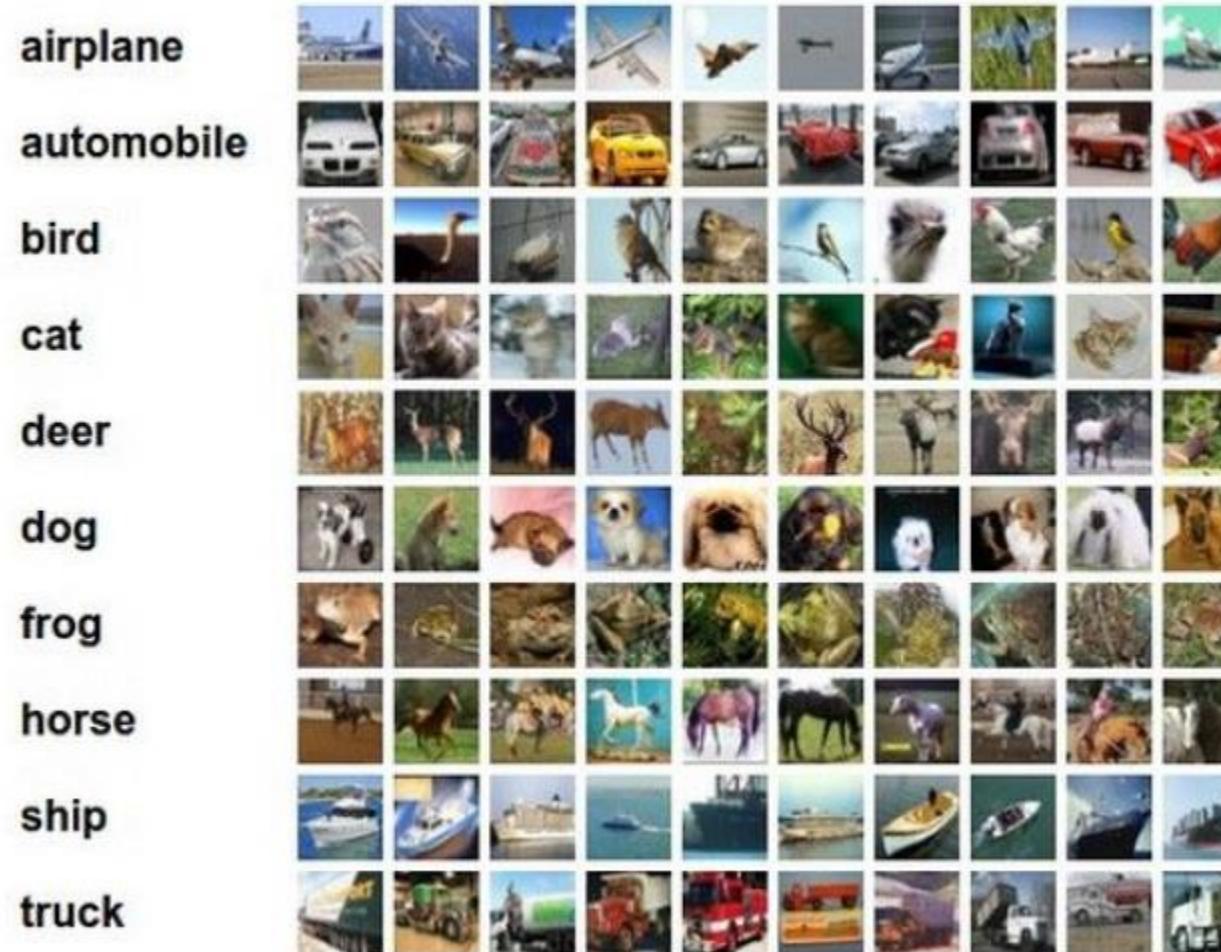
K值的选择，距离度量和分类决策规则是该算法的三个基本要素

问题：该算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数

解决：不同的样本给予不同权重项

数据库样例： CIFAR-10

10类标签
50000个训练数据
10000个测试数据
大小均为32*32



如何计算的呢：

L1 distance:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

-

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

=

add → 456

测试结果：



最近邻代码：

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

记录所有训练数据

对于每一个测试数据找出与其L1距离最小的样本的标签，作为它的标签

超参数：

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

问题：

- 1.对于距离如何设定？
- 2.对于K近邻的K该如何选择？
- 3.如果说有的话，其它的超参数该怎么设定呢？

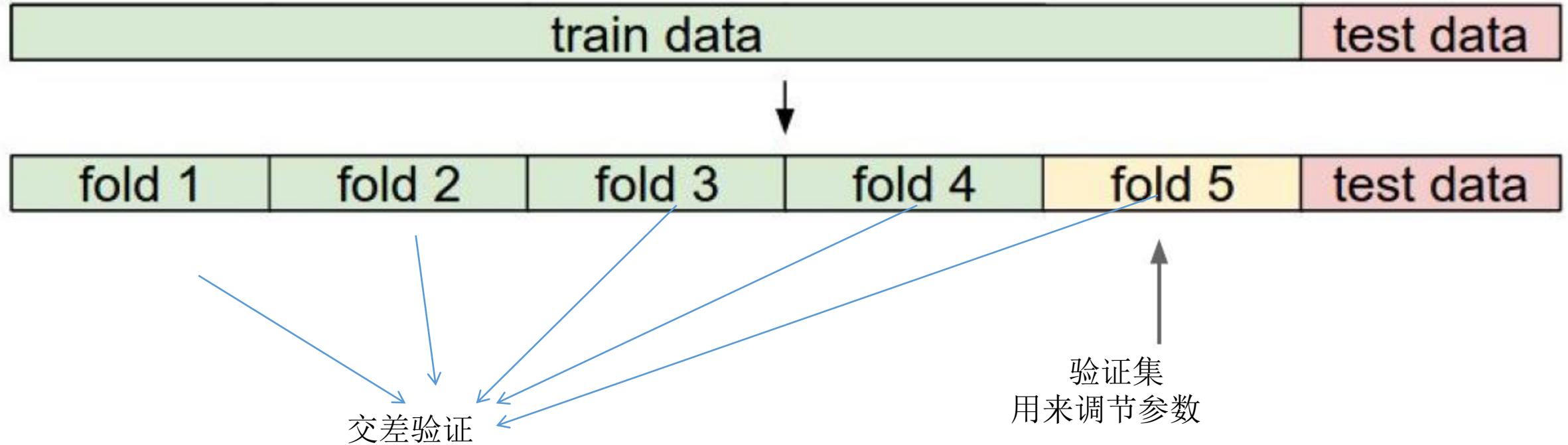
找到最好的参数：



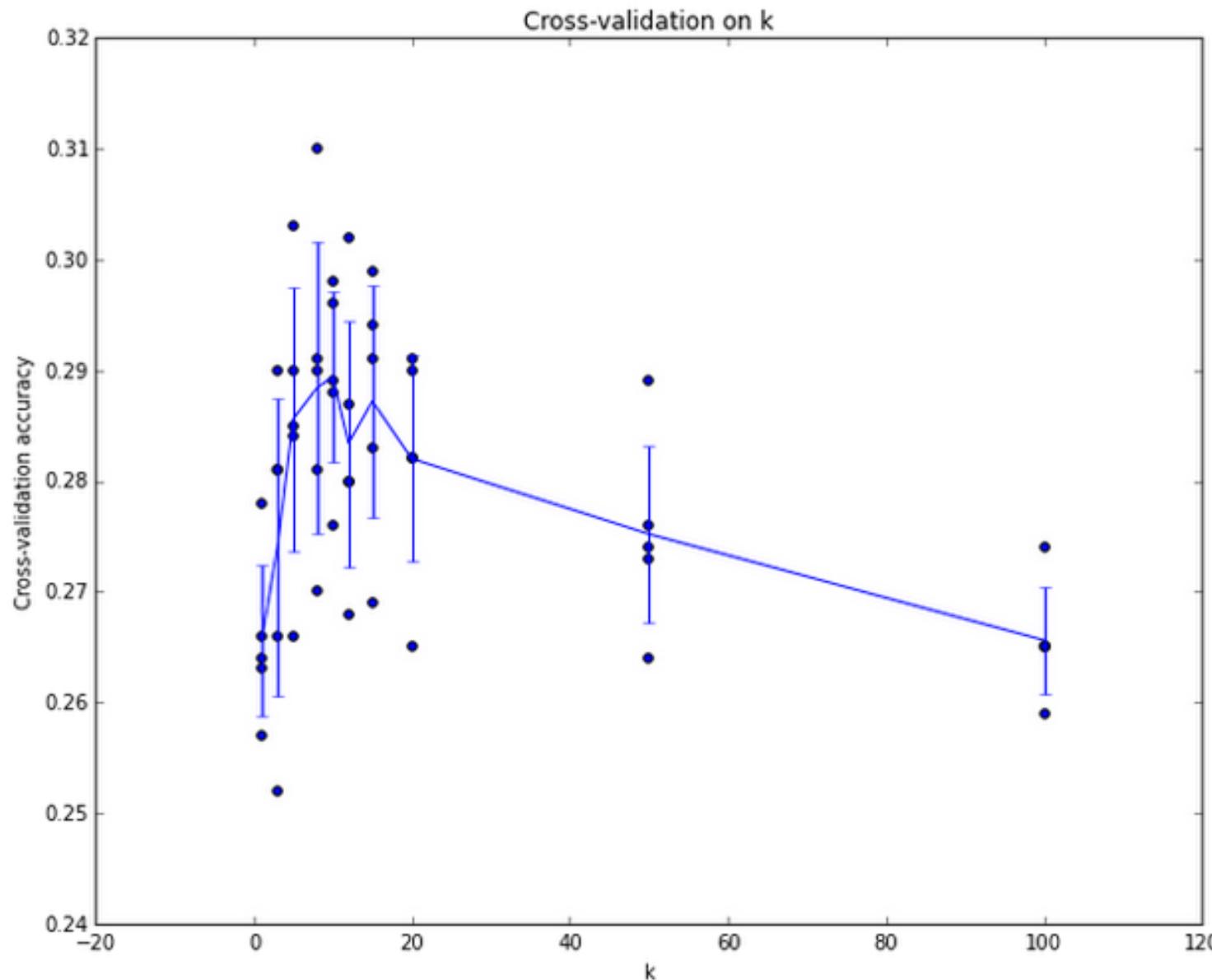
多次用测试数据试验，找到做好的一组参数组合？

错误的的想法，测试数据只能最终用

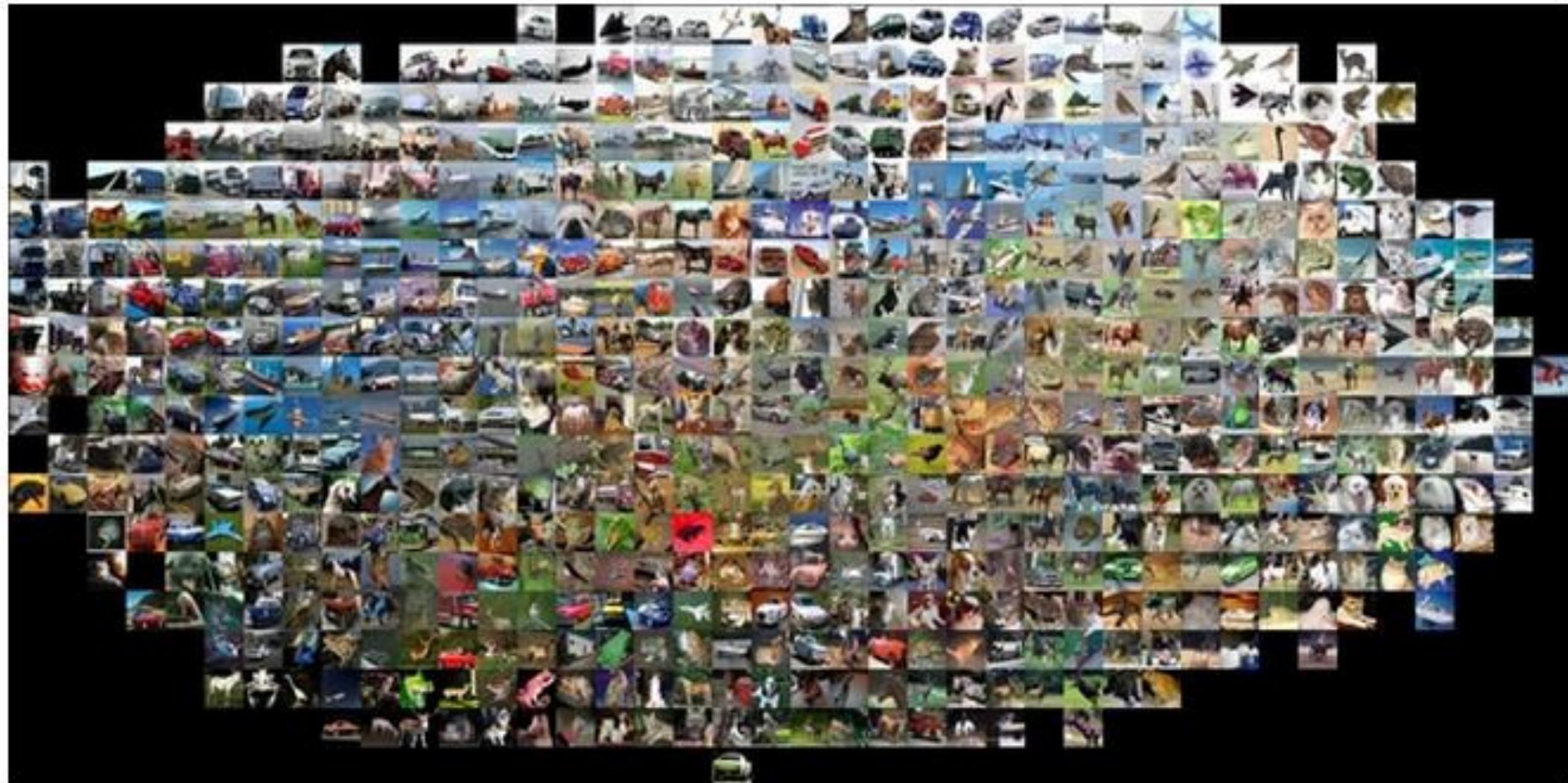
测试结果：



K-近邻



背景主导



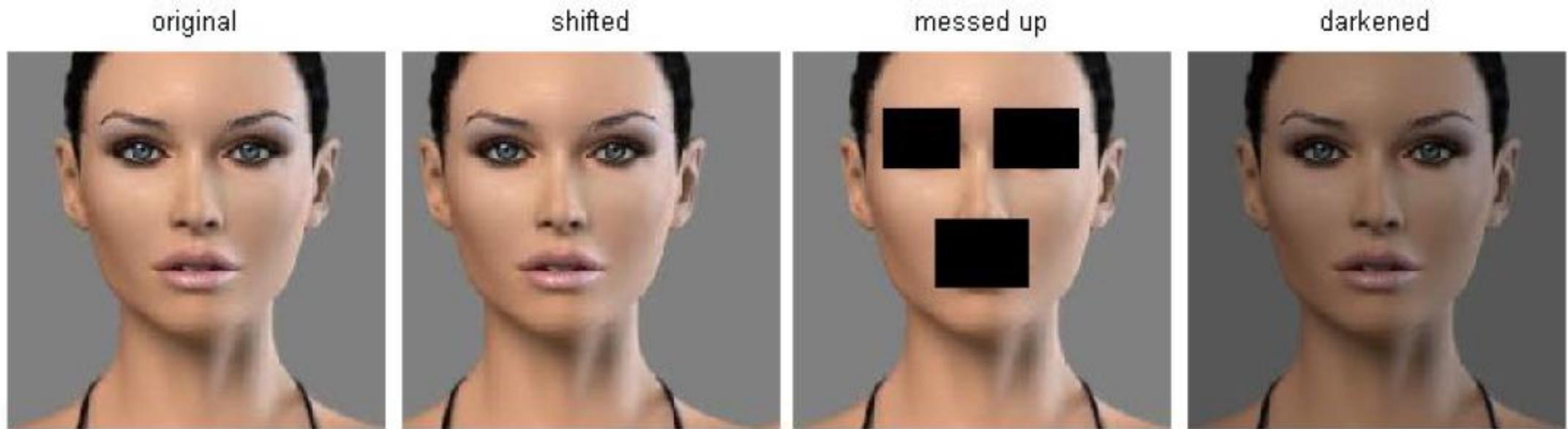
K-近邻

- 1.选取超参数的正确方法是：将原始训练集分为训练集和验证集，我们在验证集上尝试不同的超参数，最后保留表现最好那个
- 2.如果训练数据量不够，使用交叉验证方法，它能帮助我们在选取最优超参数的时候减少噪音。
- 3.一旦找到最优的超参数，就让算法以该参数在测试集跑且只跑一次，并根据测试结果评价算法。
- 4.最近邻分类器能够在CIFAR-10上得到将近40%的准确率。该算法简单易实现，但需要存储所有训练数据，并且在测试的时候过于耗费计算能力
- 5.最后，我们知道了仅仅使用L1和L2范数来进行像素比较是不够的，图像更多的是按照背景和颜色被分类，而不是语义主体分身。

