Classification

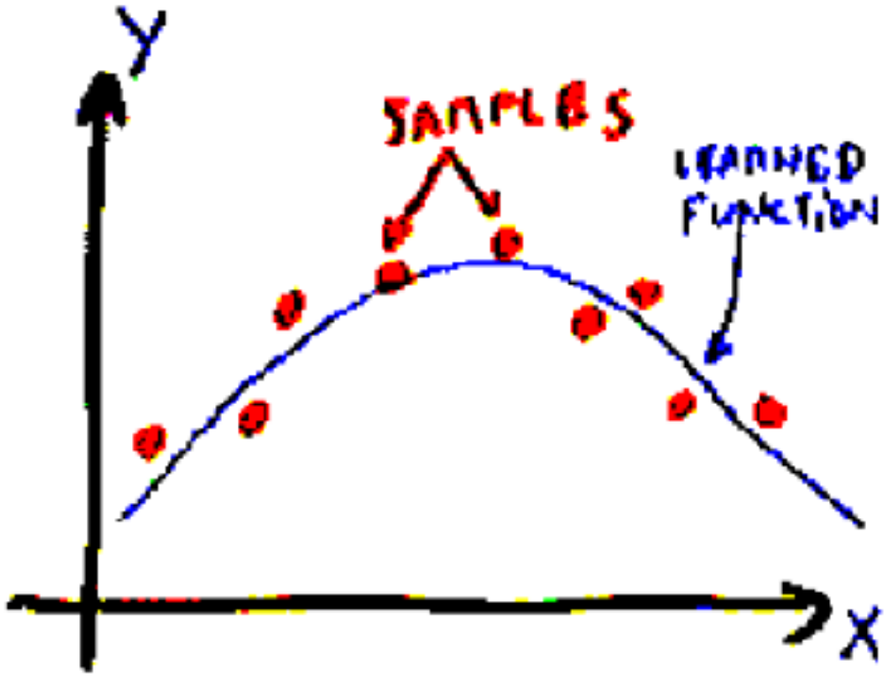
Reference

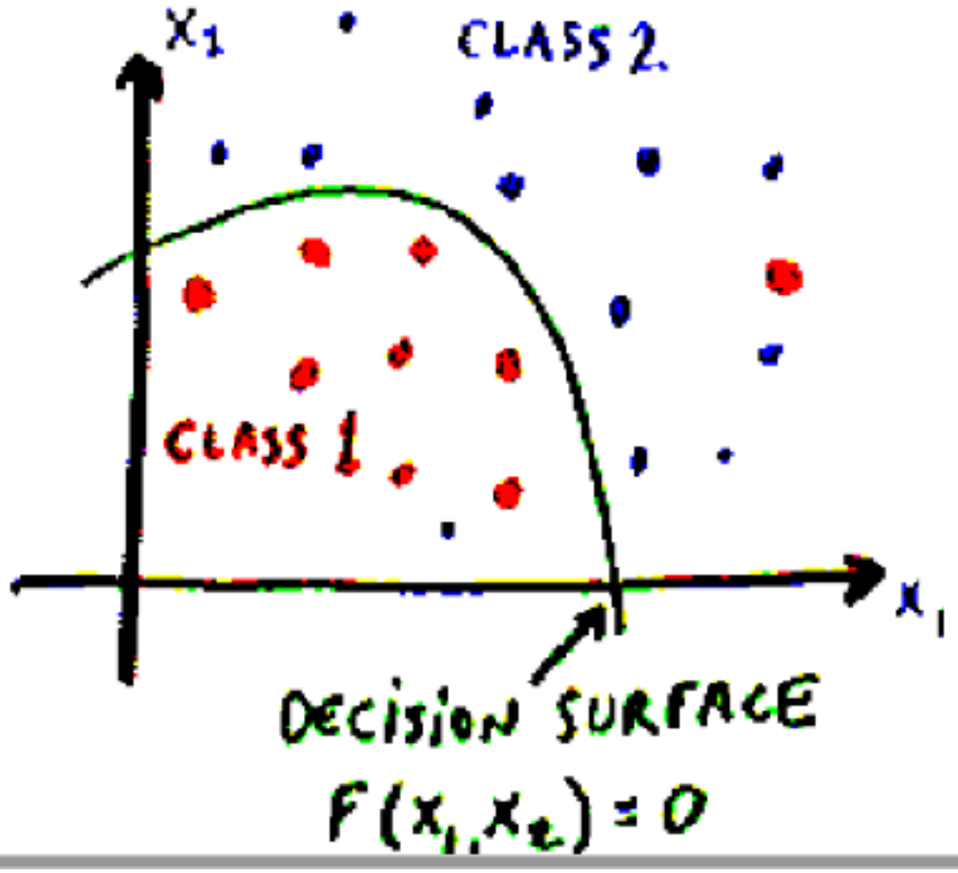
- 数据科学导引，欧高炎 et al.
- 机器学习，周志华.
- The nature of statistical learning theory. Vapnik.
- Foundations of Machine Learning. Mohri, etc.

Reference

- Statistical Modeling: The Two Cultures. Breiman.
  - https://projecteuclid.org/euclid.ss/1009213726
- Problems of empirical inference science. Vapnik.
  - www.lancaster.ac.uk/users/esqn/windsor04/handouts/vapnik.pdf
- Machine Learning and Pattern Recognition. Yann LeCun.
  - https://cs.nyu.edu/~yann/2010f-G22-2565-001/

Two kinds of (supervised) learning



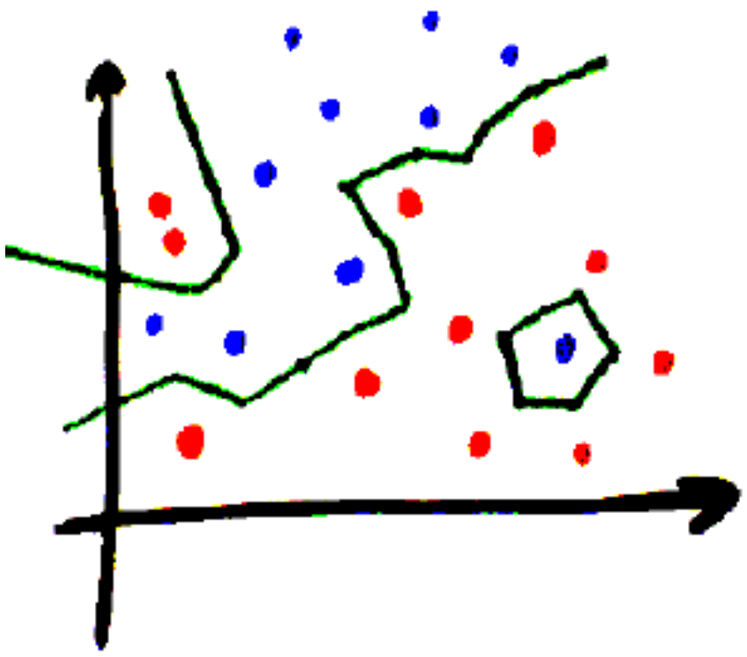


- Regression: also known as "curve fitting" or "function approximation". Learn a continuous input-output mapping from a limited number of examples (possibly noisy).
- Classification: outputs are discrete variables (category labels). Learn a decision boundary that separates one class from the other. Generally, a "confidence" is also desired (how sure are we that the input belongs to the chosen category).

Classification methods

- Geometrical:
  - Nearest Neighbor
  - Logistic regression
  - Support Vector Machine
- Symbolism:
  - Decision Tree
- Connectionism:
  - Perceptron
  - Neural networks
- Bayesian:
  - Naive Bayes

Nearest Neighbor Classifier

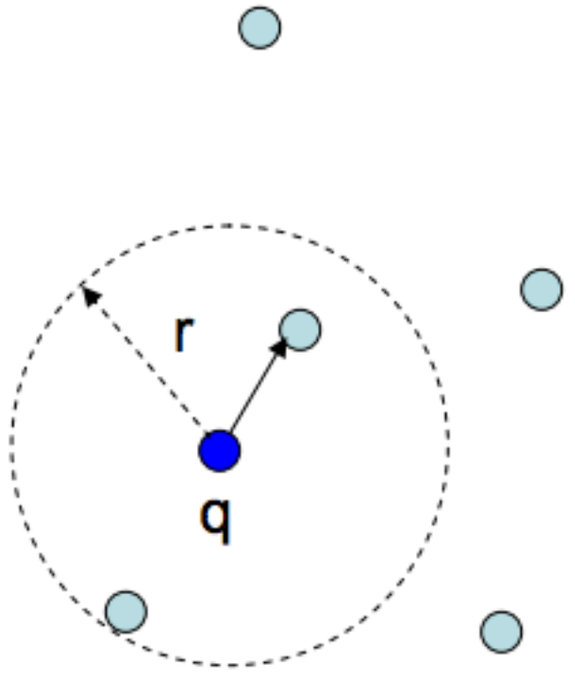


- Instead of insisting that the input be exactly identical to one of the training samples, let's compute the "distances" between the input and all the memorized samples (aka the prototypes).
- 1-Nearest Neighbor Rule: pick the class of the nearest prototype.
- K-Nearest Neighbor Rule: pick the class that has the majority among the K nearest prototypes.
- PROBLEM: What is the right distance measure?
- PROBLEM: This is horrendously expensive if the number of prototypes is large.
- PROBLEM: do we have any guarantee that we get the best possible performance as the number of training samples increases?

Nearest Neighbor

Definition

- Given: a set P of n points in R^d
- Nearest Neighbor: for any query q, returns a point $p \in P$ minimizing $||p-q||$
- r-Near Neighbor: for any query q, returns a point $p \in P$ s.t. $||p-q|| \leq r$ (if it exists)



https://people.csail.mit.edu/indyk/helsinki-1.pdf

Distance Metric

•常用的距离包括:

•欧式距离

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{\sum_{i=1}^{d} (x_{1i} - x_{2i})^2}.$$

•曼哈顿距离

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{i=1}^{d} |x_{1i} - x_{2i}|$$

•马氏距离

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2)^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)}.$$