K-近邻

1. 预处理你的数据：对你数据中的特征进行归一化（normalize），让其具有零平均值（zero mean）和单位方差（unit variance）。
2. 如果数据是高维数据，考虑使用降维方法，比如PCA
3. 将数据随机分入训练集和验证集。按照一般规律，70%-90% 数据作为训练集
4. 在验证集上调优，尝试足够多的k值，尝试L1和L2两种范数计算方式。

K近邻: never used



(不同的变换和原图具有相同的L2距离)

线性分类：



(32x32x3)

image parameters

$$f(\mathbf{x}, \mathbf{W})$$

每个类别的得分

得分函数：



[32x32x3]

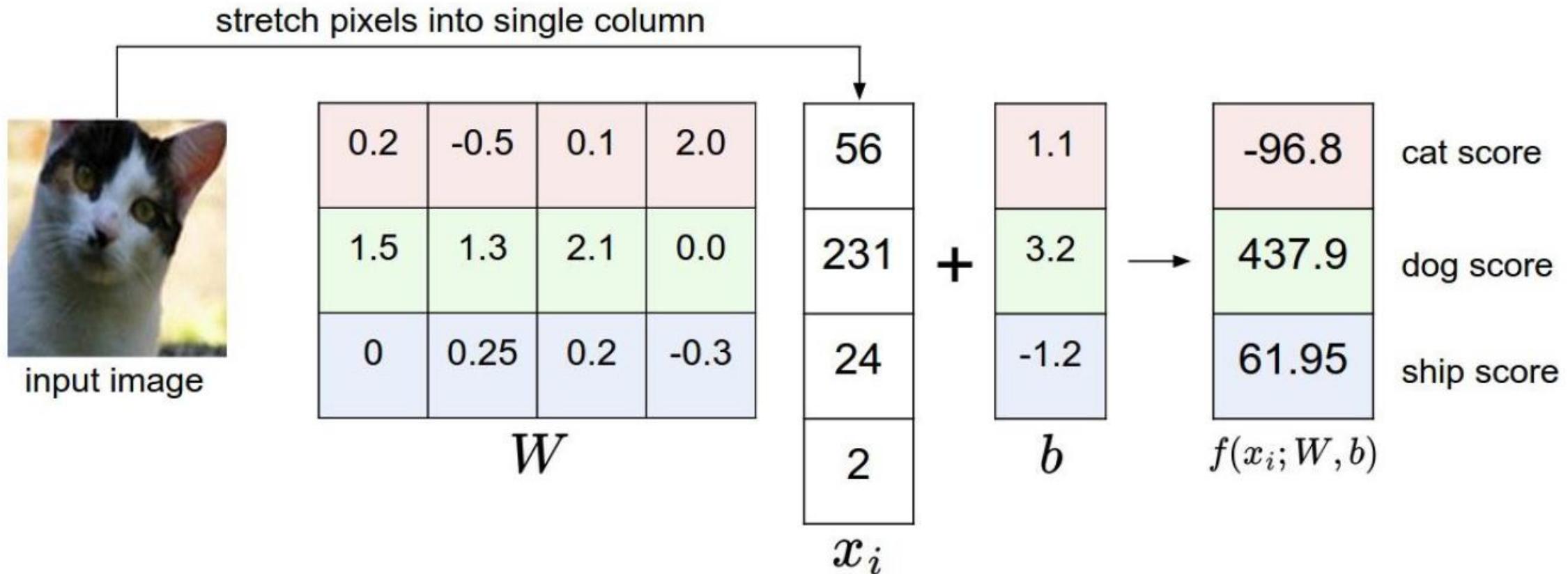
$$f(x, W) = \boxed{W} \boxed{x}$$

10x1 **10x3072** **3072x1**

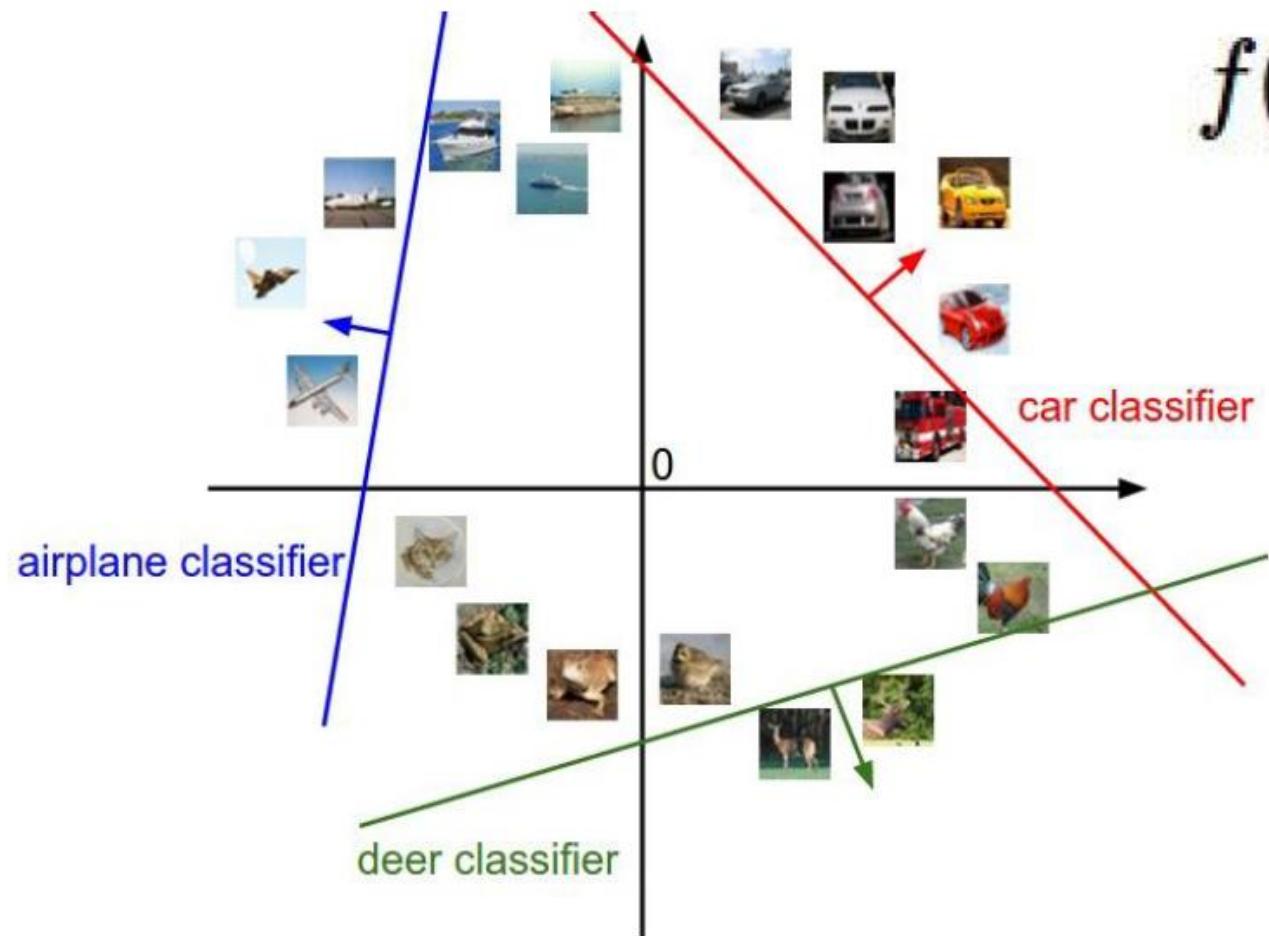
(+b) **10x1**

权重

实例



$$f(x_i, W, b) = Wx_i + b$$



损失函数：



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

损失函数：

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

$$= \max(0, 5.1 - 3.2 + 1)$$

$$+ \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$

$$= \max(0, 1.3 - 4.9 + 1)$$

$$+ \max(0, 2.0 - 4.9 + 1)$$

$$= \max(0, -2.6) + \max(0, -1.9)$$

$$= 0 + 0$$

$$= 0$$

$$= \max(0, 2.2 - (-3.1) + 1)$$

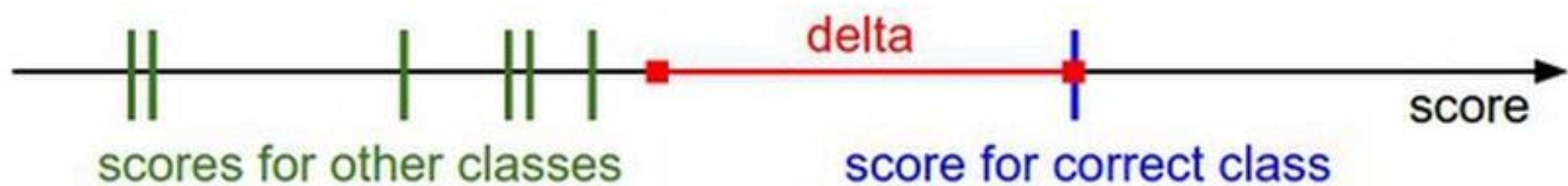
$$+ \max(0, 2.5 - (-3.1) + 1)$$

$$= \max(0, 5.3) + \max(0, 5.6)$$

$$= 5.3 + 5.6$$

$$= 10.9$$

损失函数：



损失函数：

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

正则化：

正则化惩罚项

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

L2正则化：

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

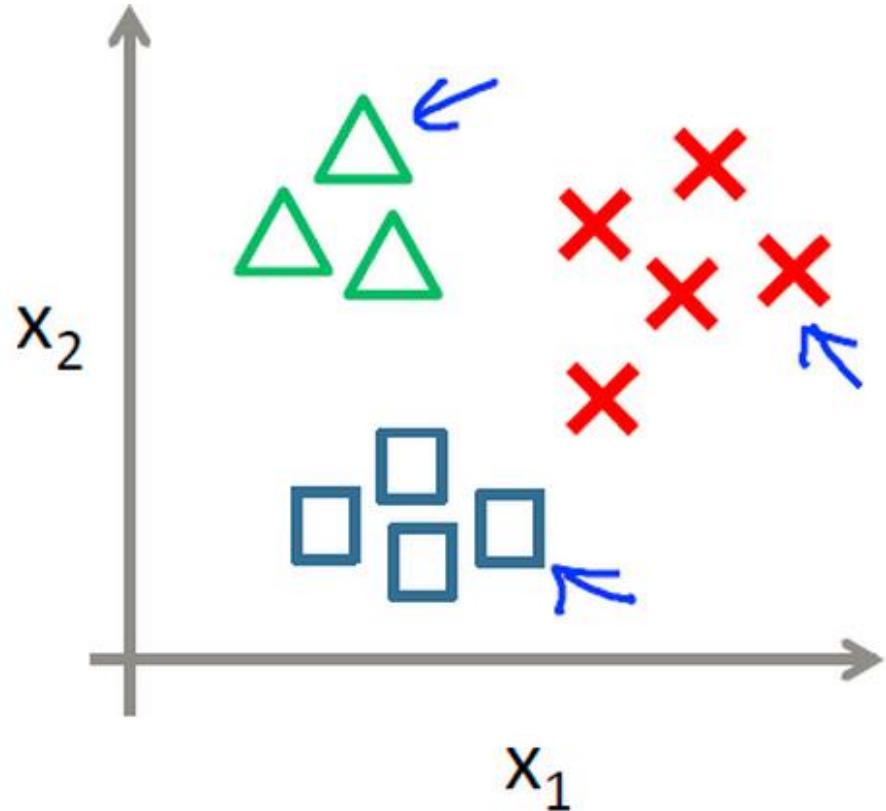
$$w_1^T x = w_2^T x = 1$$

损失函数终极版：

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

Softmax分类器：

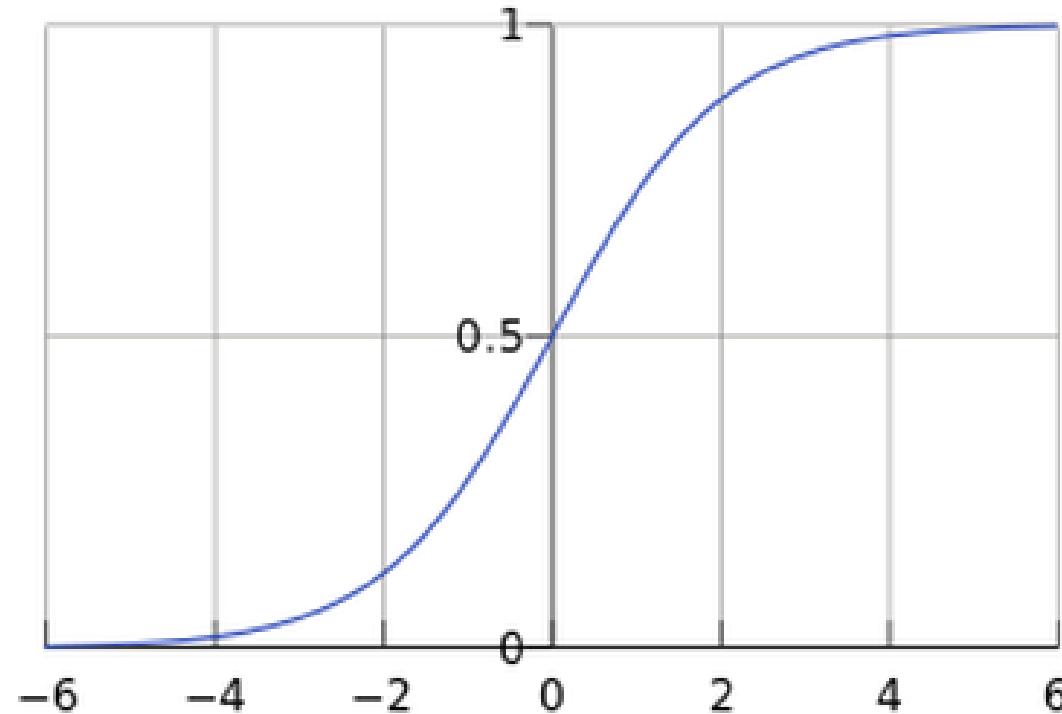
多类别分类



Softmax的输出是概率

Sigmoid函數：

$$g(z) = \frac{1}{1+e^{-z}}$$



Softmax分类器：

Softmax的输出（归一化的分类概率）

损失函数：交叉熵损失（**cross-entropy loss**）

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$
 被称作**softmax** 函数

其输入值是一个向量，向量中元素为任意实数的评分值

输出一个向量，其中每个元素值在0到1之间，且所有元素之和为1

softmax:



$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

where

$$\mathbf{s} = f(x_i; W)$$

$$L_i = -\log P(Y = y_i | X = x_i)$$

cat	3.2	$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$
car	5.1	
frog	-1.7	

Softmax实例：



$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat
car
frog

3.2
5.1
-1.7

exp

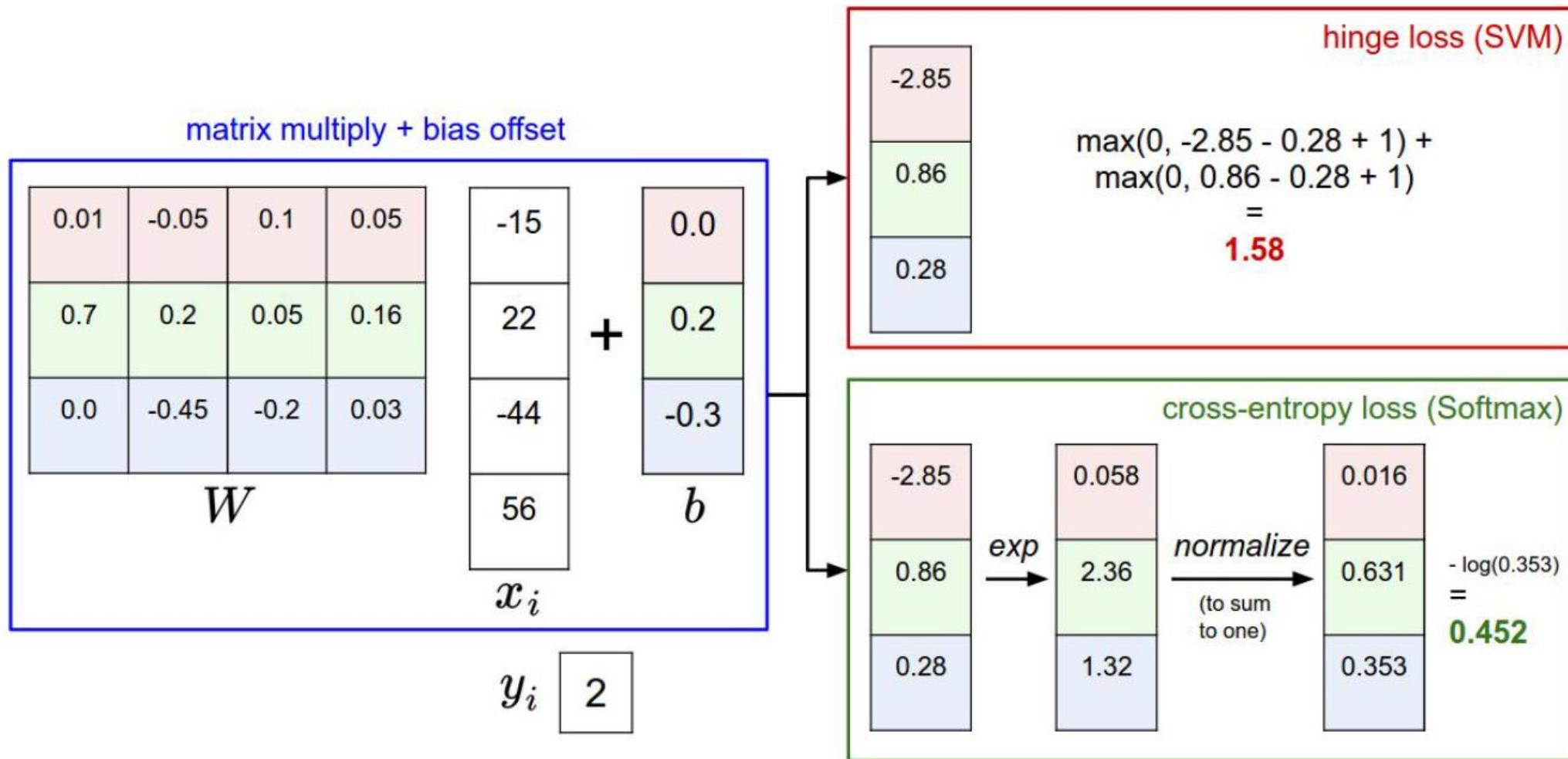
24.5
164.0
0.18

normalize

0.13
0.87
0.00

$$\begin{aligned} L_i &= -\log(0.13) \\ &= 0.89 \end{aligned}$$

损失函数对比：



最优化：

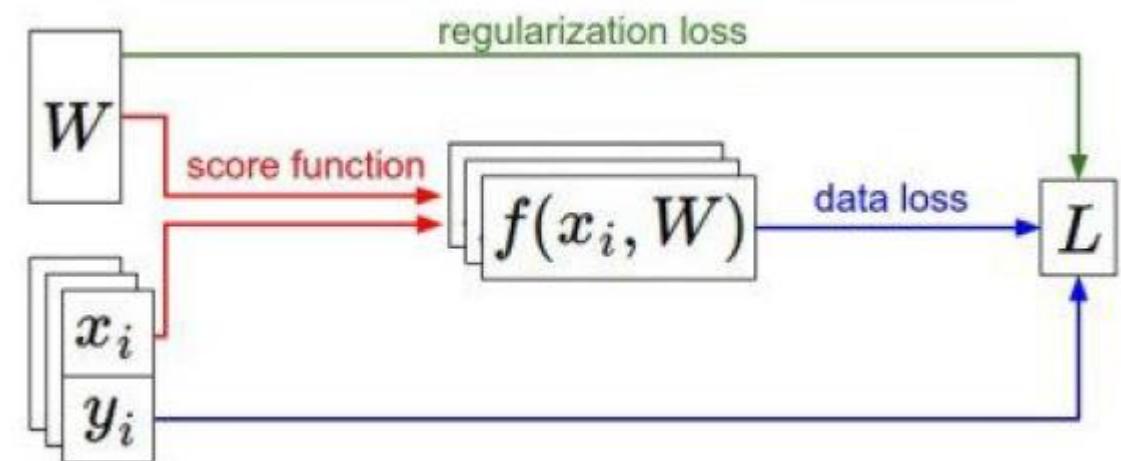
$$s = f(x; W) \stackrel{\text{e.g.}}{=} Wx$$

Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$
$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \quad \text{Full loss}$$



粗暴的想法：

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
    W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
    loss = L(X_train, Y_train, W) # get the loss over the entire training set
    if loss < bestloss: # keep track of the best solution
        bestloss = loss
        bestW = W
    print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

粗暴想法的结果：

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

找到山坡最低点：



找到山坡最低点：

```
W = np.random.randn(10, 3073) * 0.001 # 生成随机初始W
bestloss = float("inf")
for i in xrange(1000):
    step_size = 0.0001
    Wtry = W + np.random.randn(10, 3073) * step_size
    loss = L(Xtr_cols, Ytr, Wtry)
    if loss < bestloss:
        W = Wtry
        bestloss = loss
print 'iter %d loss is %f' % (i, bestloss)
```

跟隨梯度：

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

```
def eval_numerical_gradient(f, x):
    """
    a naive implementation of numerical gradient of f at x
    - f should be a function that takes a single argument
    - x is the point (numpy array) to evaluate the gradient at
    """

    fx = f(x) # evaluate function value at original point
    grad = np.zeros(x.shape)
    h = 0.00001

    # iterate over all indexes in x
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:

        # evaluate function at x+h
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h # increment by h
        fjh = f(x) # evaluate f(x + h)
        x[ix] = old_value # restore to previous value (very important!)

        # compute the partial derivative
        grad[ix] = (fjh - fx) / h # the slope
        it.iternext() # step to next dimension

    return grad
```

梯度下降：

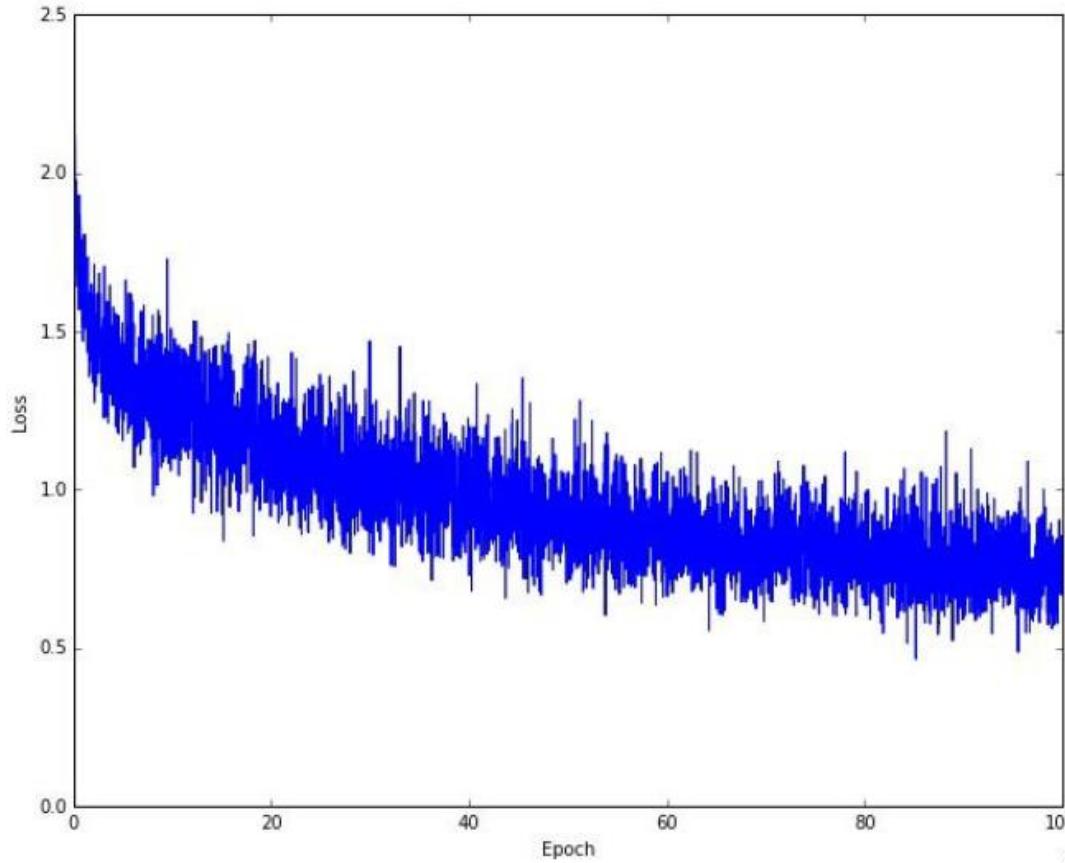
```
while True:  
    weights_grad = evaluate_gradient(loss_fun, data, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

梯度下降：

```
while True:  
    data_batch = sample_training_data(data, 256) # sample 256 examples  
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