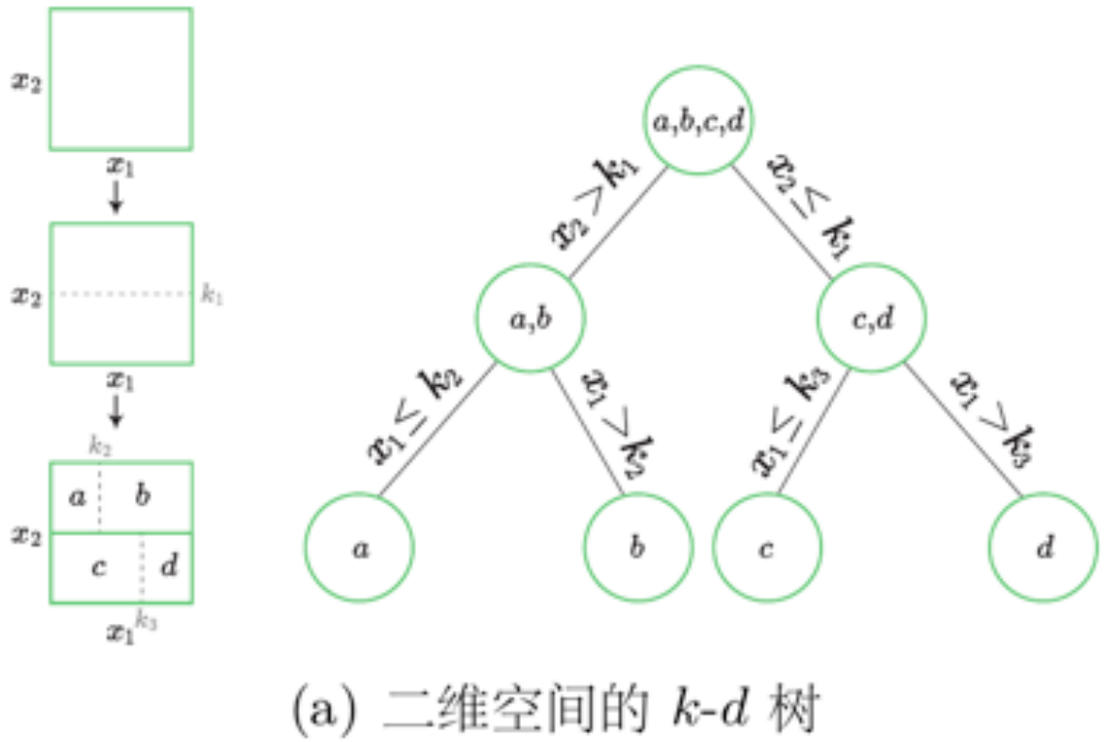
•余弦（相似度）

$$\cos(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{\boldsymbol{x}_1^{\mathrm{T}} \boldsymbol{x}_2}{\|\boldsymbol{x}_1\|_2 \|\boldsymbol{x}_2\|_2}$$

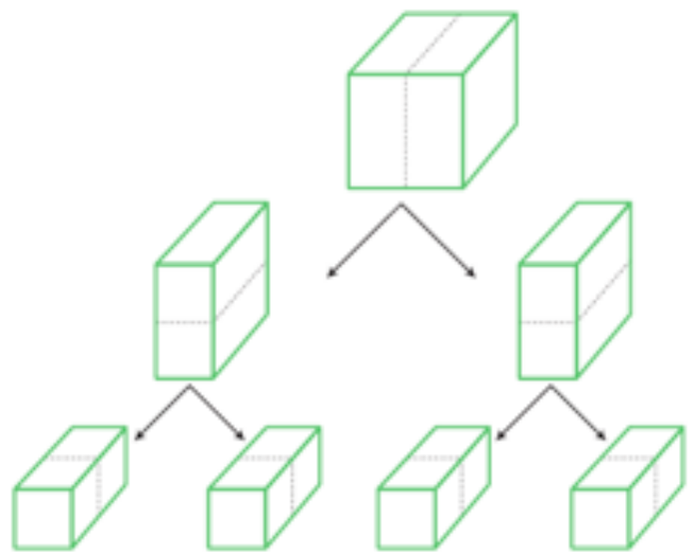
Fast NN search: kd-Trees

•K近邻算法最常用的数据结构为kd树(k-dimensional tree)，它是二叉搜索树在多维空间上的扩展

• 在K近邻算法中，kd树的作用是对训练数据集构建索引，从而在预测时，能够快速找到与测试样本近似的样本。



(a) 二维空间的 k-d 树



(b) 三维空间的 k-d 树

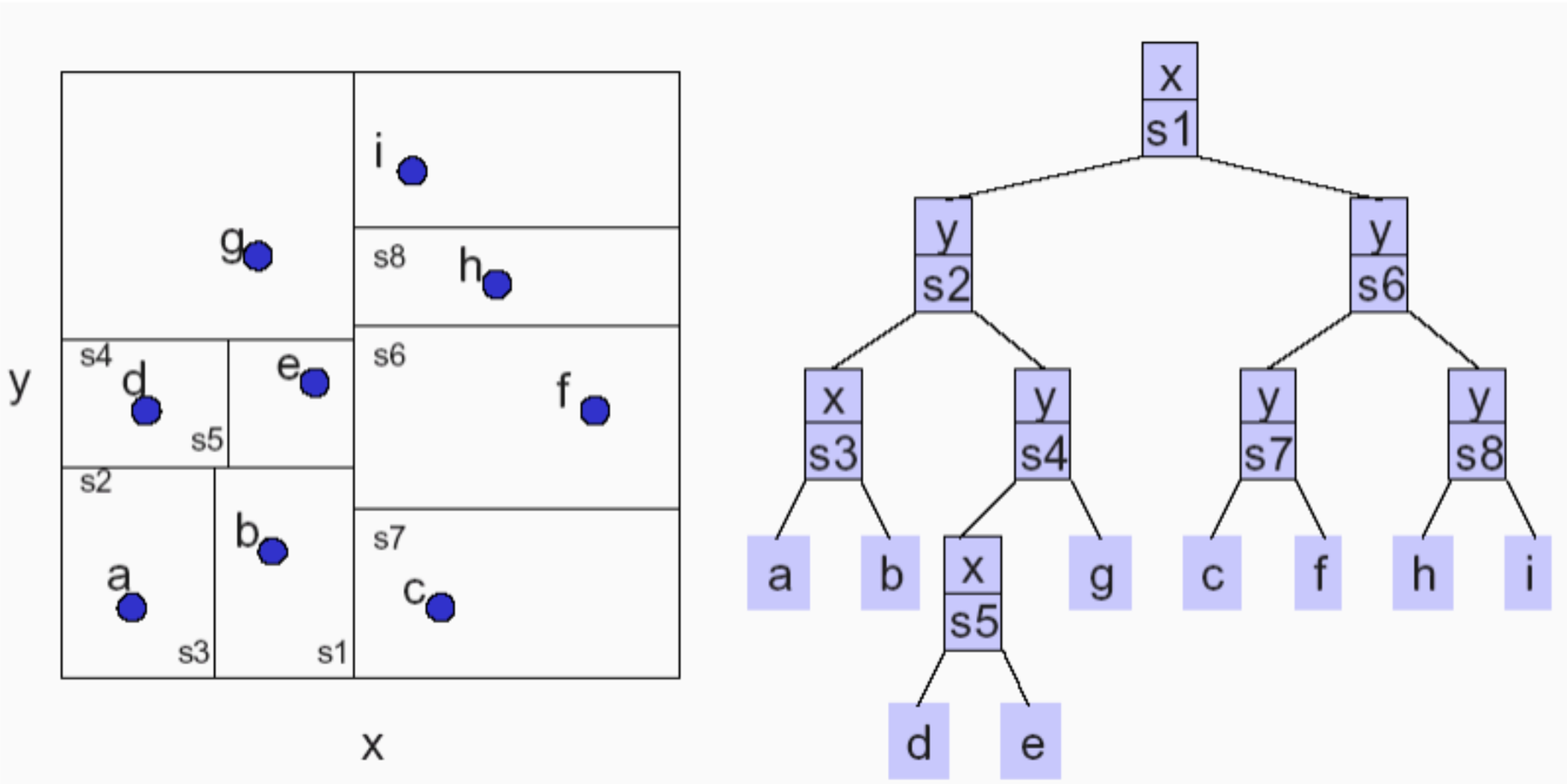
kd-Trees

- Invented in 1970s by Bentley
  - JL Bentley, Binary Search Trees Used for Associative Searching, Communications of the ACM, 1975.

- Idea: Each level of the tree divides the search space by comparing against 1 dimension.

https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf

kd-Trees construction

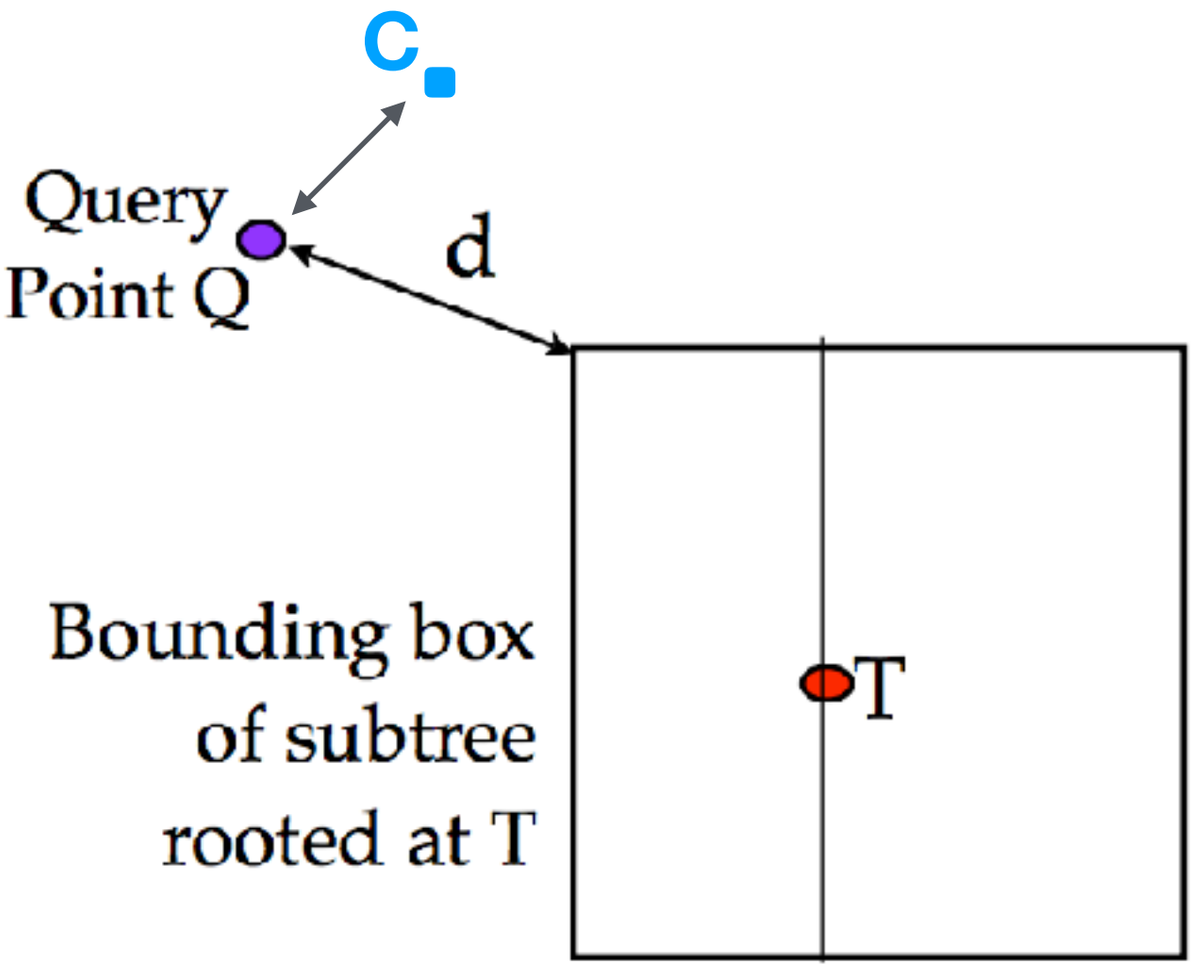


http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf

kd-Trees construction

- Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts
- Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree.