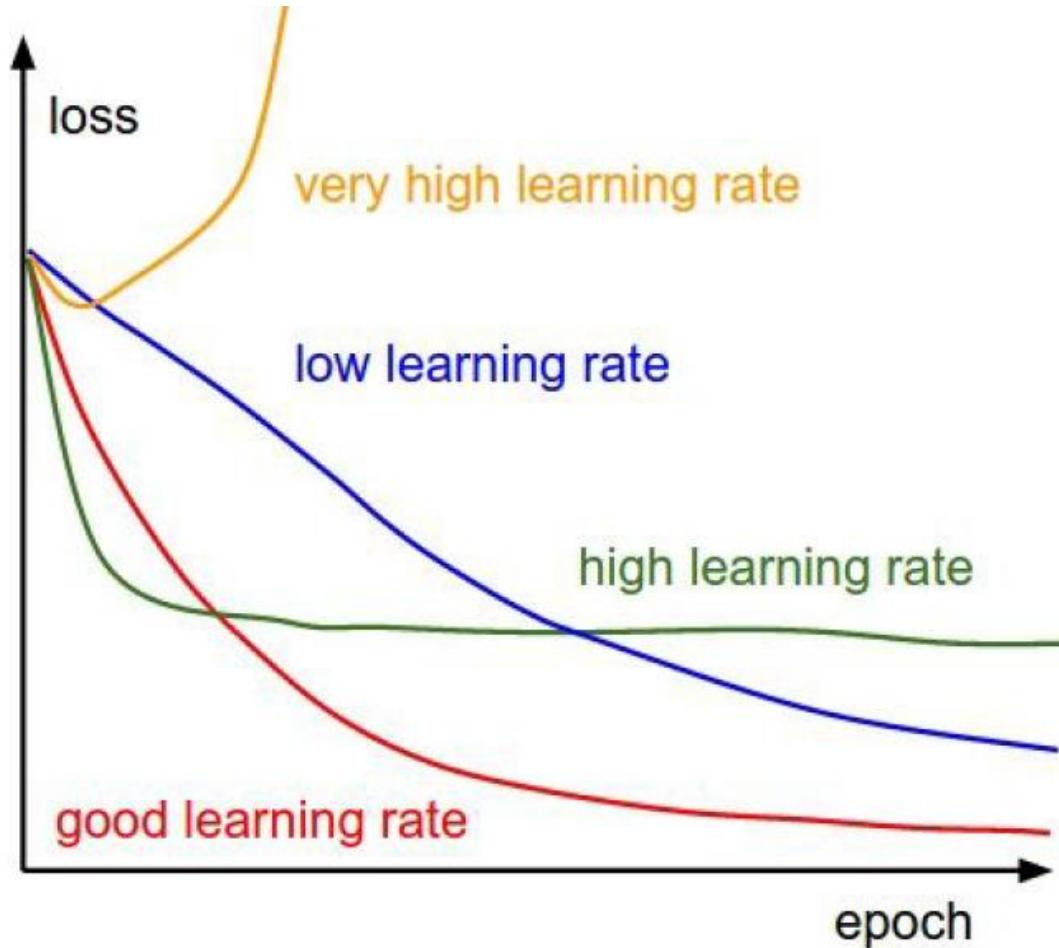
Bachsize通常是2的整数倍（32, 64, 128）

梯度下降：



训练网络时的LOSS值可视化结果

学习率：

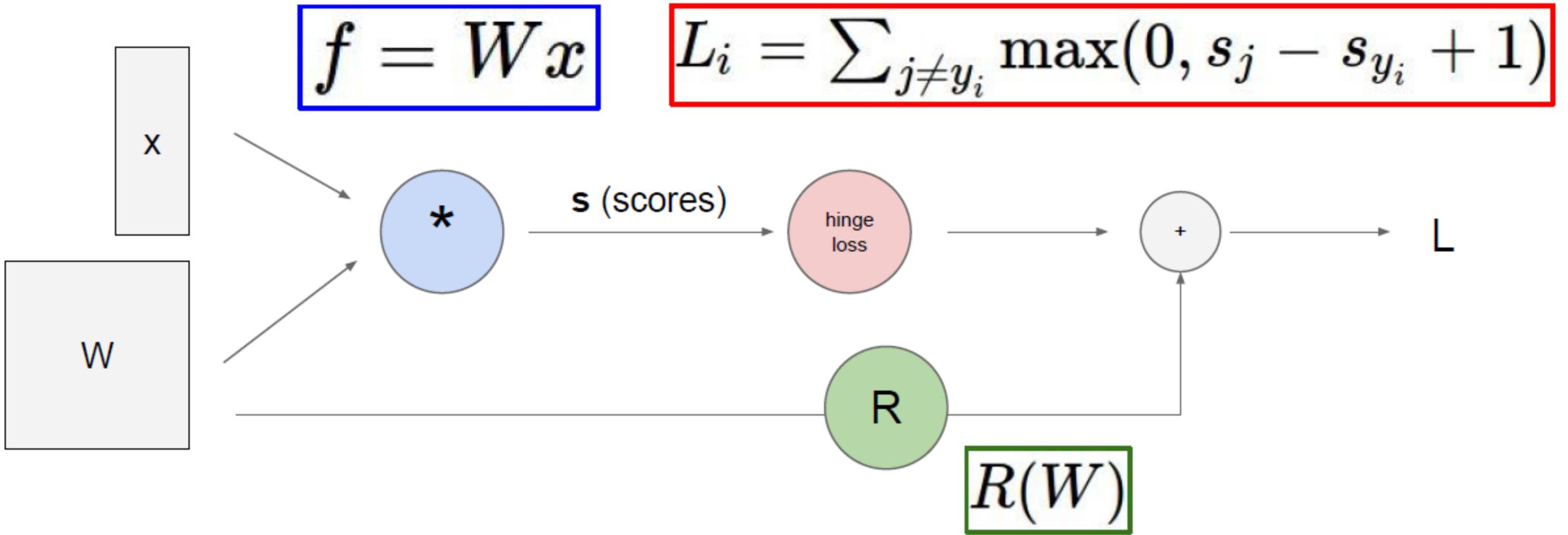


训练网络时的LOSS值可视化结果

学习率：

```
while True:  
    data_batch = sample_training_data(data, 256) # sample 256 examples  
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

反向传播：



反向传播：

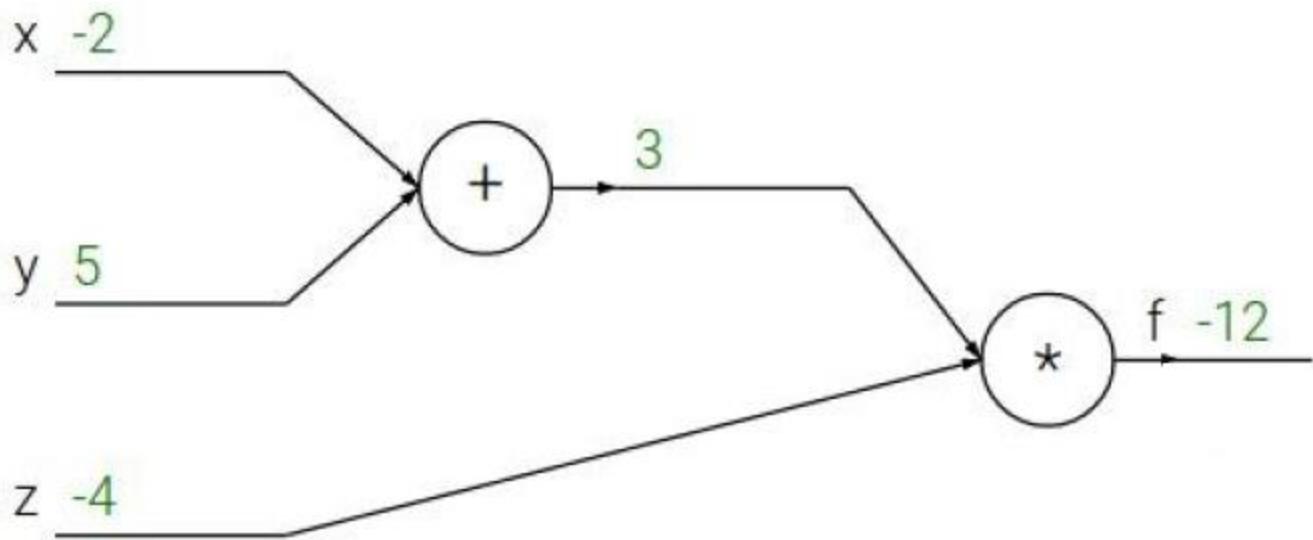
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

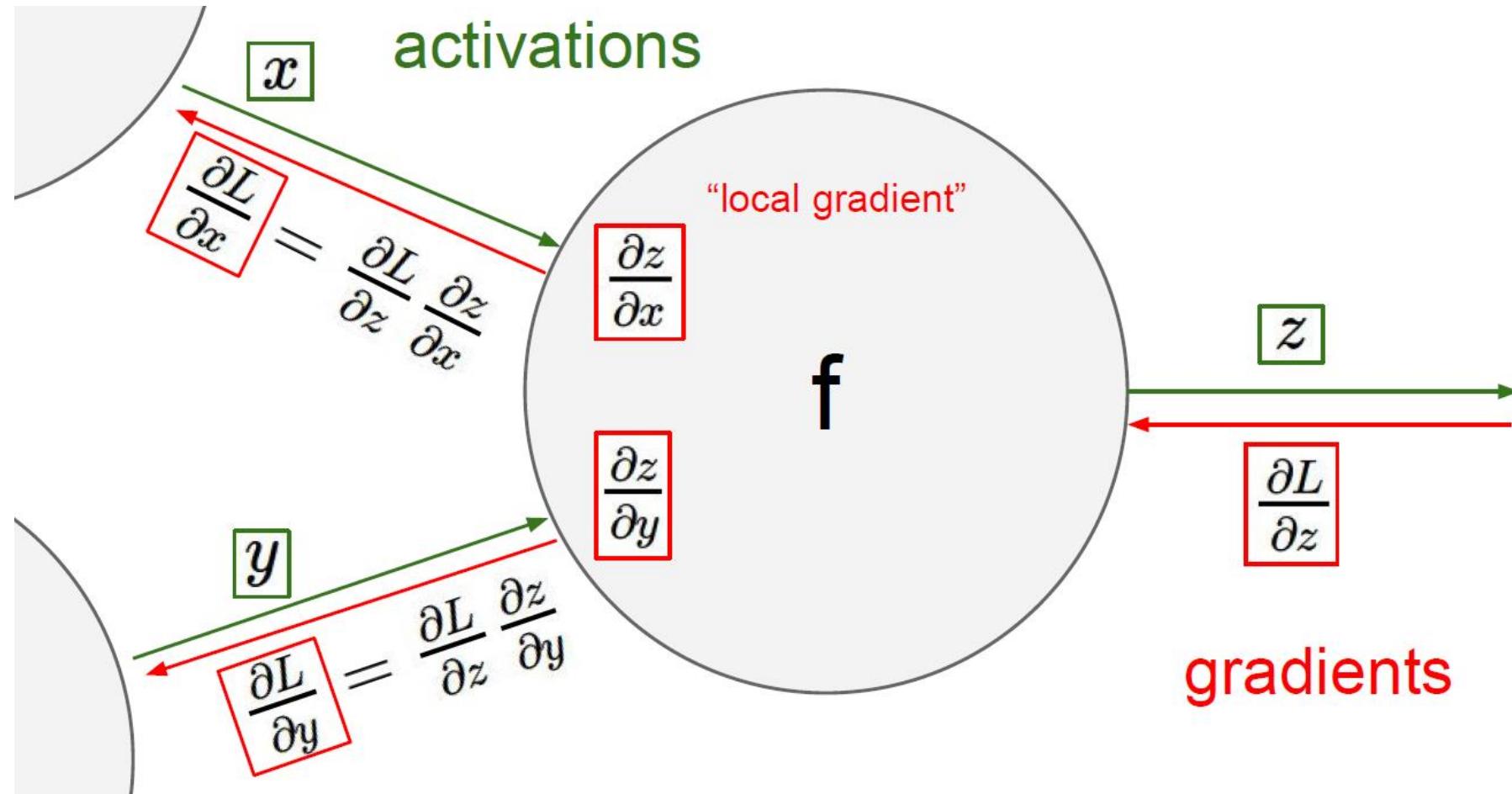
$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

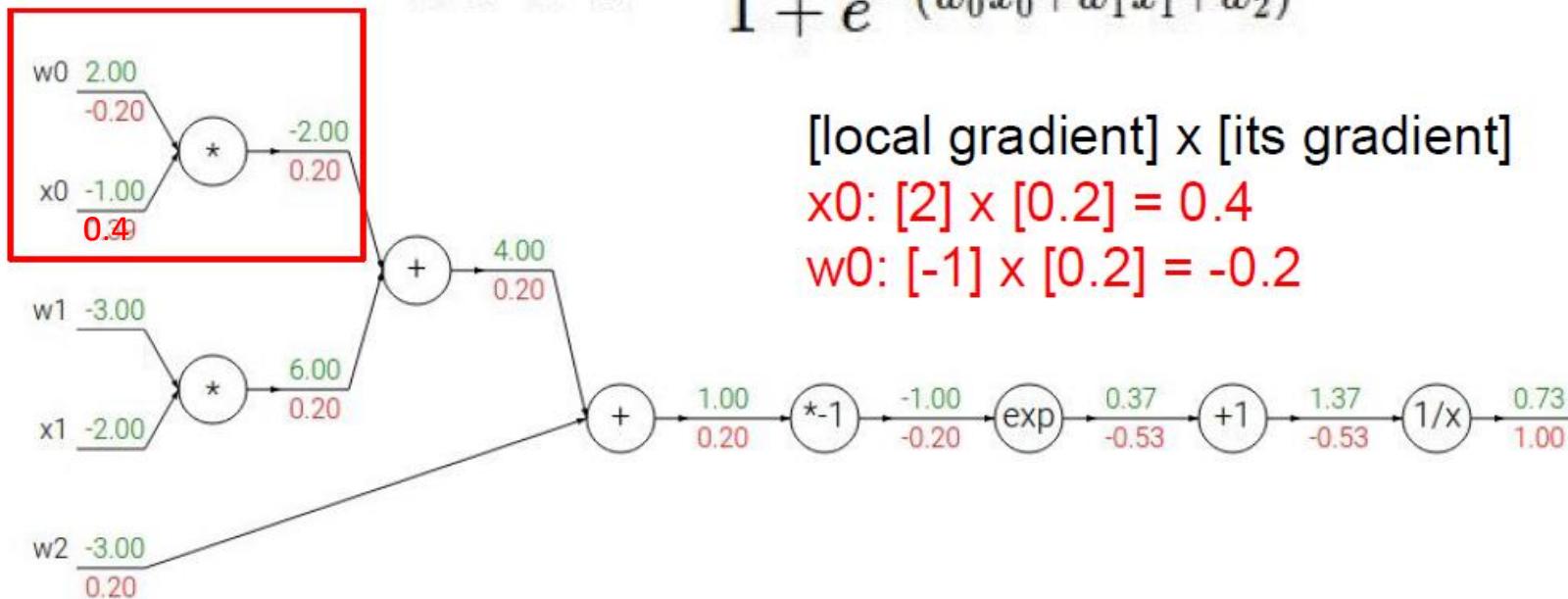


反向传播：



反向传播：

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

\rightarrow

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

\rightarrow

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

\rightarrow

$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

\rightarrow

$$\frac{df}{dx} = 1$$

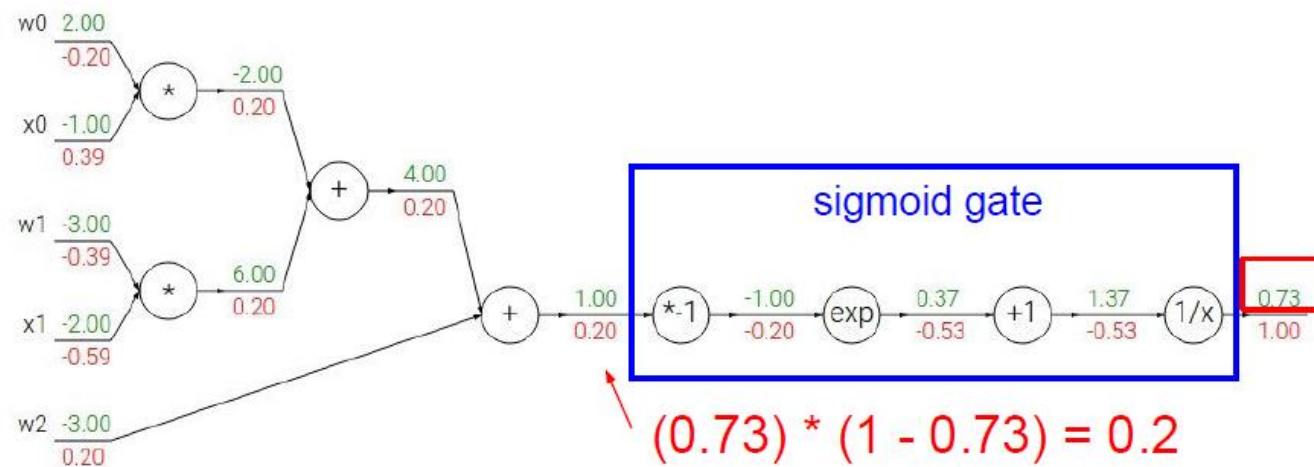
反向传播：

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x)$$

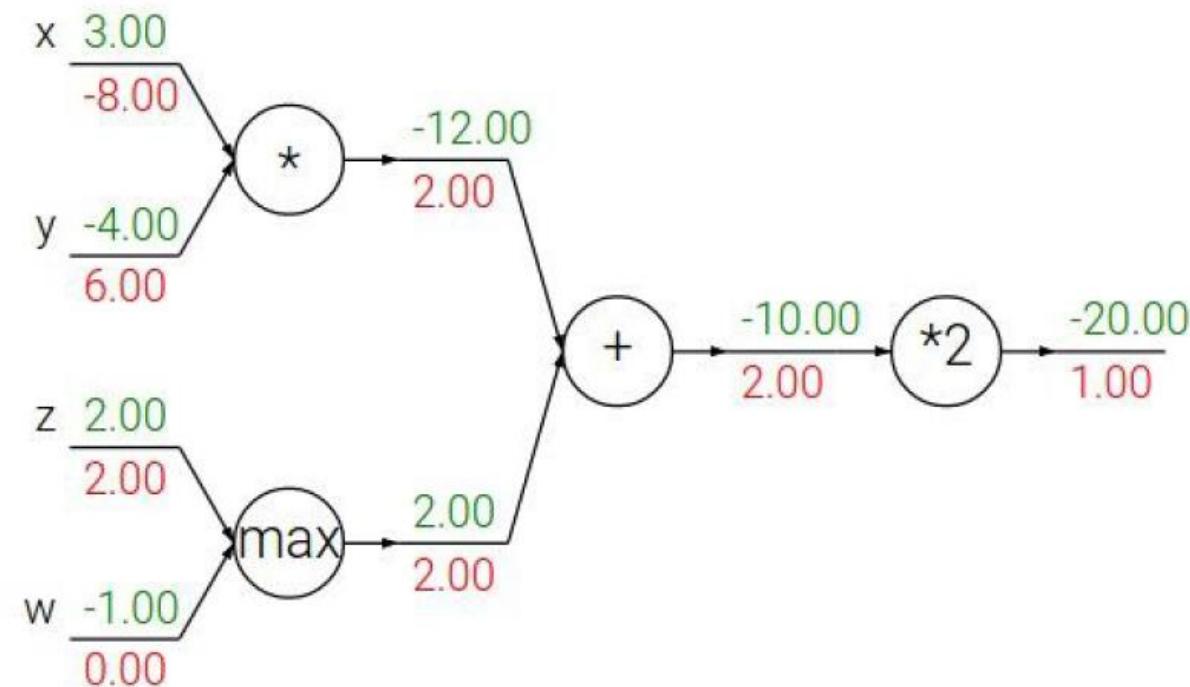


反向传播：

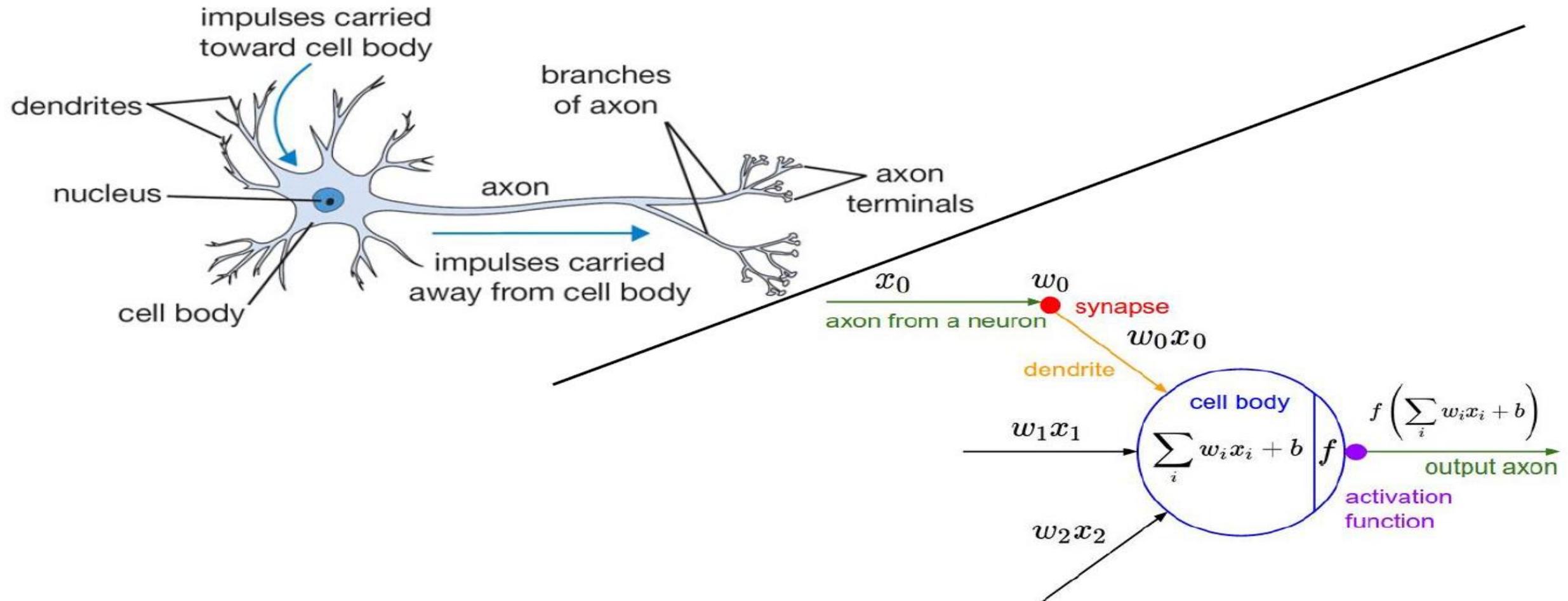
加法门单元：均等分配

MAX门单元：给最大的

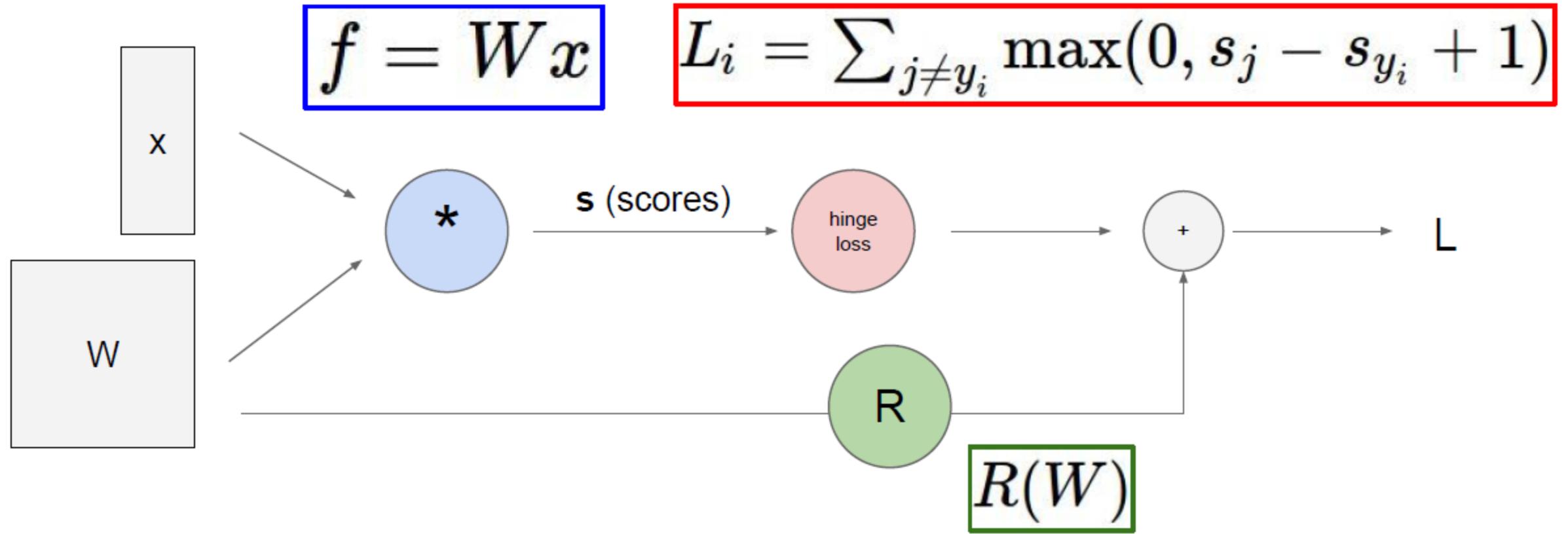
乘法门单元：互换的感觉



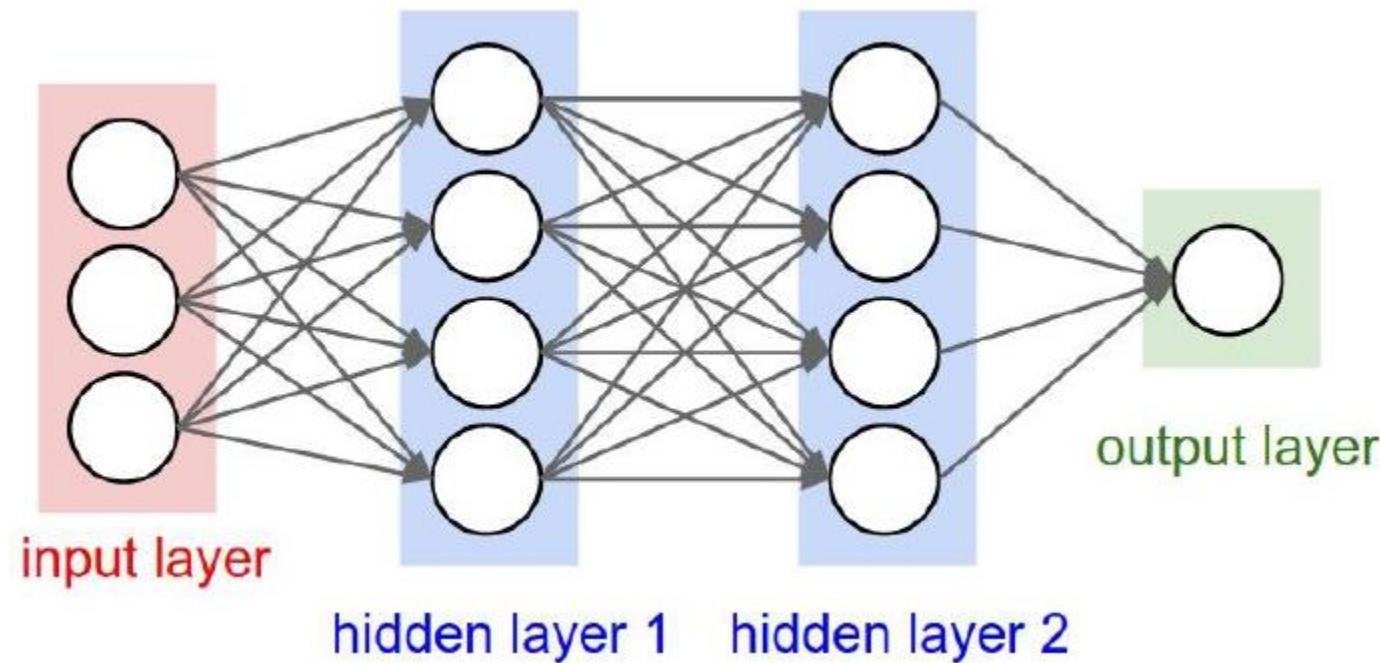
神经网络：



神经网络：



神经网络：

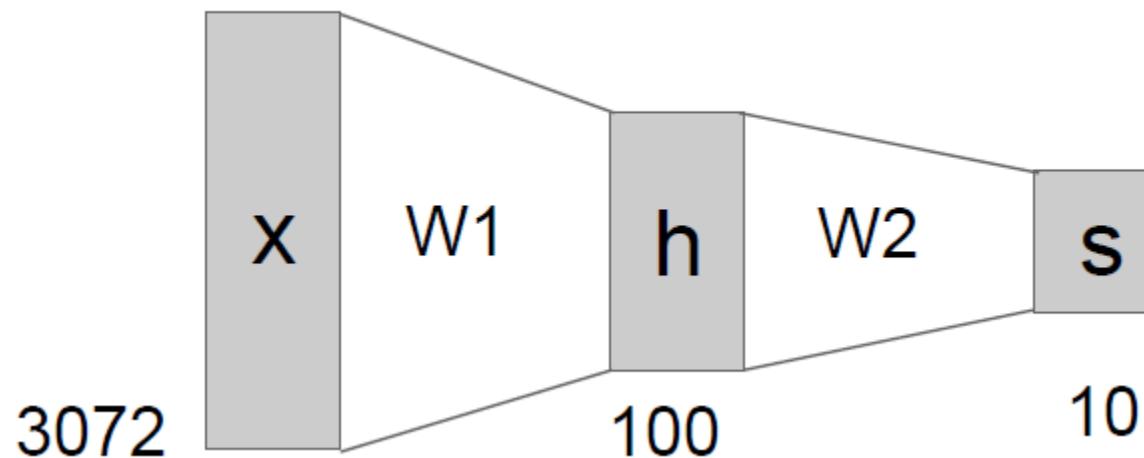


神经网络：

线性方程:

$$f = Wx$$

非线性方程: $f = W_2 \max(0, W_1 x)$



神经网络：

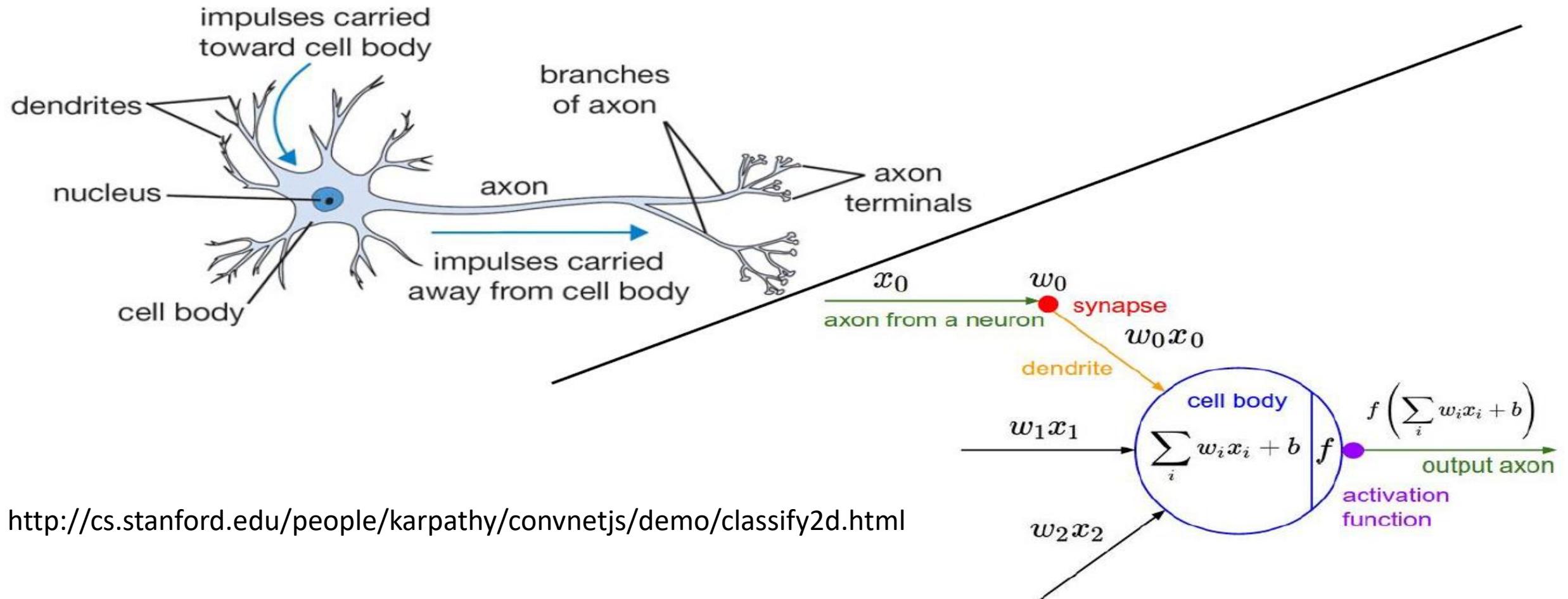
单层的神经网络：

$$f = W_2 \max(0, W_1 x)$$

双层的神经网络：

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

神经网络：

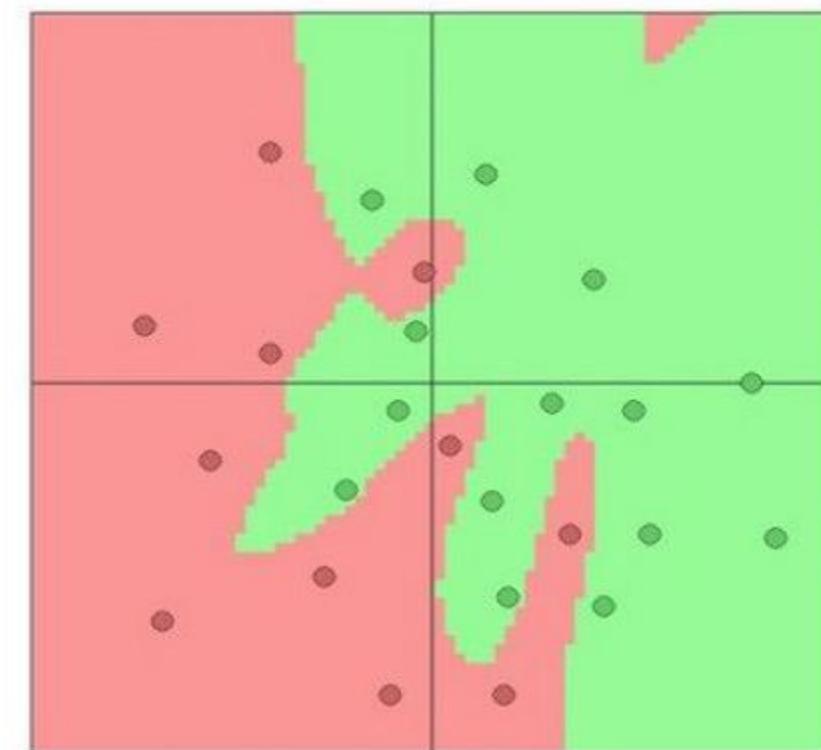


<http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

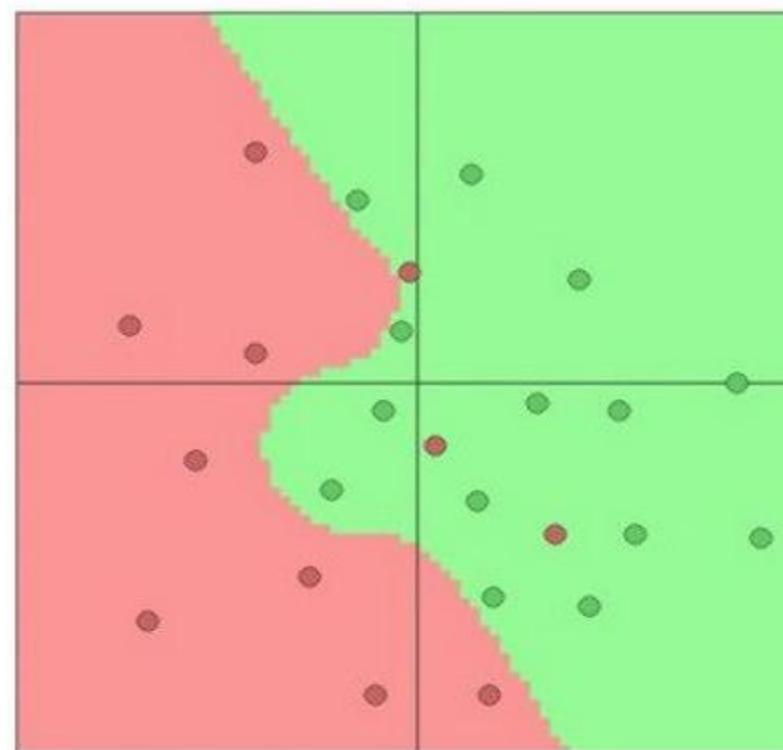
神经网络：

正则化项在神经网络中的重要作用

$$\lambda = 0.001$$