Nearest neighbor search pruning



If d > dist(C, Q), then no point in BB(T) can be closer to Q than C. Hence, no reason to search subtree rooted at T.

https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf

Classification methods

- Geometrical:
  - Nearest Neighbor
  - Logistic regression
  - Support Vector Machine
- Symbolism:
  - Decision Tree
- Connectionism:
  - Perceptron
  - Neural networks
- Bayesian:
  - Naive Bayes

Decision Tree

- 符号学习

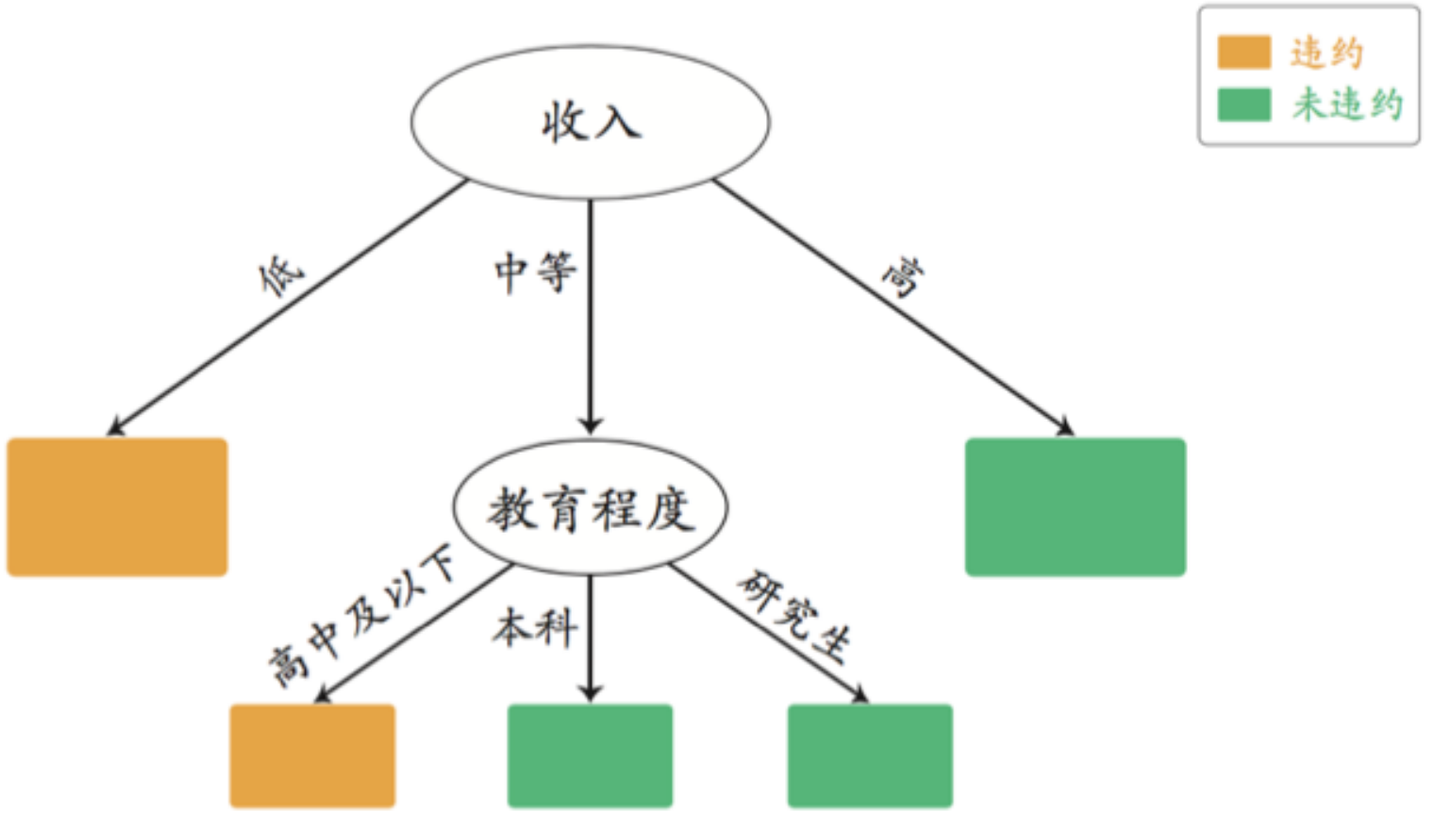
•流感诊断：望闻问切 —— 头痛？发热？等等，诊断结果为感冒或流感

•银行放贷决策：借贷人基本信息——收入？教育程度？婚姻状况？等

编号	性别	收入	教育程度	婚姻状态	是否违约
1	男	高	研究生	未婚	未违约
2	男	低	本科	已婚	违约
3	女	高	高中及以下	未婚	未违约
4	女	中等	高中及以下	已婚	未违约
5	男	高	本科	已婚	未违约
6	男	中等	本科	已婚	违约
7	男	中等	高中及以下	已婚	未违约
8	女	中等	研究生	未婚	违约
9	女	中等	研究生	未婚	违约
10	男	低	高中及以下	未婚	违约

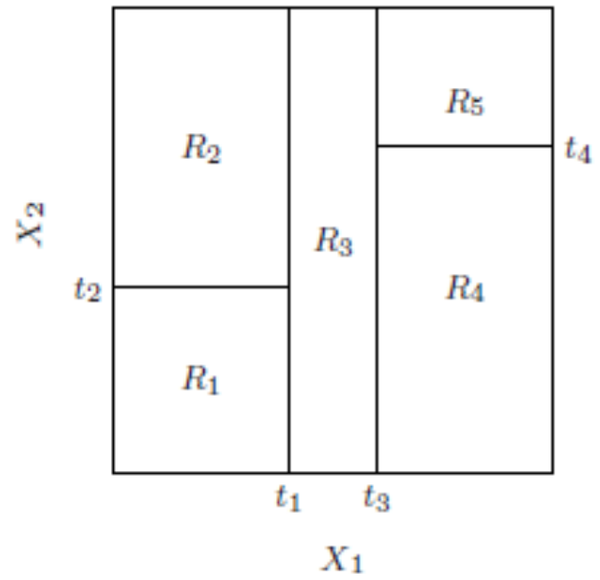
Decision Tree

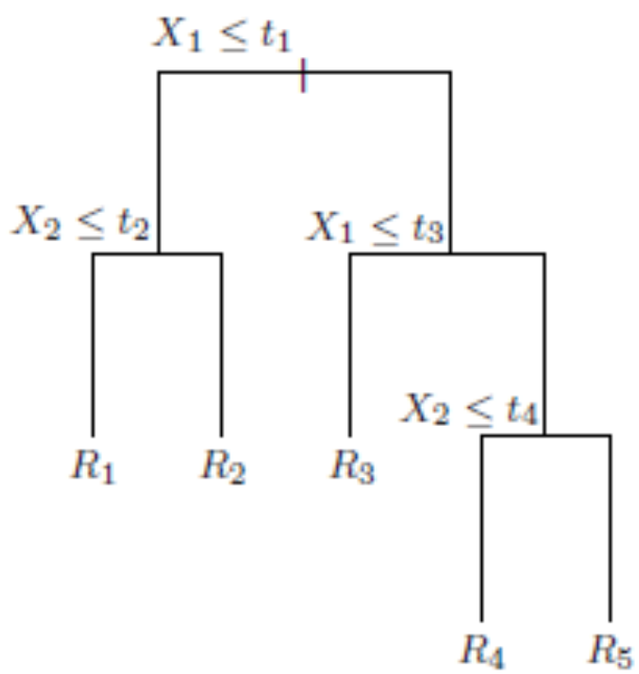
- 可能的一棵决策树
- 规则1：若借贷人收入高，则借贷人不会违约
- 规则2：若借贷人收入中等且为本科或研究生学历，则借贷人不会违约
- 规则3：若借贷人收入中等且为高中及以下学历，则借贷人会违约
- 规则4：若借贷人收入低，则借贷人会违约



Decision Tree

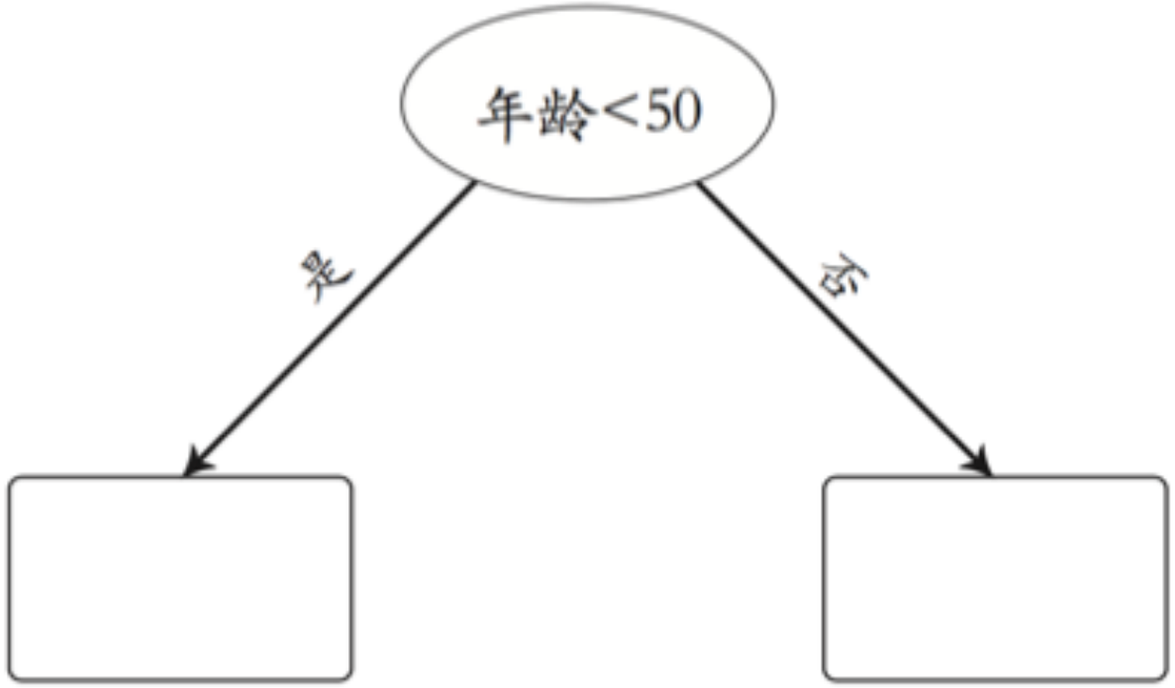
- 决策树是一种基于树形结构的算法
- 内部节点表示一个特征，叶节点表示一个类
- 决策树等价于对空间的方块划分





Decision Tree

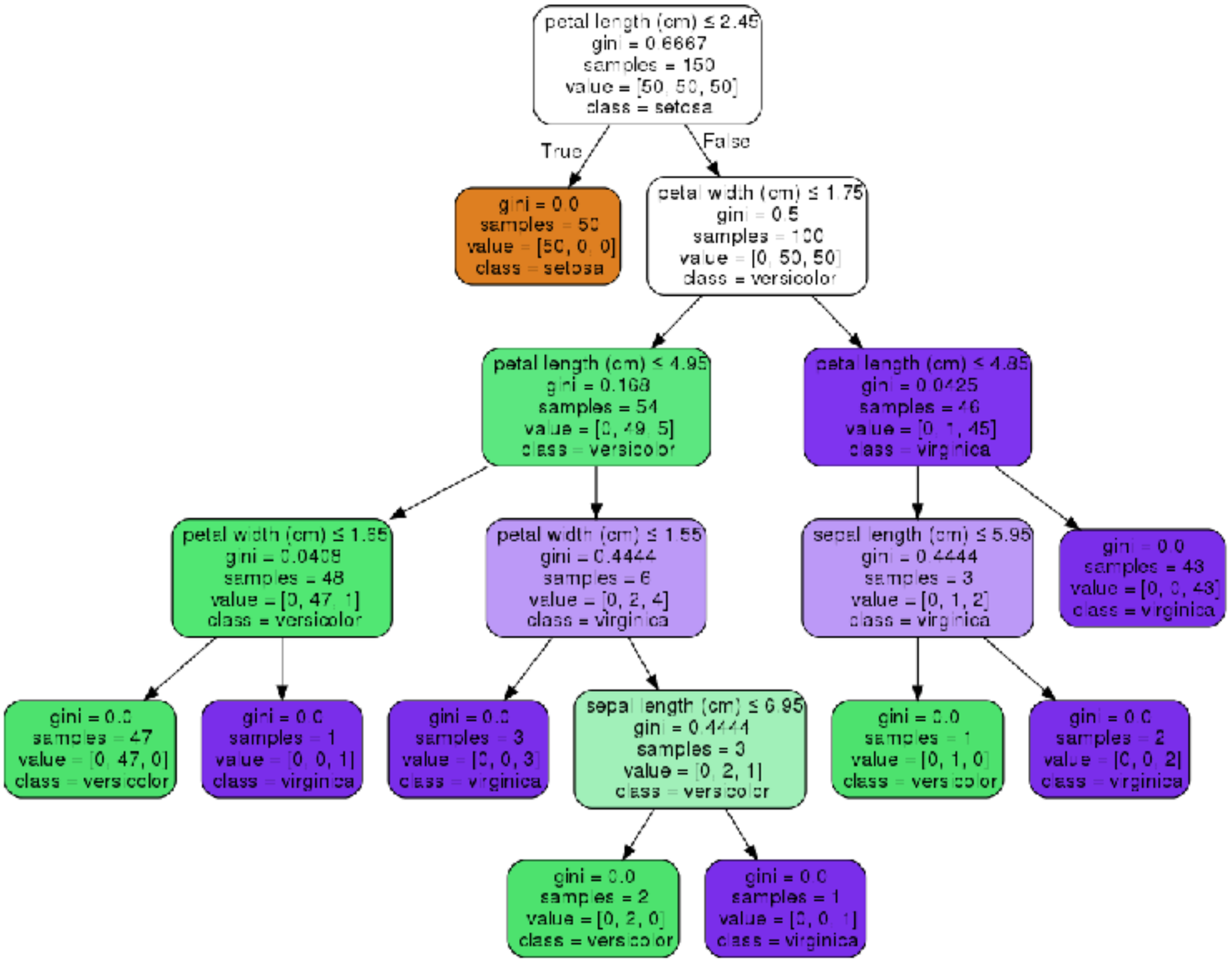
- 从根节点开始，选择节点对应特征（如年龄）
- 选择节点特征分割点，根据分割点分裂节点（如50）
- 核心问题：如何选择节点特征和分割点?



Decision Tree Construction

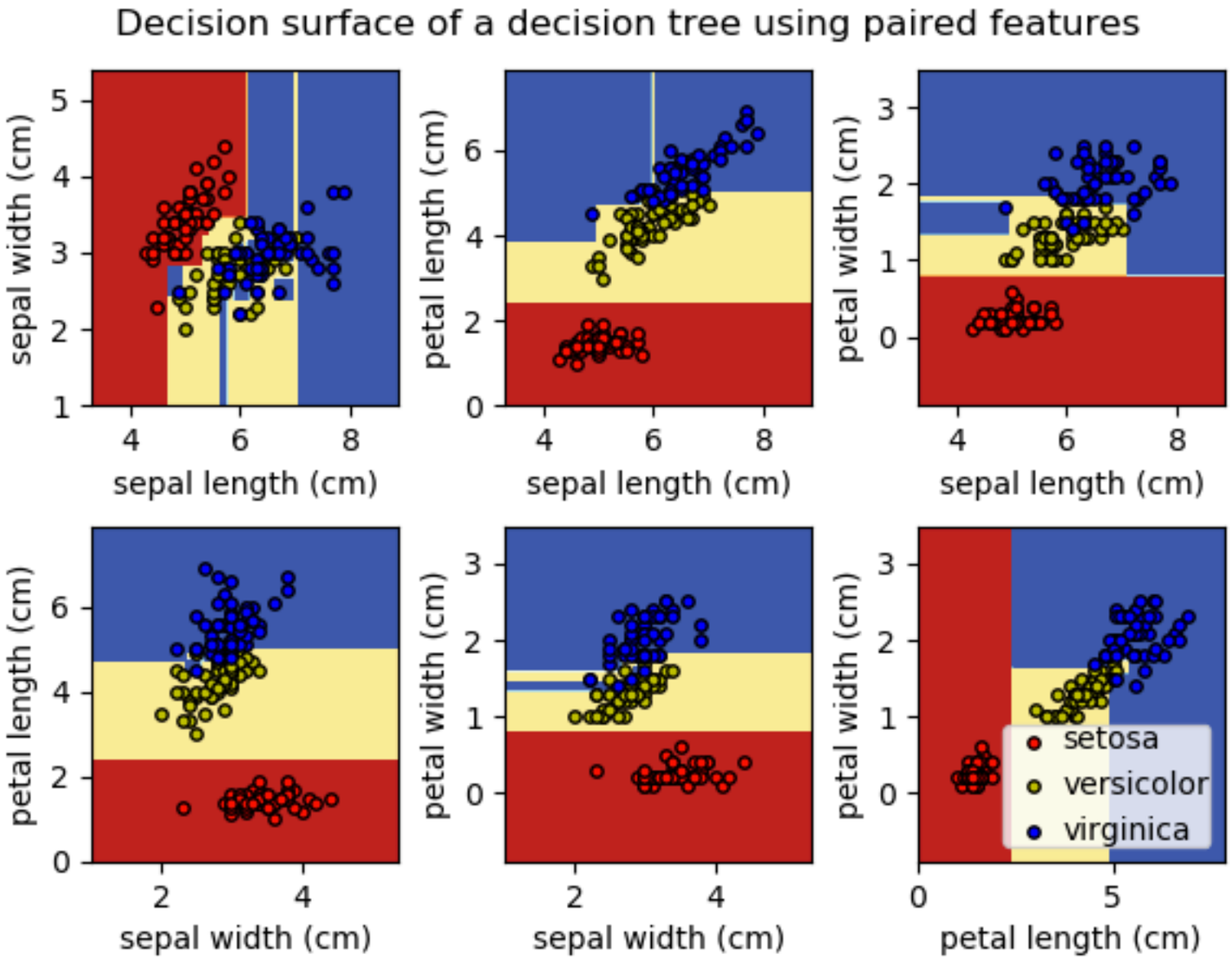
- 决策树生成的递归算法：(样本集合，特征)
  - 生成一个新的节点
  - 如果该节点中的所有样本都属于同一类别，设为叶节点
  - 否则寻找最优特征来划分这些样本
    - 如果没有特征，或者所有特征的取值都相同，设为叶节点
    - 否则取一个特征对样本划分
      - » 将所有样本划分(Partition)，分配到某个分支节点
      - » 如果该样本集合为空，将分支节点设为叶节点；否则对每一个分支节点剩余的样本集合和剩余的特征递归

Decision Tree Construction



https://scikit-learn.org/stable/modules/tree.html

Decision boundary (surface)



https://scikit-learn.org/stable/modules/tree.html

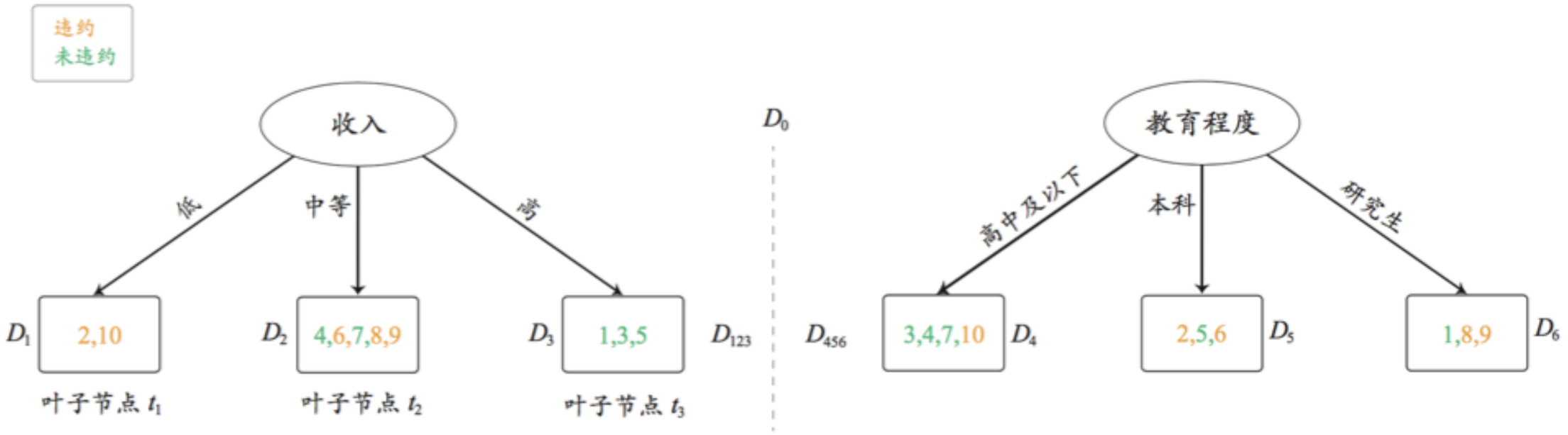
Decision Tree Construction

- Main Algorithms

算法	特征类型	不纯度度量	子节点数量	目标类型
ID3	离散型	信息熵 (信息增益)	K≥2	离散型
C4.5	离散型、连续型	信息熵 (信息增益率)	K≥2	离散型
C5.0	离散型、连续型	信息熵 (信息增益率)	K≥2	离散型
CART	离散型、连续型	Gini指数	K=2	离散型、连续型

Decision Tree Construction

- 不纯度 (impurity)：表示落在当前节点的样本类别分布的均衡程度
- 节点分裂后，节点不纯度应该更低（纯度越来越高）
- 选择特征及对应分割点，使得分裂前后的不纯度(impurity)下降最大



比较$\mathrm{Imp}(D_0) - \mathrm{Imp}(D_{123})$与$\mathrm{Imp}(D_0) - \mathrm{Imp}(D_{456})$的大小来选择节点

特征：收入？教育程度？

Decision Tree Construction

- 节点不纯度的度量

  Gini impurity (vs. Gini coefficient)

  信息熵 (entropy)

  误分率 (misclassification error)