$$\lambda = 0.01$$

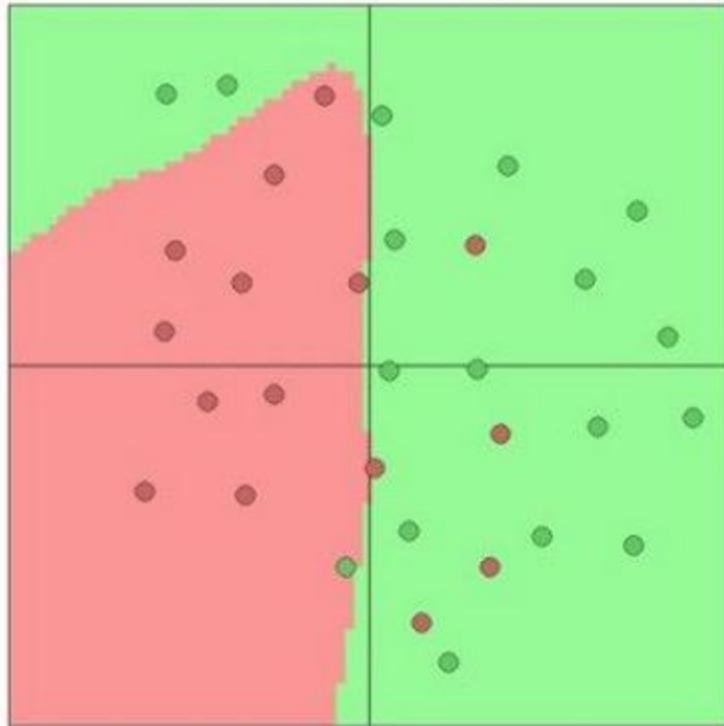


$$\lambda = 0.1$$

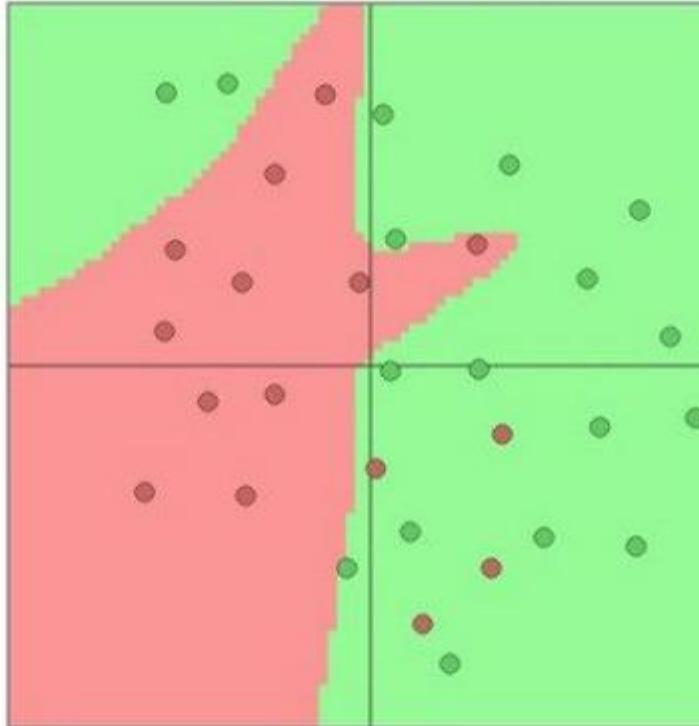


神经网络：

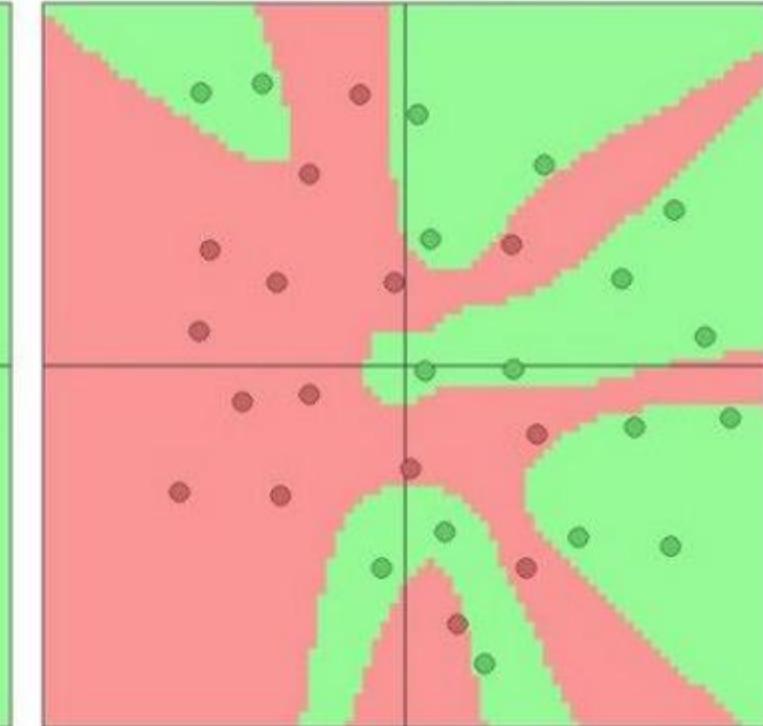
3 hidden neurons



6 hidden neurons

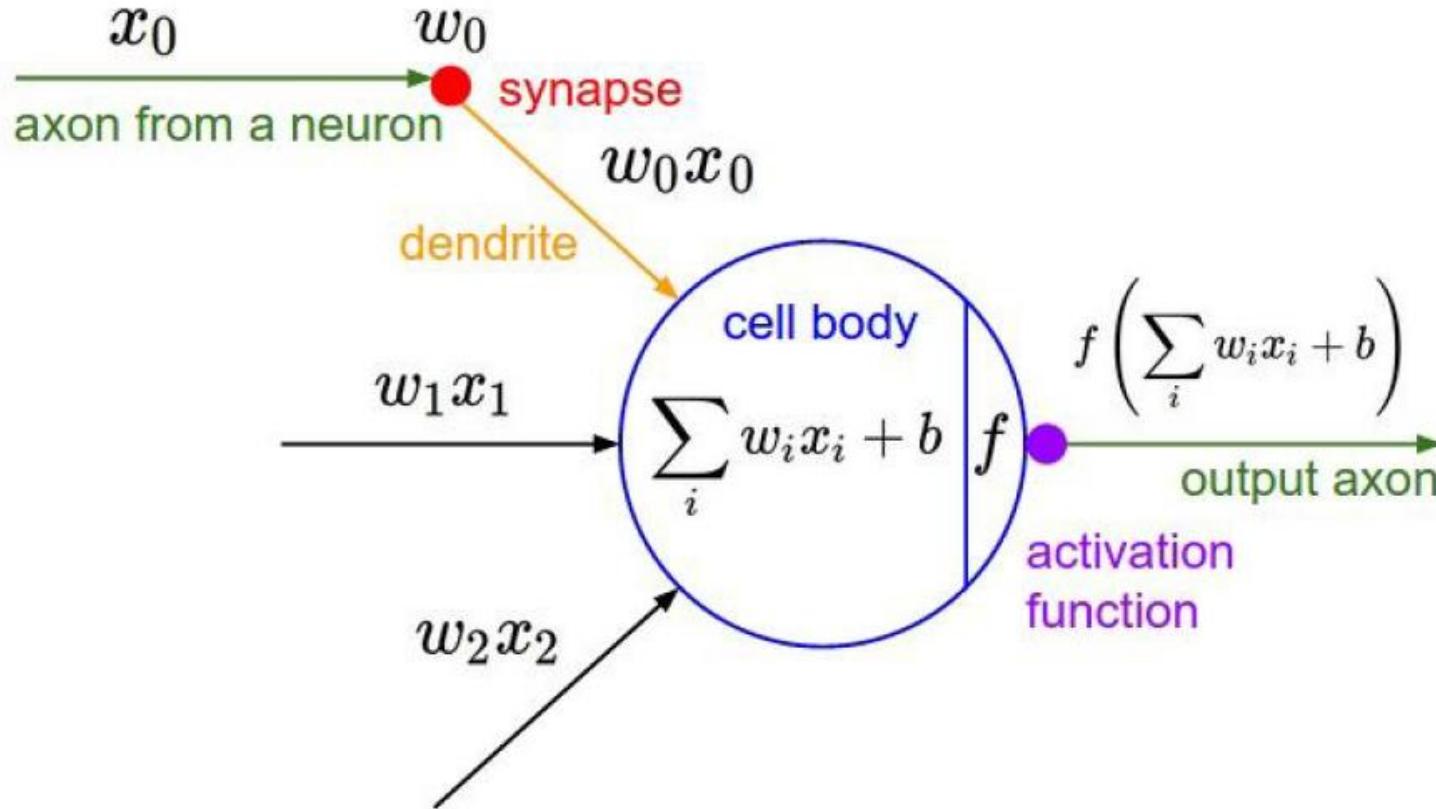


20 hidden neurons



越多的神经元，就越能够表达能复杂的模型

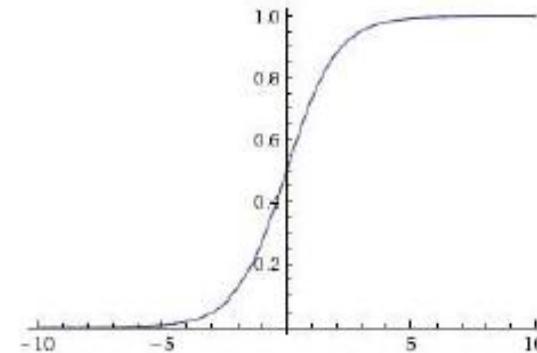
激活函数：



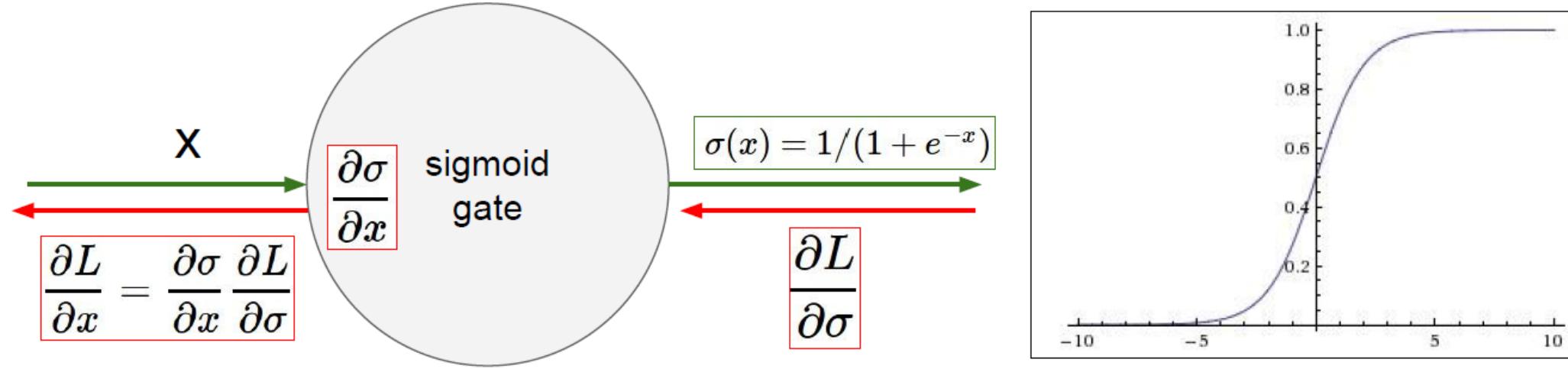
激活函数：

Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

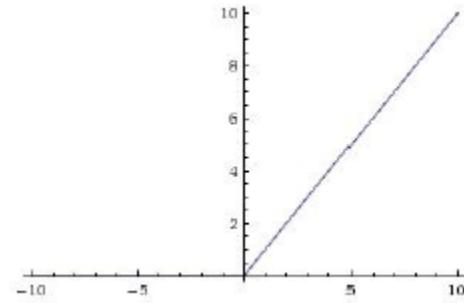


激活函数：

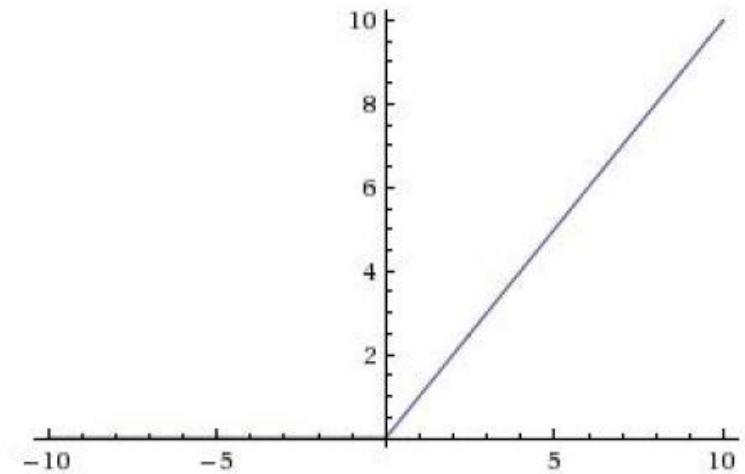
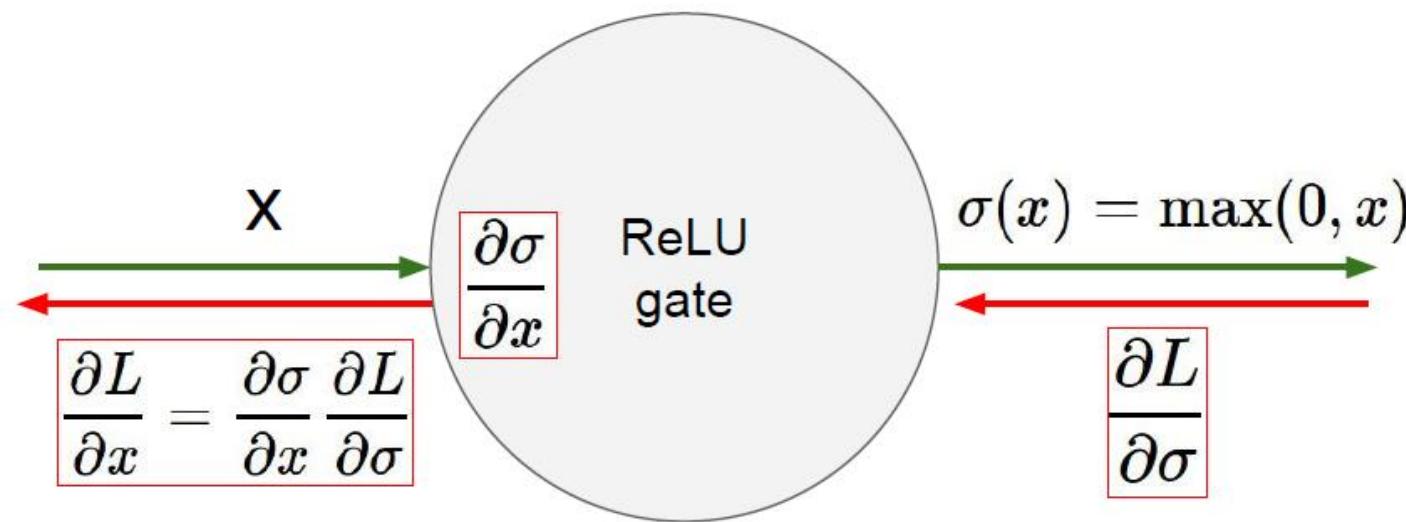


激活函数：

ReLU $\max(0, x)$

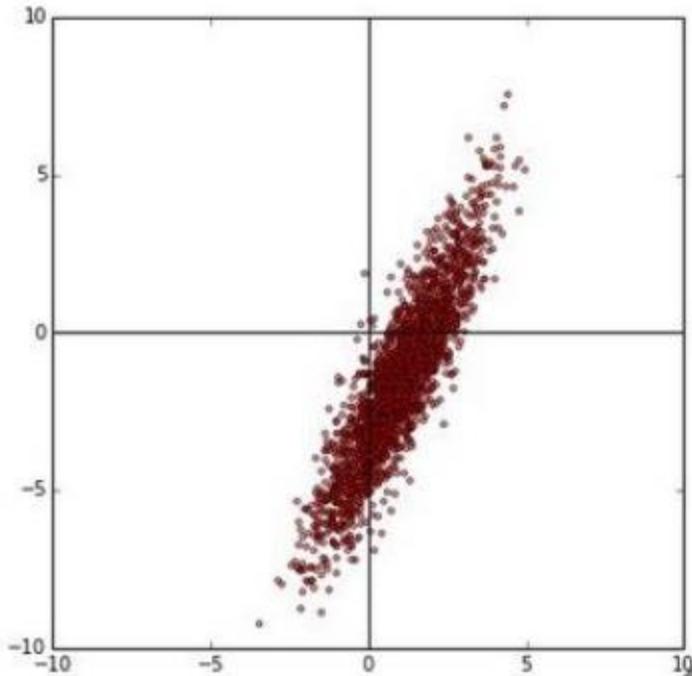


激活函数：

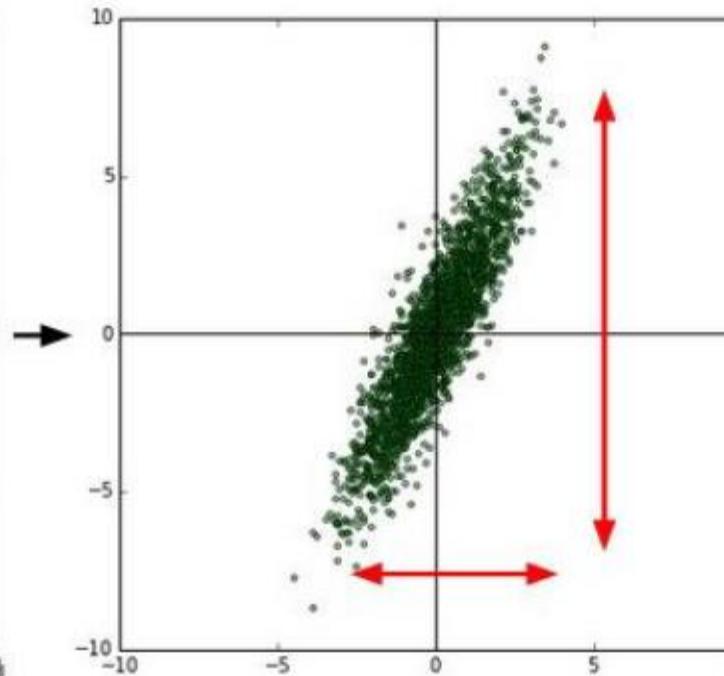


数据预处理：

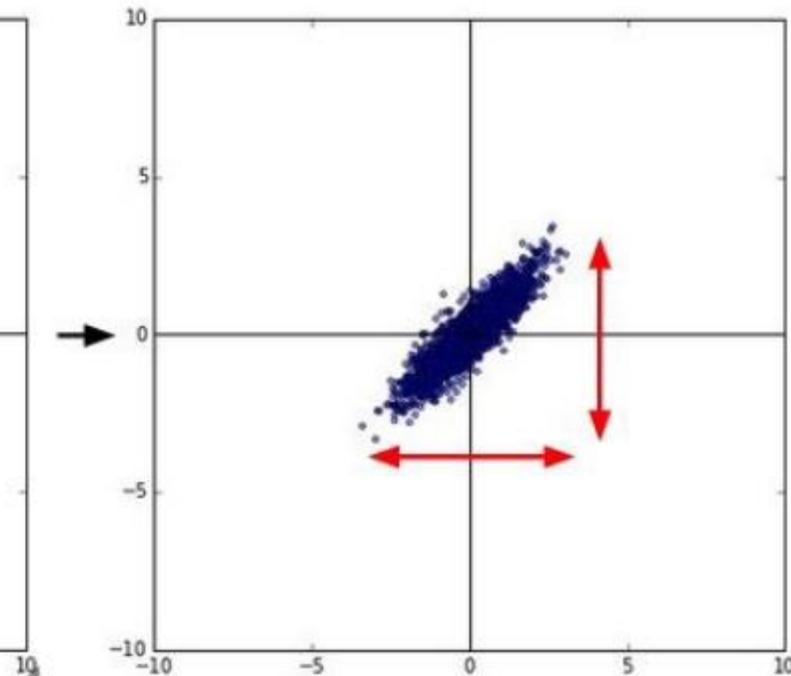
original data



zero-centered data



normalized data

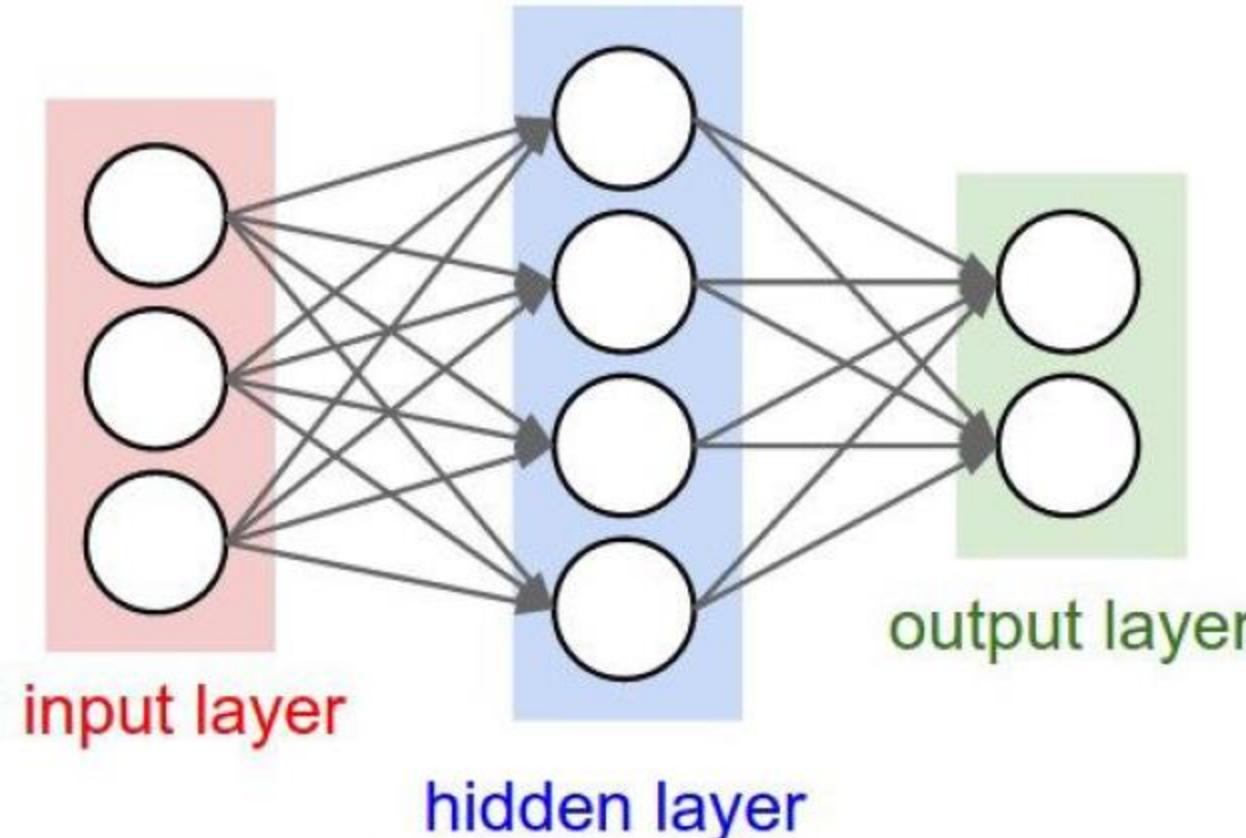