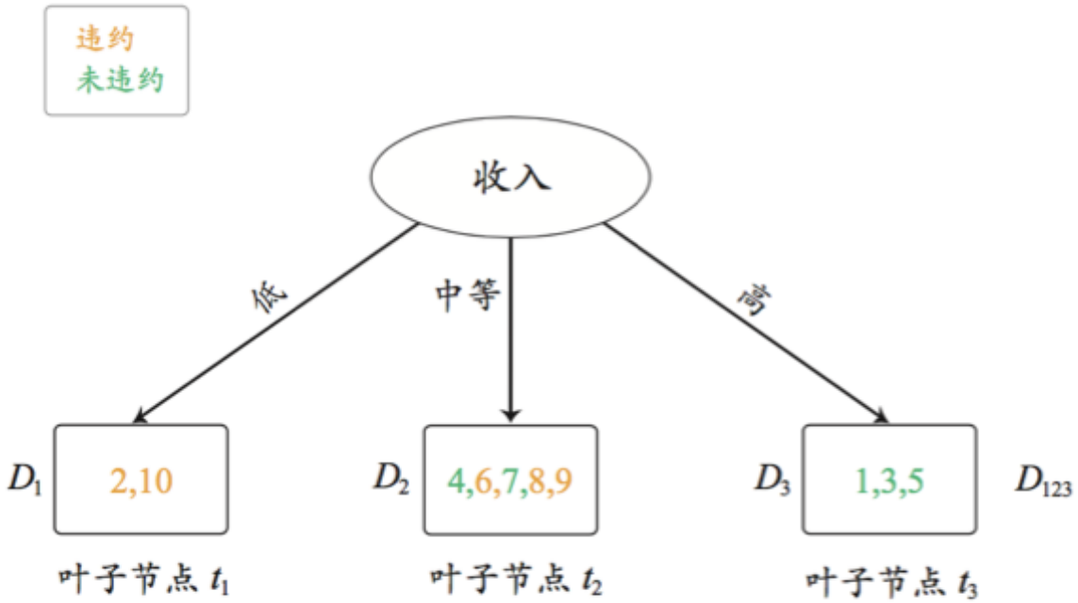
- 决策树的剪枝(Pruning)

  如果树分太细，会造成模型过于复杂和过度拟合(overfitting)。

Gini impurity

- 假设节点对应的样本集合有C类样本，每一类所占份额为p_c，那么
- Gini impurity = $1-\sum_{c=1}^{C} p_c^2$
- 如果C=1，Gini impurity=0

Gini impurity (splitting)



节点编号	违约样本数	未违约样本数
t_1	2	0
t_2	3	2
t_3	0	3

$$\text{Gini}(t_1) = 1 - \left(\left(\frac{2}{2}\right)^2 + \left(\frac{0}{2}\right)^2 \right) = 0;$$

$$\text{Gini}(t_2) = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 \right) = 0.480;$$

$$\text{Gini}(t_3) = 1 - \left(\left(\frac{0}{3}\right)^2 + \left(\frac{3}{3}\right)^2 \right) = 0.$$

Gini impurity (splitting)

- 当一个节点 t 被分成 K 个子节点时，分割的Gini指数为

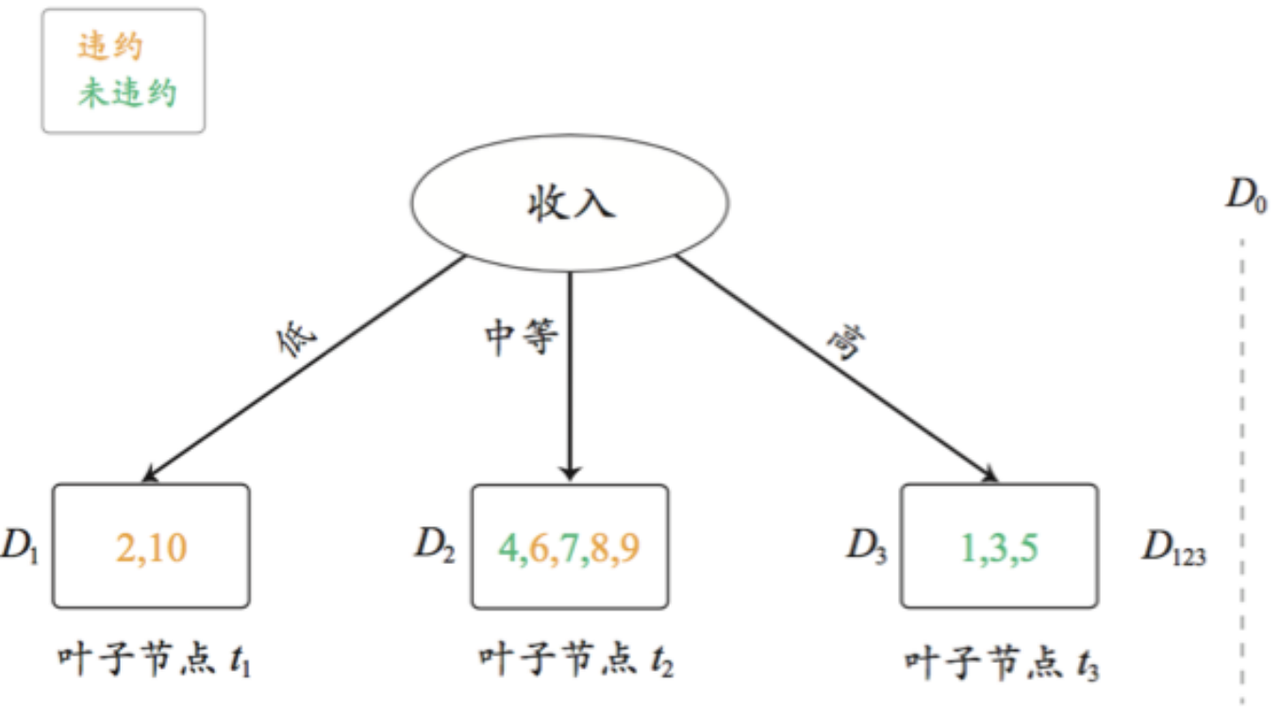
$$\mathrm{Gini}_{\mathrm{split}} = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{Gini}(t_k)$$

← 分裂后某个字节点的Gini指数

按照子节点中样本数量进行加权

- n_k 是子节点 t_k 的样本数量，n 是父节点 t 的样本数量
- 挑选使$\mathrm{Gini}(t) - \mathrm{Gini}_{\mathrm{split}}$最大的特征进行分裂

Gini impurity (splitting)



D_0

$$\text{Gini}(t_0) = 1 - (\frac{5}{10})^2 - (\frac{5}{10})^2 = 0.5$$

$$\text{Gini}_{\text{split}} = \frac{2}{10} \times 0 + \frac{5}{10} \times 0.480 + \frac{3}{10} \times 0 = 0.240,$$

$$\text{Gini}(t_0) - \text{Gini}_{\text{split}} = 0.260$$

- 同理计算出根据特征“教育程度”、“婚姻状态”和“性别”分裂根结点后的Gini指数下降值分别为0.173、0.020、0，故最终选择**“收入”**进行第一次分裂

Entropy

Entropy

> My greatest concern was what to call it. I thought of calling it 'information', but the word was overly used, so I decided to call it 'uncertainty'. When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, 'You should call it **entropy**, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.

—Conversation between Claude Shannon and John von Neumann regarding what name to give to the "measure of uncertainty" or attenuation in phone-line signals [33]

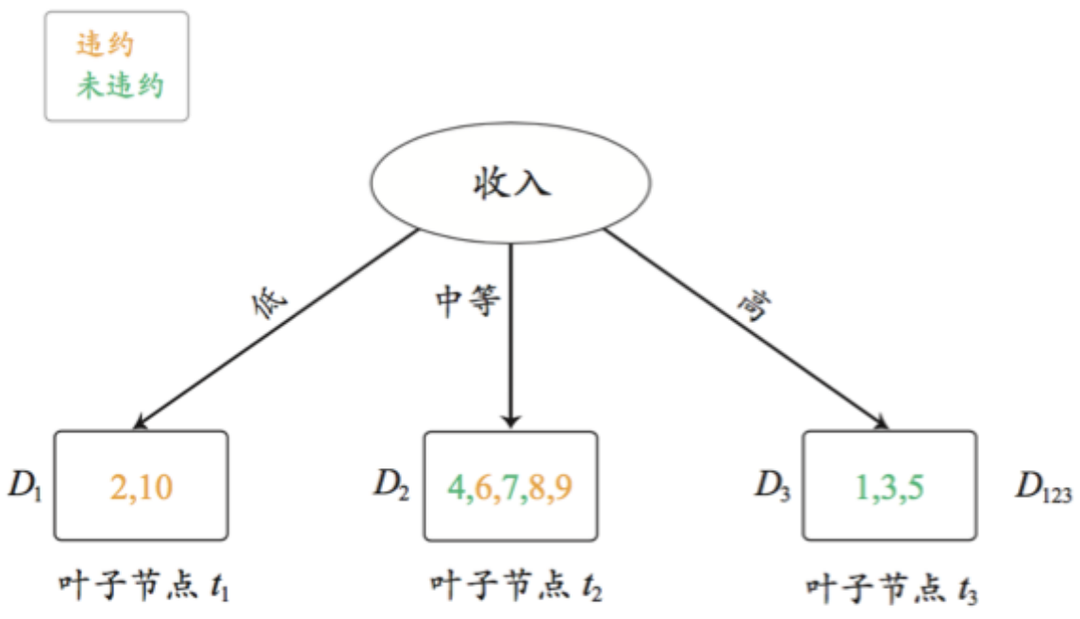
Credit: Wikipedia

Entropy (impurity)

- 假设节点t对应的样本集合有C类样本，每一类所占份额为 p_c，那么
- Entropy(t) = $\sum_c p_c \log_2(\frac{1}{p_c})$

- 当样本均匀分布在每一个类中时， 熵为 $\log_2(C)$，说明不纯度大
- 当所有的样本属于同一个类时， 熵为0，说明不纯度小

Entropy (impurity)



D_0

$$\text{Entropy}(t_1) = -\frac{2}{2}\log_2\frac{2}{2} - \frac{0}{2}\log_2\frac{0}{2} = 0;$$

$$\text{Entropy}(t_2) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971;$$

$$\text{Entropy}(t_3) = -\frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0.$$

Entropy impurity (splitting)

- 节点分裂前后的信息熵的下降值，称为信息增益 (information gain)

$$\text{InfoGain} = \text{Entropy}(t_0) - \sum_{k=1}^{K} \frac{n_k}{n} \text{Entropy}(t_k)$$

- n_k 是子节点 t_k 的样本数量，n 是父节点 t_0 的样本数量
- 挑选使 InfoGain 最大的特征进行分裂
- ID3算法采用信息增益。
- 缺点：倾向于分裂成很多的小节点（节点的样本数较小），容易造成过度拟合

Entropy impurity (splitting)

- 使用节点分裂的子节点的样本数信息，得到信息增益率

$$\text{SplitInfo} = -\sum_{k=1}^{K} \frac{n_k}{n} \log_2(\frac{n_k}{n})$$

$$\text{InfoGainRatio} = \frac{\text{InfoGain}}{\text{Splitinfo}}$$

- n_k是子节点 k 的样本数量， n 是父节点t的样本数量
- C4.5算法采用信息增益率。**避免分裂成过多的子节点**

Larger InfoGain -> Smaller impurity
Smaller SplitInfo -> Less Splitting

误分率 (impurity)

- 含义：当按照多数类来预测当前节点样本的类别时，被错误分类的数据的比例
- 节点 t 的误分率为 $1 - \max(p_1, \cdots, p_K)$
- 当样本均匀分布在每一个类中， 误分率为1-1/K，不纯度最大
- 当样本分布在同一个类中， 误分率为0，不纯度最小

Decision Pruning

- 如果树分太细，会造成模型过于复杂和过度拟合。

- 预剪枝。在决策树生成的过程中，设置一个不纯度下降值阈值，当考虑分裂节点时，如果不纯度下降值小于给定的阈值，则停止对节点进行分裂。

- 后剪枝。利用训练集完整建立决策树模型，建立完成后再对决策树进行剪枝操作

- 平衡 Impurity(拟合度)和树的大小(复杂度)

Decision Tree Summary

- 决策树生成后转换成IF-THEN规则的集合，使得决策树模型的**可解释性较好**

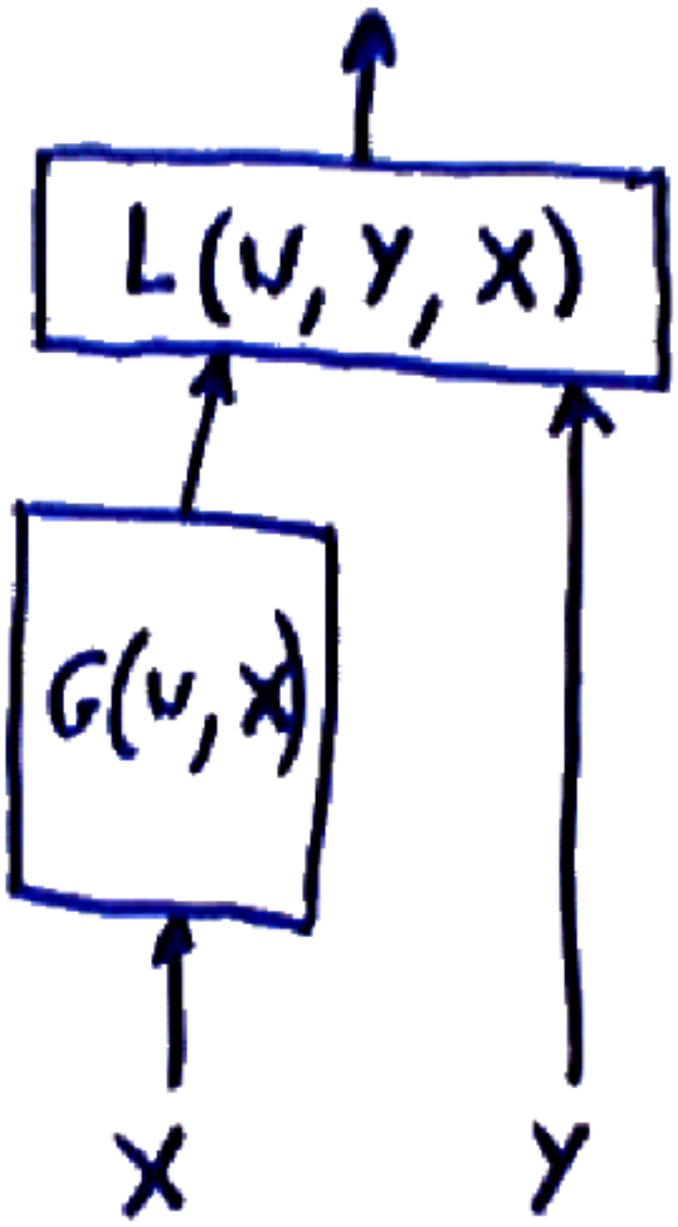
- 模型不稳定，方差大，训练集的改变很容易导致性能变化
  - 解决方法：集成方法，例如随机森林和AdaBoost

- 决策树由于使用贪婪算法的思想，即在每次分裂时选择当前情况下带来信息增益最高的属性进行分裂，**容易陷入局部最优解**

Classification methods

- Geometrical:
  - Nearest Neighbor
  - Logistic regression
  - Support Vector Machine
- Symbolisme:
  - Decision Tree
- Connectionism:
  - Perceptron
  - Neural networks
- Bayesian:
  - Naive Bayes

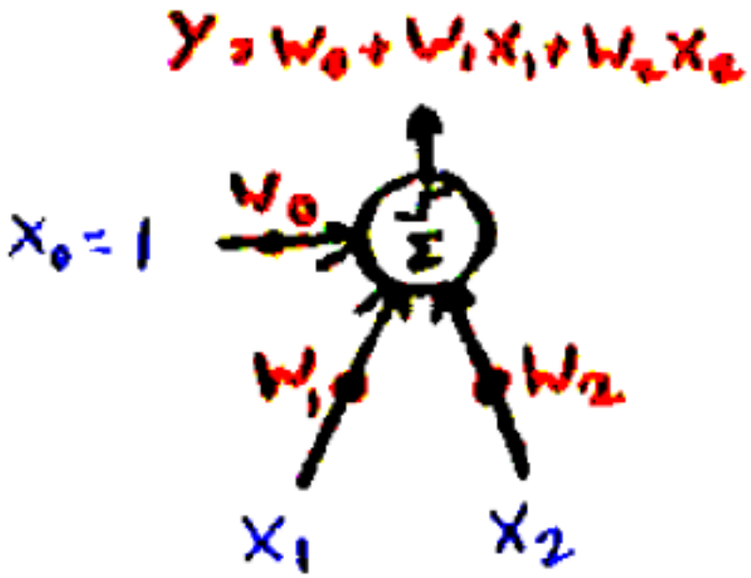
Learning as Function estimation

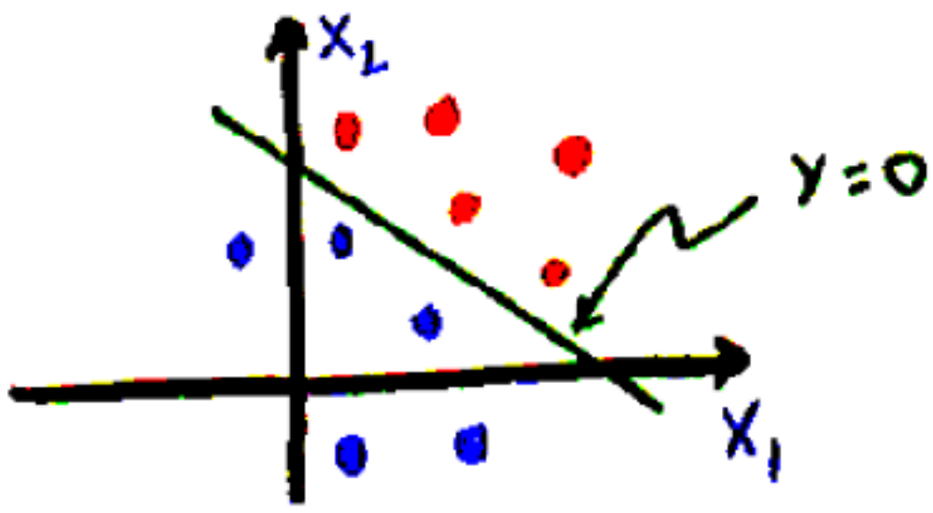


- pick a **machine** $G(W, X)$ parameterized by W. It can be complicated and non-linear, but it better be differentiable with respect to W.
- pick a **per-sample loss function** $L(Y, G(W, X))$.
- pick a training set
  $\mathcal{S} = (X^1, Y^1), (X^2, Y^2), ...(X^P, Y^P).$
- find the W that minimizes
  $\mathcal{L}(W, \mathcal{S}) = \frac{1}{P} \sum_i L(Y^i, G(W, X^i))$

Linear Classifier

Historically, the Linear Classifier was designed as a highly simplified model of the neuron (McCulloch and Pitts 1943, Rosenblatt 1957):





$$y = f(\sum_{i=0}^{i=N} w_i x_i)$$

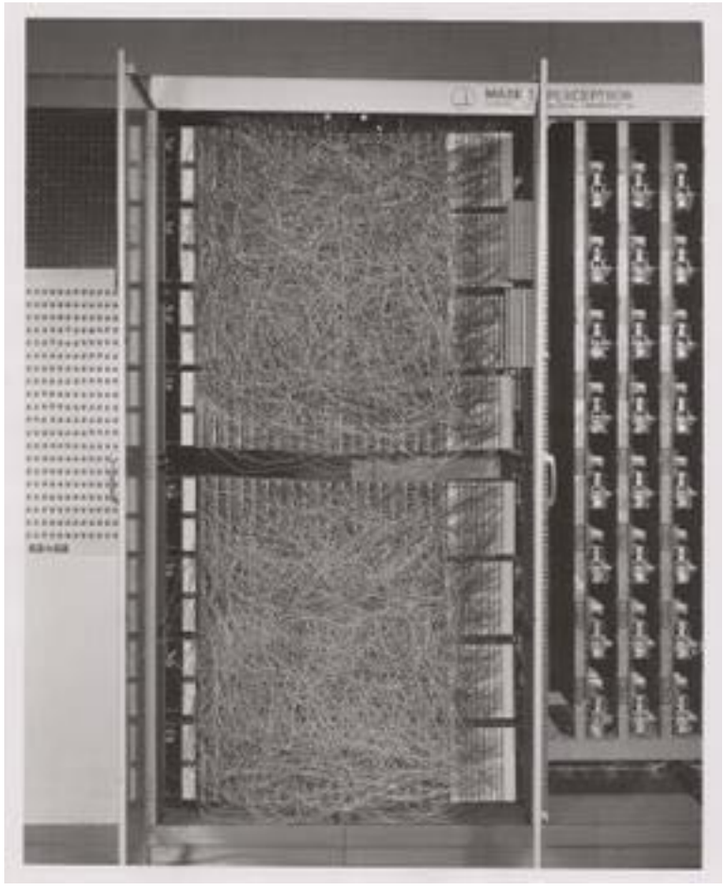
With f is the threshold function: $f(z) = 1$ iff $z > 0$, $f(z) = -1$ otherwise. x_0 is assumed to be constant equal to 1, and w_0 is interpreted as a bias.
In vector form: $W = (w_0, w_1 ....w_n), X = (1, x_1 ...x_n)$:

$$y = f(W'X)$$

The hyperplane $W'X = 0$ partitions the space in two categories. W is orthogonal to the hyperplane.