```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

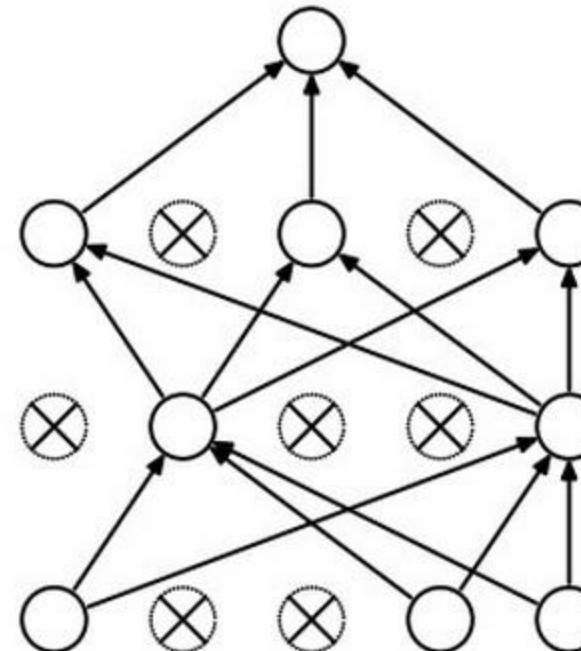
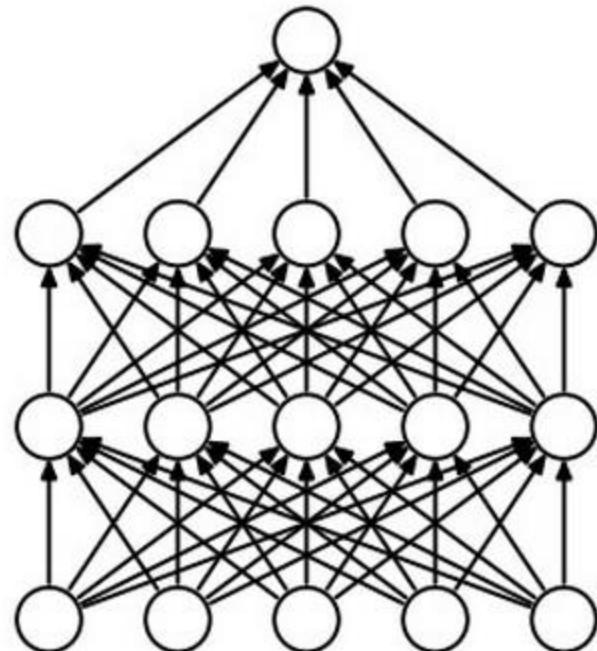
权重初始化：

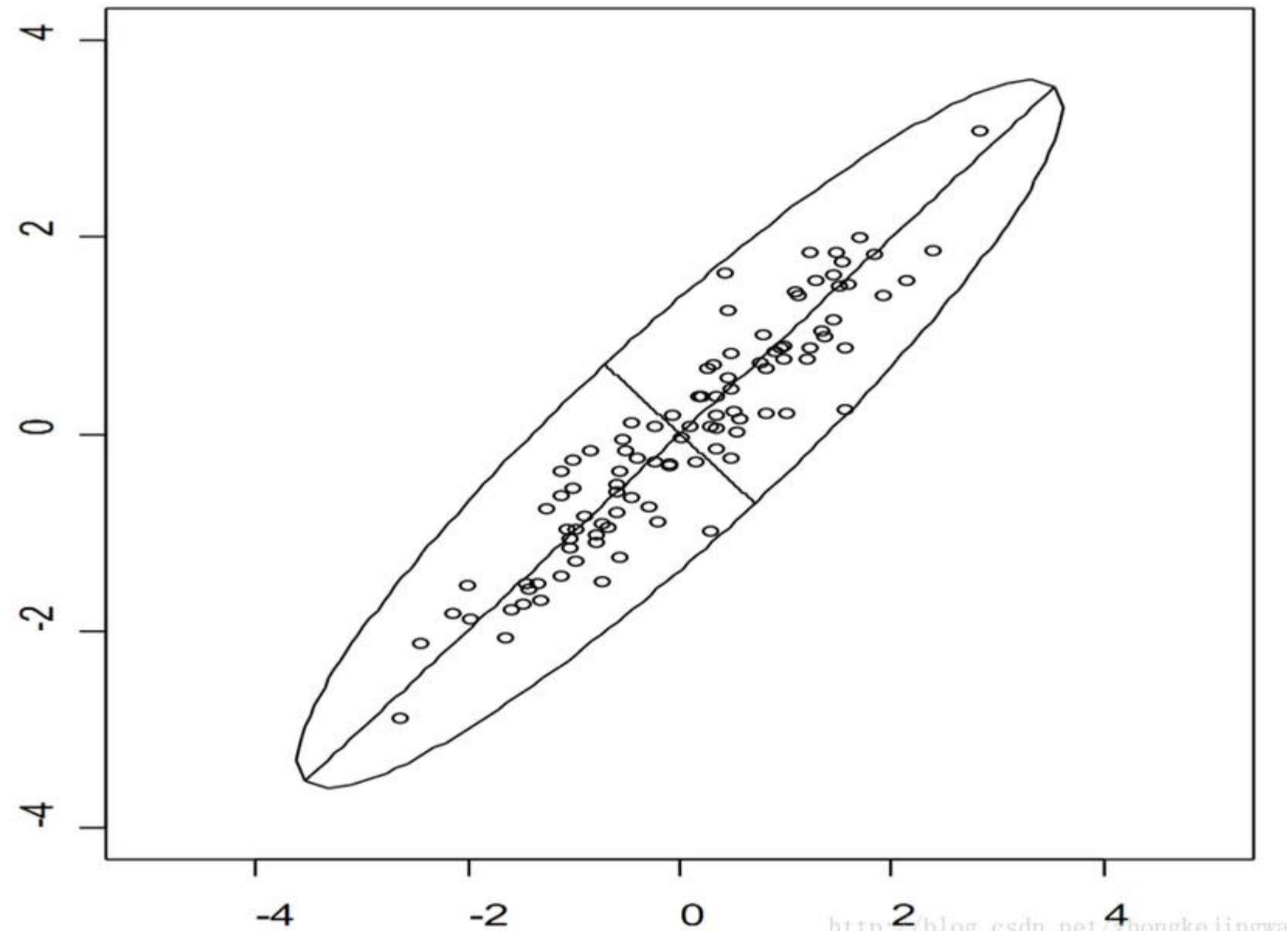
全零值初始化？



```
W = 0.01 * np.random.randn(D, H)
```

DROP-OUT:





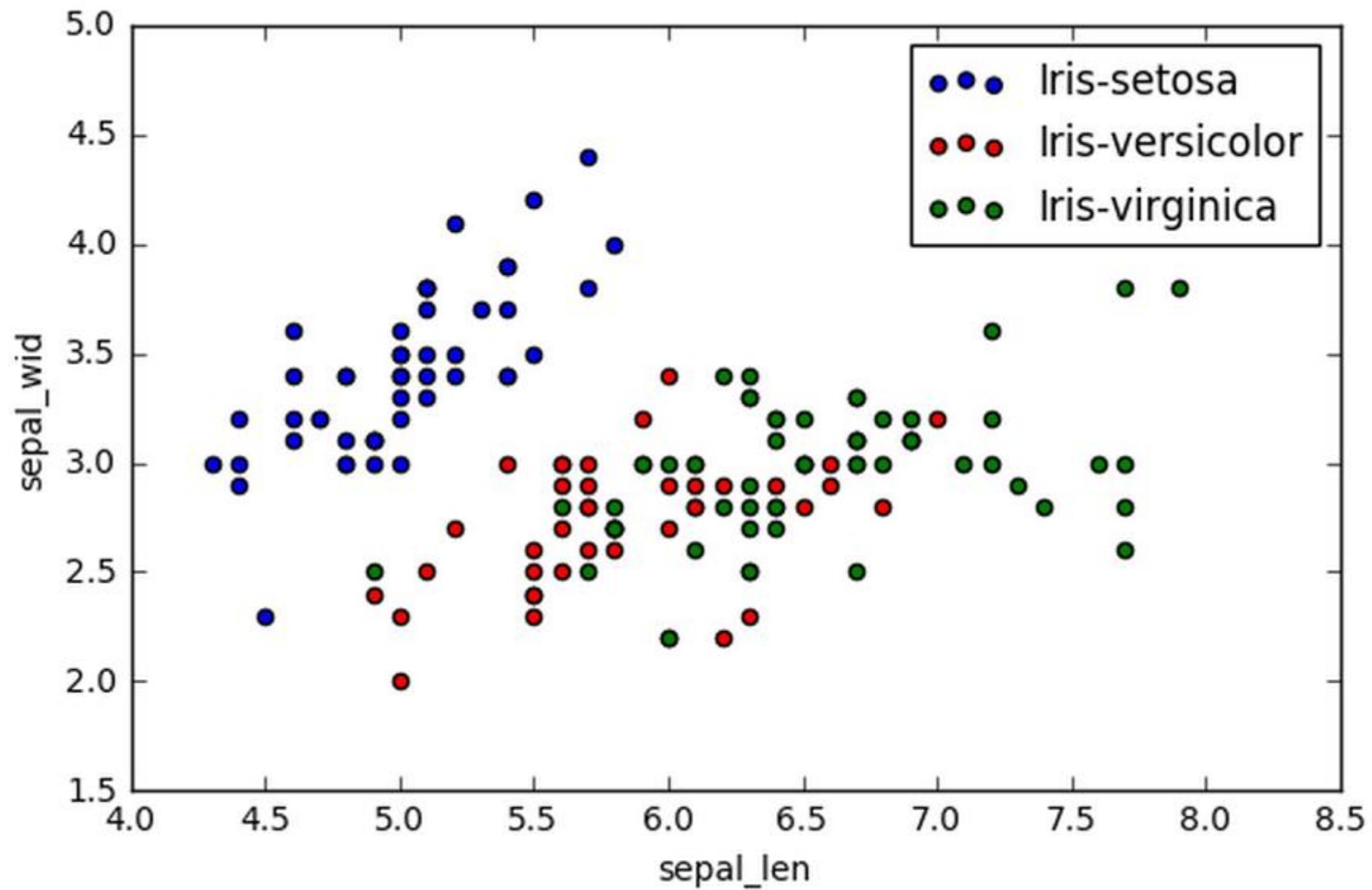
两个特征之间的协方差

$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)$$

协方差矩阵

$$\Sigma = \frac{1}{n-1} ((\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}}))$$

where $\bar{\mathbf{x}}$ is the mean vector $\bar{\mathbf{x}} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_i$.



SVD

A =

5	5	0	5
5	0	3	4
3	4	0	3
0	0	5	3
5	4	4	5
5	4	5	5

$$A = U \Sigma V^T$$

A
 U
 Σ
 V^T

m × n
 m × m
 m × n
 n × n

U =

-0.4472	-0.5373	-0.0064	-0.5037	-0.3857	-0.3298
-0.3586	0.2461	0.8622	-0.1458	0.0780	0.2002
-0.2925	-0.4033	-0.2275	-0.1038	0.4360	0.7065
-0.2078	0.6700	-0.3951	-0.5888	0.0260	0.0667
-0.5099	0.0597	-0.1097	0.2869	0.5946	-0.5371
-0.5316	0.1887	-0.1914	0.5341	-0.5485	0.2429

S =

17.7139	0	0	0
0	6.3917	0	0
0	0	3.0980	0
0	0	0	1.3290
0	0	0	0
0	0	0	0

Vtranspose =

-0.5710	-0.2228	0.6749	0.4109
-0.4275	-0.5172	-0.6929	0.2637
-0.3846	0.8246	-0.2532	0.3286
-0.5859	0.0532	0.0140	-0.8085

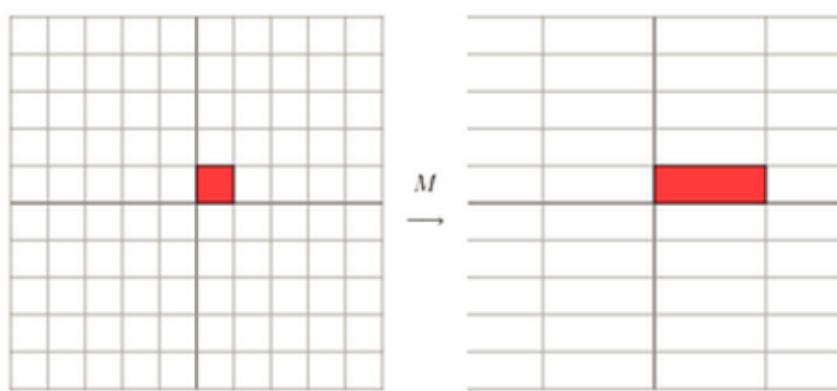
SVD

如果说一个向量 v 是方阵 A 的特征向量，将一定可以表示成下面的形式：

$$Av = \lambda v$$

$$M = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

它其实对应的线性变换是下面的形式：



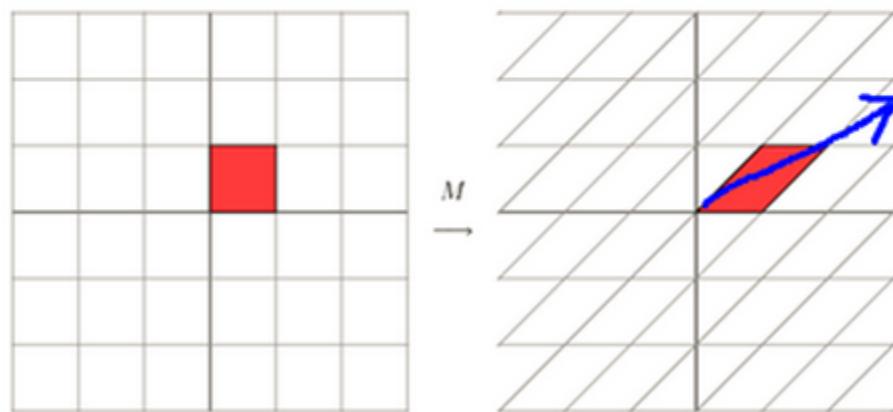
因为这个矩阵 M 乘以一个向量 (x,y) 的结果是：

$$\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3x \\ y \end{bmatrix}$$

SVD

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

它所描述的变换是下面的样子：



这其实是在平面上对一个轴进行的拉伸变换（如蓝色的箭头所示），在图中，蓝色的箭头是一个最主要的变化方向（变化方向可能有不止一个），如果我们想要描述好一个变换，那我们就描述好这个变换主要的变化方向就好了。反过头来看看之前特征值分解的式子，分解得到的 Σ 矩阵是一个对角阵，里面的特征值是由大到小排列的，这些特征值所对应的特征向量就是描述这个矩阵变化方向（从主要的变化到次要的变化排列）