Perceptron

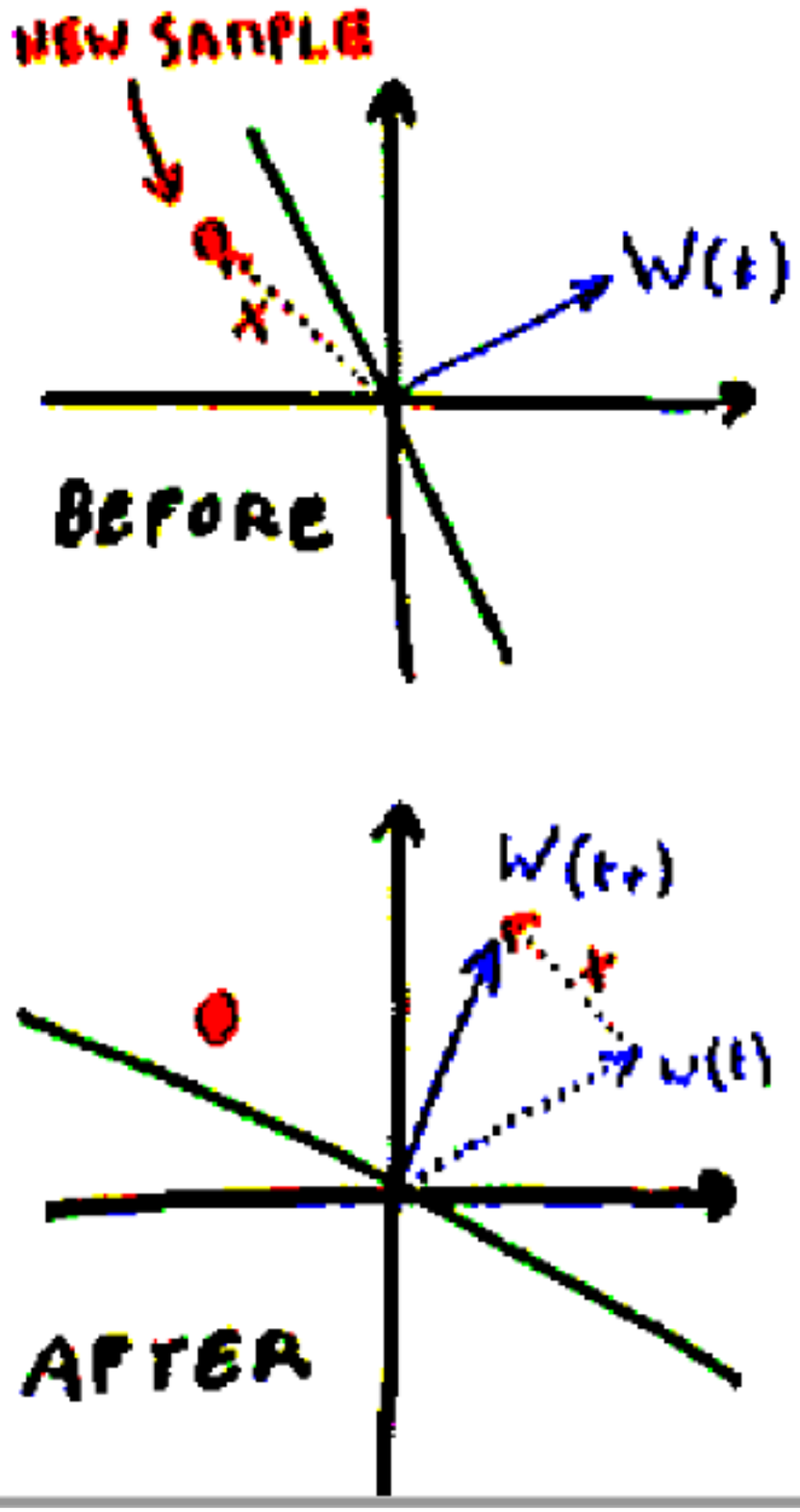
- A first model of a learning machine.
- The perceptron algorithm was invented in 1957 by Frank Rosenblatt.
- The Mark I Perceptron machine was the first implementation of the perceptron algorithm.



The New York Times reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

from Wikipedia

A simple idea: Error Correction



We have a **training set** $\mathcal{S}$consisting of P input-output pairs: $\mathcal{S} = (X^1, y^1), (X^2, y^2), ...(X^P, y^P)$.
A very simple algorithm:
- show each sample in sequence repetitively
- if the output is correct: do nothing
- if the output is -1 and the desired output +1: increase the weights whose inputs are positive, decrease the weights whose inputs are negative.
- if the output is +1 and the desired output -1: decrease the weights whose inputs are positive, increase the weights whose inputs are negative.

More formally, for sample p:

$$w_i(t+1) = w_i(t) + (y_i^p - f(W'X^p))x_i^p \cdot \frac{1}{2}$$

This simple algorithm is called the Perceptron learning procedure (Rosenblatt 1957).

The Perceptron Algorithm

Theorem: If the classes are linearly separable (i.e. separable by a hyperplane), then the Perceptron procedure will converge to a solution in a finite number of steps.
Proof: Let's denote by W^* a normalized vector in the direction of a solution. Suppose all X are within a ball of radius R. Without loss of generality, we replace all X^p whose y^p is -1 by $-X^p$, and set all y^p to 1. Let us now define the margin $M = min_p W^* X^p$. Each time there is an error, $W.W^*$ increases by at least $X.W^* \geq M$. This means $W_{final}.W^* \geq NM$ where N is the total number of weight updates (total number of errors). But, the change in square magnitude of W is bounded by the square magnitude of the current sample X^p, which is itself bounded by R^2. Therefore, $|W_{final}|^2 \leq NR^2$. combining the two inequalities $W_{final}.W^* \geq NM$ and $|W_{final}| \leq \sqrt{N}R$, we have

$$W_{final}.W^*/|W_{final}| \geq \sqrt{(N)}M/R$$

. Since the left hand side is upper bounded by 1, we deduce

$$N \leq R^2/M^2$$

The Perceptron Algorithm

Theorem: If the classes are linearly separable (i.e. separable by a hyperplane), then the Perceptron procedure will converge to a solution in a finite number of steps.

Good and Bad news

The perceptron learning procedure can learn a linear decision surface, **if such a surface exists** that separates the two classes. If no perfect solution exists, the perceptron procedure will keep wobbling around.

What class of problems is **Linearly Separable**, and learnable by a Perceptron?

There are many interesting applications where the data can be represented in a way that makes the classes (nearly) linearly separable: e.g. text classification using "bag of words" representations (e.g. for spam filtering).

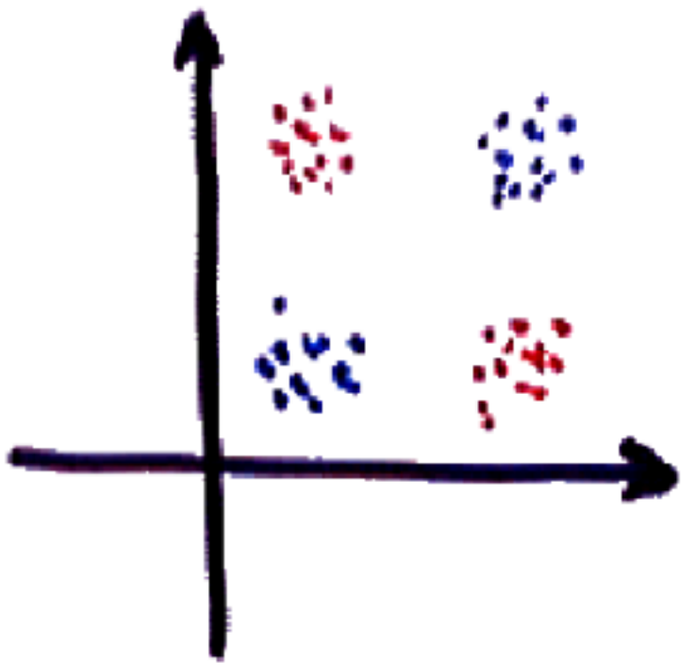
Unfortunately, the really interesting applications are generally not linearly separable. This is why most people abandonned the field between the late 60's and the early 80's.

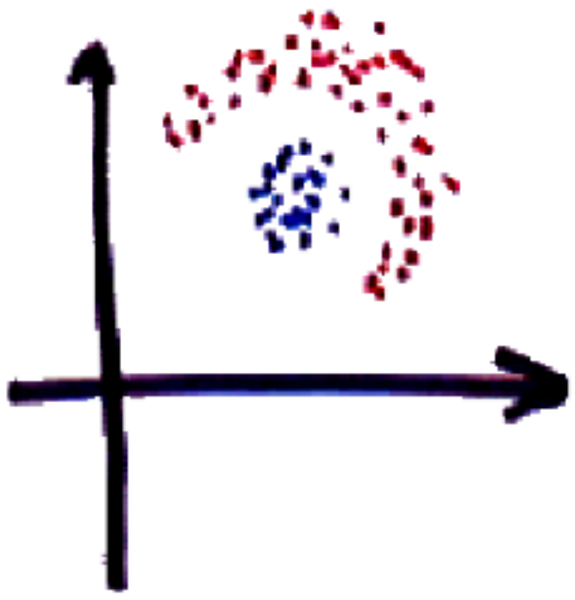
We will come back to the linear separability problem later.

Exercise

- Implement the Perceptron algorithm in your favorable programming language.
- Classify two-dimensional input data, and visualize how the perceptron algorithm learns the decision boundary.