SVD

$$A_{m \times n} \approx U_{m \times r} \Sigma_{r \times r} V^T_{r \times n}$$

$$\begin{matrix} A \\ m \times n \end{matrix} = \begin{matrix} U \\ m \times r \end{matrix} \times \begin{matrix} \Sigma \\ r \times r \end{matrix} \times \begin{matrix} V^T \\ r \times n \end{matrix}$$

SVD

Index Words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
book		1	1						
dads					1			1	
dummies	1						1		
estate						1		1	
guide	1					1			
investing	1	1	1	1	1	1	1	1	1
market	1		1						
real						1		1	
rich					2			1	
stock	1		1				1		
value				1	1				

book	0.15	-0.27	0.04
dads	0.24	0.38	-0.09
dummies	0.13	-0.17	0.07
estate	0.18	0.19	0.45
guide	0.22	0.09	-0.46
investing	0.74	-0.21	0.21
market	0.18	-0.30	-0.28
real	0.18	0.19	0.45
rich	0.36	0.59	-0.34
stock	0.25	-0.42	-0.28
value	0.12	-0.14	0.23

*	3.91	0	0
*	0	2.61	0
*	0	0	2.00

T1	T2	T3	T4	T5	T6	T7	T8	T9
0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0.00	0.34

继续看这个矩阵还可以发现一些有意思的东西，首先，左奇异向量的第一列表示每一个词的出现频繁程度，虽然不是线性的，但是可以认为是一个大概的描述，比如book是0.15对应文档中出现了2次，investing是0.74对应了文档中出现了9次，rich是0.36对应文档中出现了3次；

其次，右奇异向量中一的第一行表示每一篇文档中的出现词的个数的近似，比如说，T6是0.49，出现了5个词，T2是0.22，出现了2个词。

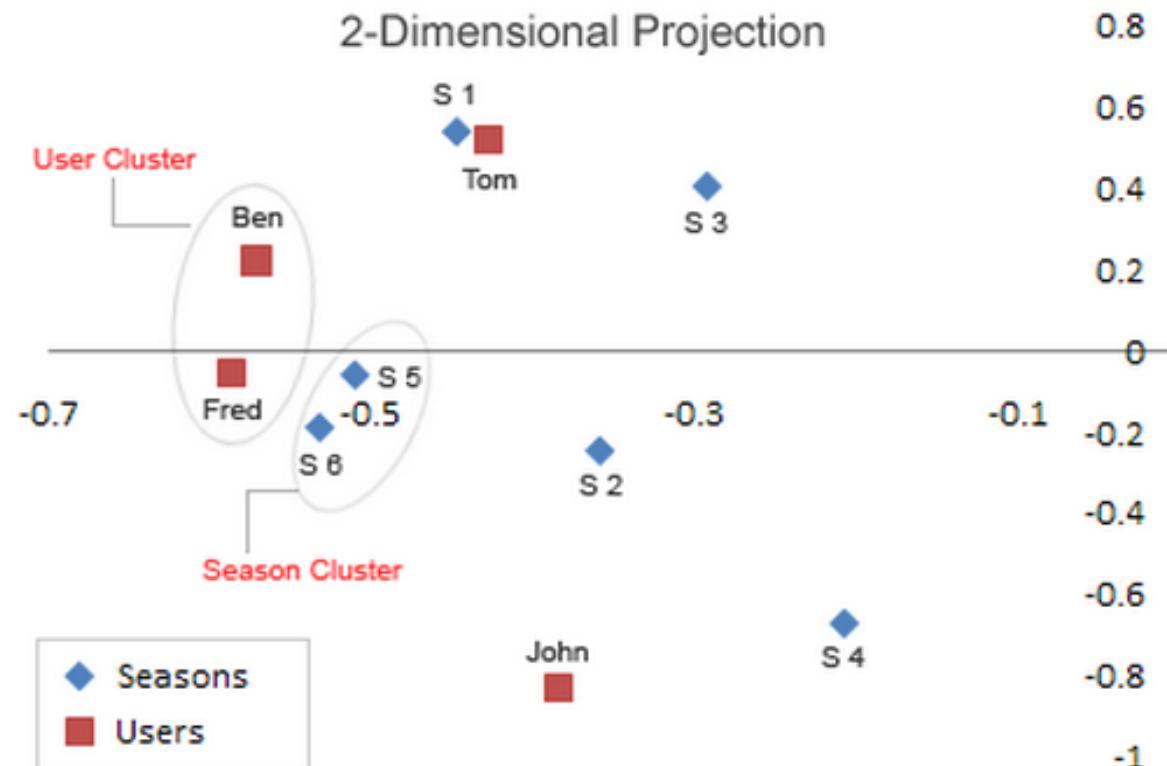
SVD

假设我们有一个矩阵，该矩阵每一列代表一个user，每一行代表一个item。

	Ben	Tom	John	Fred
Season 1	5	5	0	5
Season 2	5	0	3	4
Season 3	3	4	0	3
Season 4	0	0	5	3
Season 5	5	4	4	5
Season 6	5	4	5	5

U		S		V.transpose	
-0.4472	0.5373	17.7139	0.0000	-0.5710	0.2228
-0.3586	-0.2461	0.0000	6.3917	-0.4275	0.5172
-0.2925	0.4033			-0.3846	-0.8246
-0.2078	-0.6700			-0.5859	-0.0532
-0.5099	-0.0597				
-0.5316	-0.1887				

SVD



SVD

我们假设，现在有个名字叫Bob的新用户，并且已知这个用户对season n的评分向量为：[5 5 0 0 0 5]。（此向量为列向量）

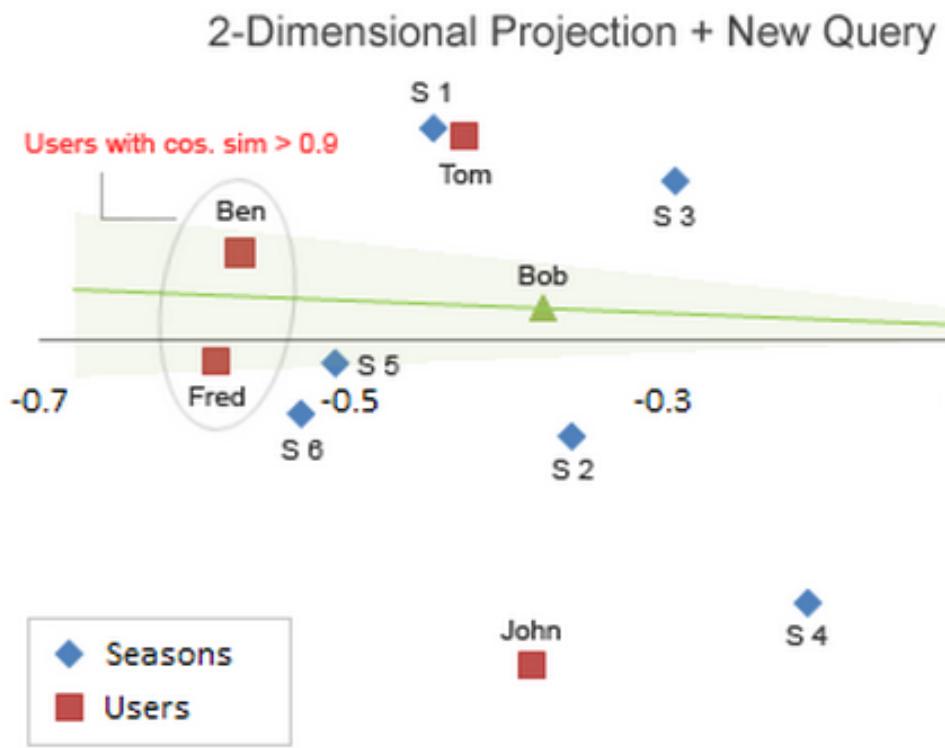
我们的任务是要对他做出个性化的推荐。

我们的思路首先是利用新用户的评分向量找出该用户的相似用户。

$$Bob_{2D} = Bob^T x U_2 x S_k^{-1}$$
$$[5 \ 5 \ 0 \ 0 \ 0 \ 5] \cdot x \begin{bmatrix} -0.4472 & 0.5373 \\ -0.3586 & -0.2461 \\ -0.2925 & -0.4033 \\ -0.2078 & -0.6700 \\ -0.5099 & -0.0597 \\ -0.5316 & -0.1187 \end{bmatrix} x \begin{bmatrix} 17.7139 & 0 \\ 0 & 6.3917 \end{bmatrix}^{-1}$$

$$Bob_{2D} = [-0.3775 \quad 0.0802]$$

SVD



找出最相似的用户，即ben。

观察ben的评分向量为：【5 5 3 0 5 5】。

对比Bob的评分向量：【5 5 0 0 0 5】。

然后找出ben评分过而Bob未评分的item并排序，即【season 5 : 5 , season 3 : 3】。

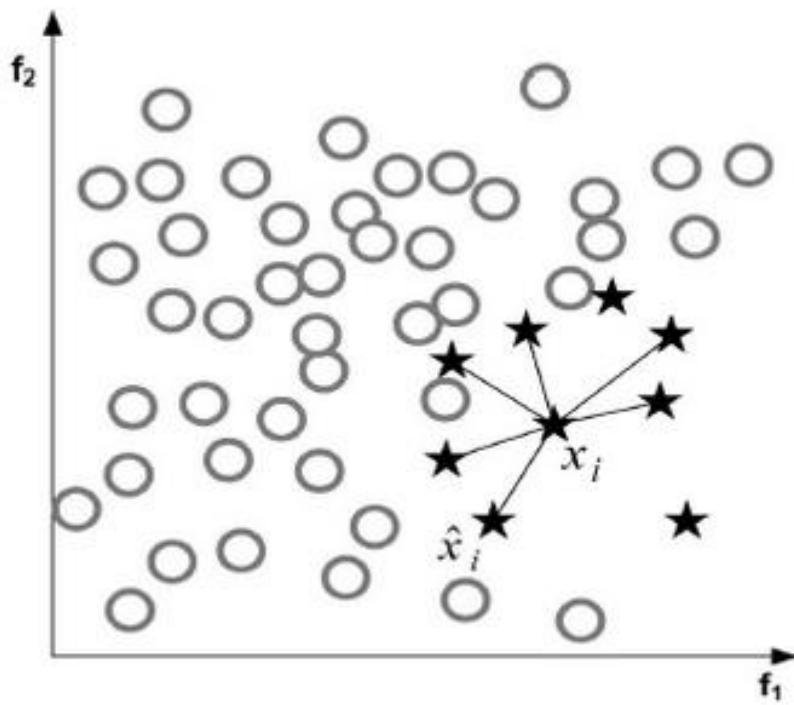
即推荐给Bob的item依次为 season5 和 season3。

(1)对于少数类中每一个样本 x ，以欧氏距离为标准计算它到少数类样本集中所有样本的距离，得到其k近邻。

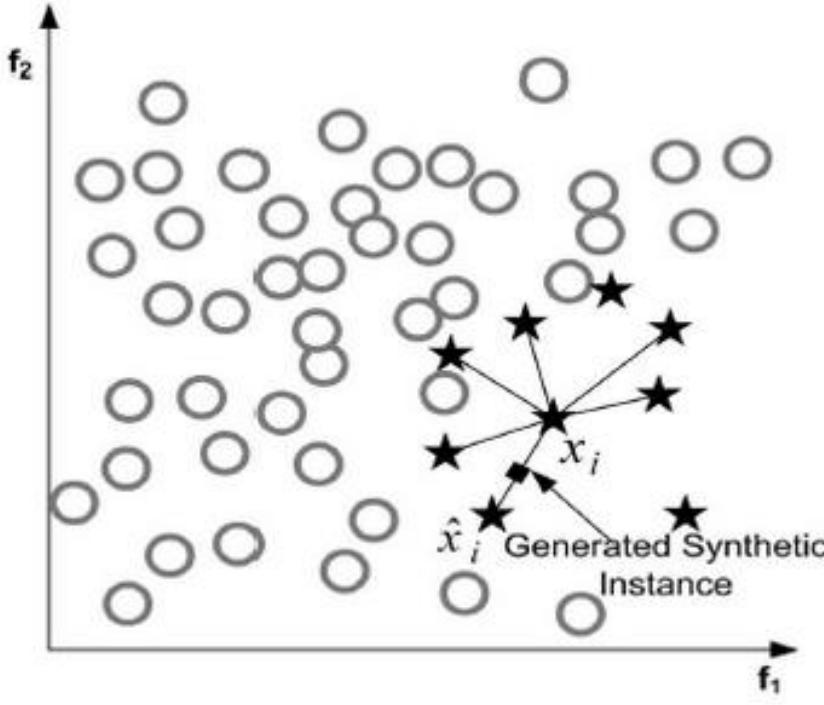
(2)根据样本不平衡比例设置一个采样比例以确定采样倍率N，对于每一个少数类样本 x ，从其k近邻中随机选择若干个样本，假设选择的近邻为 x_n 。

(3)对于每一个随机选出的近邻 x_n ，分别与原样本按照如下的公式构建新的样本。

$$x_{new} = x + rand(0,1) \times (\tilde{x} - x)$$



(a)



(b)

假如某个班级有男生**80**人,女生**20**人,共计**100**人.目标是找出所有女生.

现在某人挑选出**50**个人,其中**20**人是女生,另外还错误的把30个男生也当作女生挑选出来了.

	相关(Relevant),正类	无关(NonRelevant),负类
被检索到 (Retrieved)	true positives(TP 正类判定为正类,例子中就是正确的判定"这位是女生")	false positives(FP 负类判定为正类,"存伪",例子中就是分明是男生却判断为女生,当下伪娘横行,这个错常有人犯)
未被检索到 (Not Retrieved)	false negatives(FN 正类判定为负类,"去真",例子中就是,分明是女生,这哥们却判断为男生--梁山伯同学犯的错就是这个)	true negatives(TN 负类判定为负类,也就是一个男生被判断为男生,像我这样的纯爷们一准儿就会在此处)

通过这张表,我们可以很容易得到这几个值:

$$TP=20$$

$$FP=30$$

$$FN=0$$

$$TN=50$$

http:

access_time sip sport dip dport method uri host origin cookie

uagent refer data

2015-06-16 17:13:55.657 192.168.3.47 59305 183.61.116.23 80 GET /tslog?
cpt=3753&uid=431203084&sn=201&hi=20&pc=0&ctp=1&r=107633097&cf=0&vid=156272711&iku=n&pt=3720&full=0&vvid=143444318334710805d
81&lang=0 p-log.ykimg.com Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/38.0.2125.122 Safari/537.36 http://static.youku.com/v1.0.0540/v/swf/loader.swf
//优酷视频缓冲动画

2015-06-16 17:13:55.662 192.168.60.12 59210 180.149.132.99 80 GET /res/static/thirdparty/connect.jpg?
t=1434446941 pan.baidu.com
BDUSS=BtSnJjaVhMekVhbUhER01QR11iQkJ4WUtoTXQ0QU9uV0YybWxnWkR2bVZsNEpWQVFBQUFBJCQAAAAAAAAAAEAAACXv69Kw7TDtM00w7TQ3M0oAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAJUKW1WVC1tVb1; BAIDUID=1658ADA9250CC737883FBE46A28B2892:FG=1;
PANPSC=1283533338226652406%3A9qIJoSspryQ8hGVQpj0Azun90oj%2BY
%2FIIsdaagvxz9DNymsBbsjgqFFWqZROBe4Wnquwm5aOH140vM1tDIFSft6hOAD5r1J1nbYEI2TevcixKkgKHWS
%2FHadcMRgxGRz9D85dNrnQ4V6AaP8XxTJKWpuFIAM%2FsYpD6Gs%2BA
%2FjmCysH06FhFhB3SCTb6awFuyOCoUVap1E4F7haeq38RLCoCTS9jIPUg5GYY7egd6yyHymxFY%3D netdisk;5.2.0.7;PC;PC-
Windows;6.1.7601;WindowsBaiduYunGuanJia

2015-06-16 17:13:55.655 192.168.61.38 11850 183.131.67.43 80 GET
/viewimage/house/2011_11/30/bj/1322638398616_000/270x180.jpg img.soufun.com Mozilla/5.0 (Windows NT
6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari/537.36 http://newhouse.fang.com/