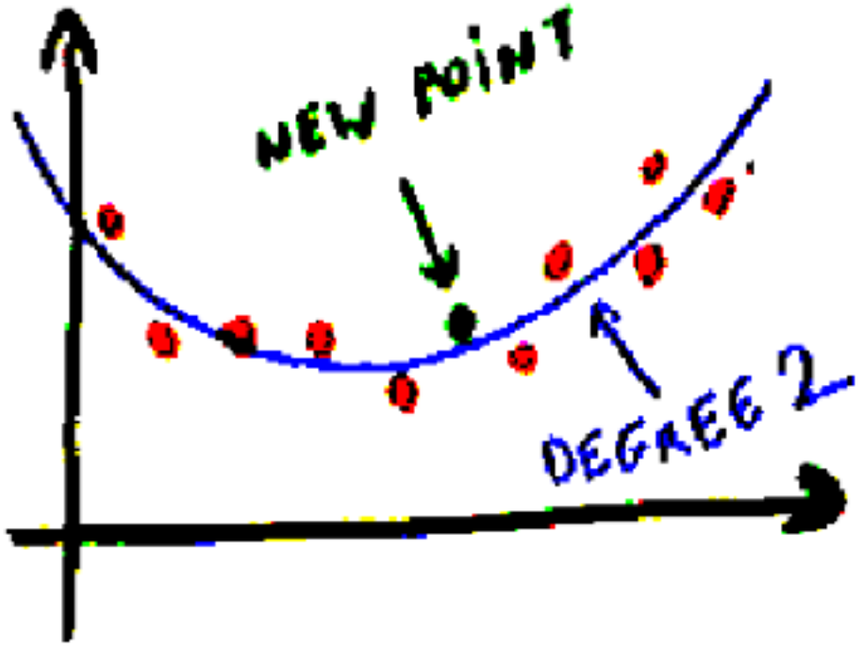
Limitations of linear machines

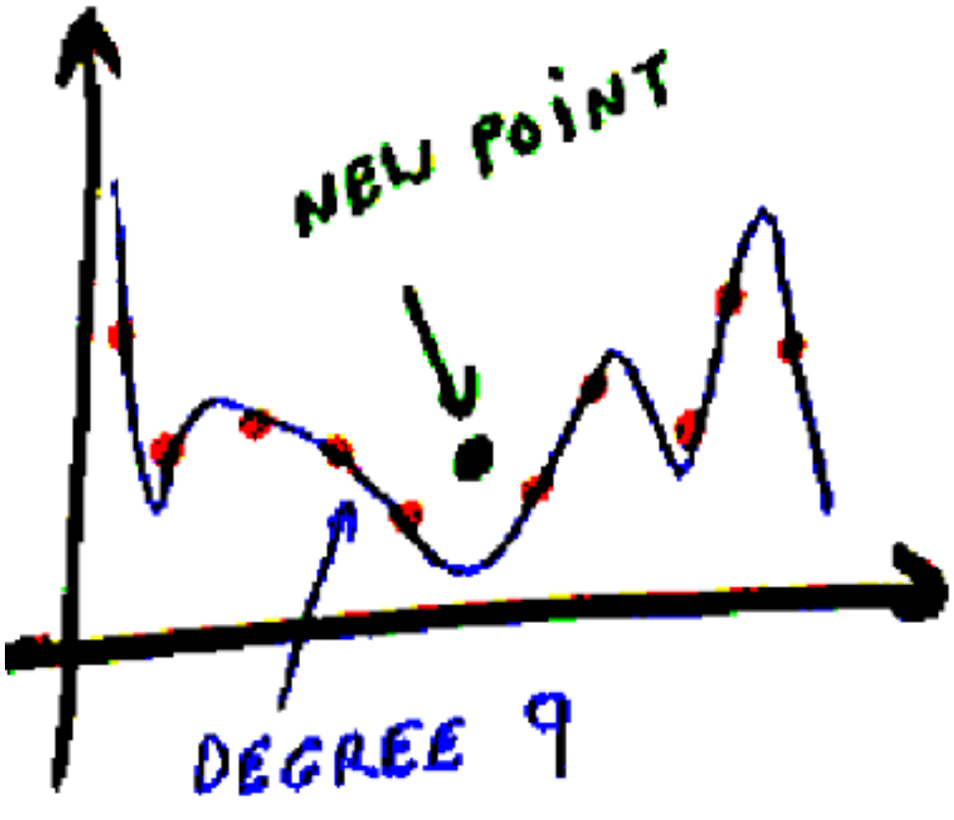




The *Linearly separable* dichotomies are the partitions that are realizable by a linear classifier (the boundary between the classes is a hyperplane).

Non-linear case and overfitting





What if we know that the process that generated our samples is non linear? We can use a richer **family of functions**, e.g. polynomials, sum of trigonometric functions....

PROBLEM: if the family of functions is too rich, we run the risk of **overfitting** the data. If the family is too restrictive we run the risk of not being able to approximate the training data very well.

QUESTIONS: How can we choose the richness of the family of functions? Can we predict the performance on new data as a function of the training error and the richness of the family of functions?

Simply minimizing the training error may not give us a solution that will do well on new data.

Statistical setting

- Training and test samples
  - In general, new inputs (test samples) are different from training samples.
- Generalization
  - The ability to produce correct outputs or behavior on previously unseen inputs
- Key question
  - how to get good generalization with a limited number of examples.

Training and test samples

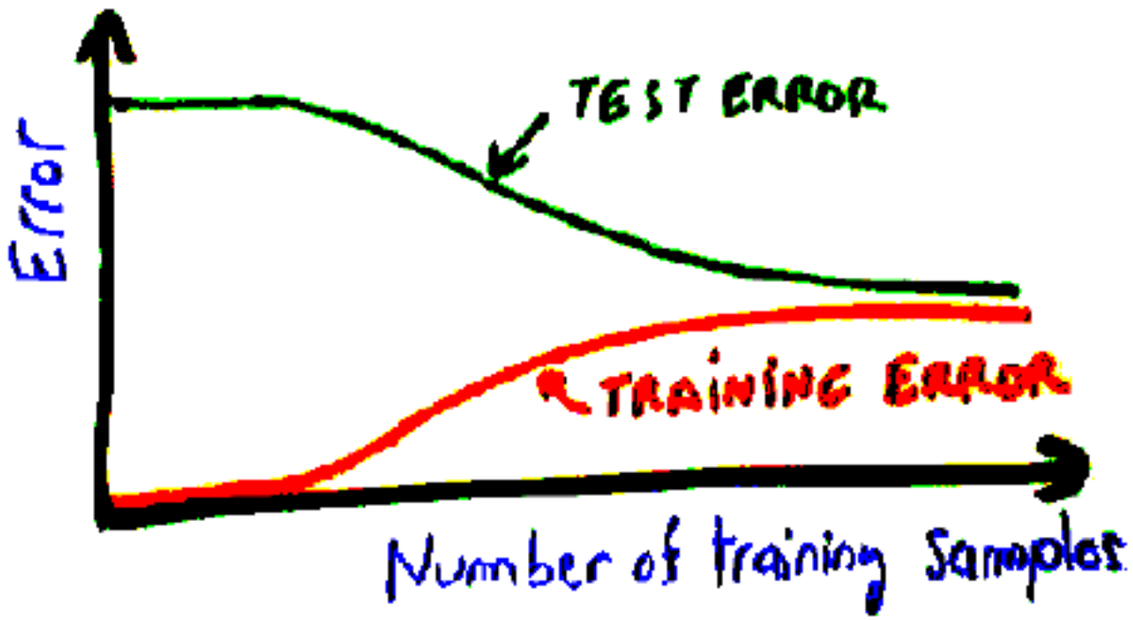
MNIST database of handwritten digits

“The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD–3 as their training set and SD–1 as their test set. However, SD–3 is much cleaner and easier to recognize than SD–1. The reason for this can be found on the fact that SD–3 was collected among Census Bureau employees, while SD–1 was collected among high–school students.

Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.”

http://yann.lecun.com/exdb/mnist/

Generalization



What we are **really** interested in is good performance on unseen data. In practice, we often partition the dataset into two subsets: a training set and a test set. We train the machine on the training set, and measure its performance on the test set.

The error on the training set (the average of the loss function) is often called the **emprical risk**. The average loss on an infinite test set drawn from the same source as the training set is often called the **expected risk**.
The number of training samples for which the training error and test error start converging toward each other is called the "capacity" of the learning machine (there are formal definitions for this that we will study later on).

The generalization can be measured by the difference between the empirical risk and the expected risk.

Machine Learning Framework

- A better theoretical framework for studying generalization, regularization and structural risk minimization is the so-called Statistical Learning Theory.
- What is learnable? in what sense? How?

Probably Approximately Correct (PAC) Learnable

- A concept c: $\mathcal{X} \rightarrow \{0, 1\}$
- $\mathcal{X}$ is the set of all possible examples
- $c \in \mathcal{C}$ a concept class

c_1 c_2

PAC Learnable

- Test (generalization) error to learn c from D using h:

$$R(h) = \mathbb{P}_{x \sim D}[h(x) \neq c(x)]$$

- Training (empirical) error based on samples $\{x_1, \cdots, x_n\}$:

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} 1_{h(x_i) \neq c(x_i)}$$

The training data $\{x_1, \cdots, x_n\}$ is a sequence of i.i.d. samples $\sim D$.

PAC Learnable

• A concept class $\mathcal{C}$ is PAC-learnable if the hypothesis h_n returned by the algorithm after observing a number of training samples
(polynomial in ϵ, δ, n and c) satisfies:

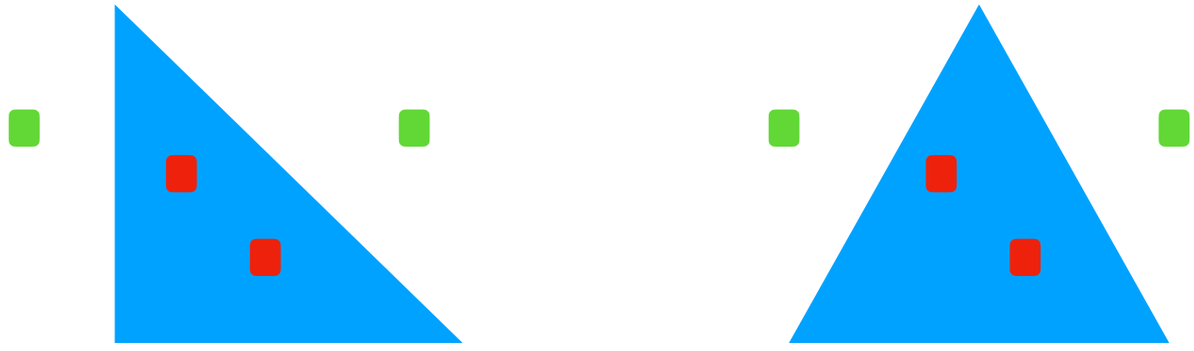
$$\mathbb{P}_{\{x_1,\cdots,x_n\}\sim D^n}[R(h_n) \leq \epsilon] \geq 1-\delta$$

$$R(h_n) = \mathbb{P}[h_n(x) \neq c(x)] = \mathbb{E}[1_{h_n(x)\neq c(x)}]$$

- This means approximately correct with high probability
- or probably approximately correct (PAC)

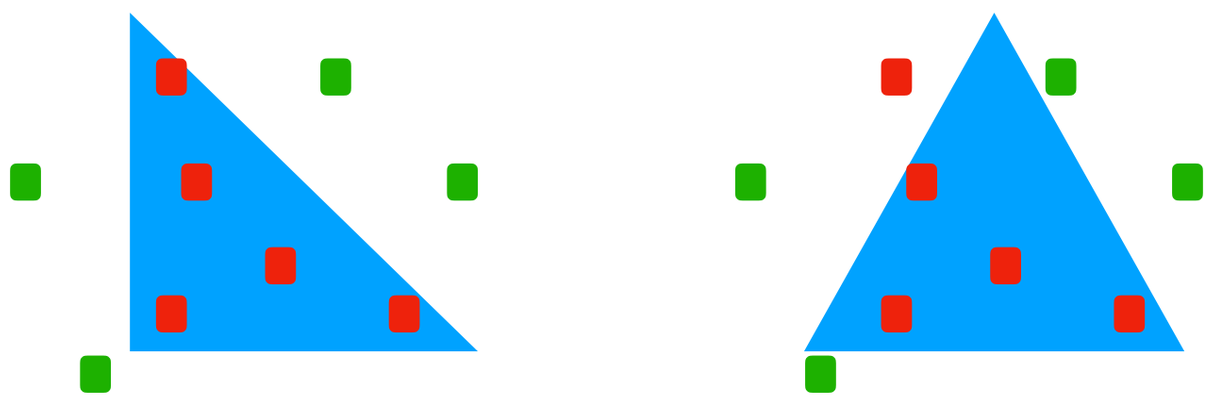
PAC Learnable

- When sample size n is small, we can not distinguish c_1 and c_2.



c_1 c_2

- As sample size n grows big, we can distinguish c_1 and c_2.



c_1 c_2

Generalization bound

- Using Hoeffding's inequality, we have

Fix a hypothesis $h : \mathcal{X} \to \{0, 1\}$, then for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$,

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{\log \frac{2}{\delta}}{2n}}$$

- Thus when n is large, training error is close to test error for a given h.
- What if we want to control all h?

Structural Risk minimization

- Uniform bound

Let H be a finite hypothesis set. Then for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$,

$$\max_{h \in H} [R(h) - \hat{R}(h)] \leq \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2n}}$$

- To control $R(h) - R(c)$, we rewrite this as

$$R(h) - \hat{R}(h) + \hat{R}(h) - R(c)$$

- R(c)>=0, we can try to control this by minimizing $\hat{R}(h) + \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2n}}$