Decision Tree Induction

Top-Down Induction of Decision Tree — TDIDT

- Error Measures Error function (loss function or cost function)

classification: Y is discrete, a small finite, unordered set of classes

error(h(x), f(x)) = 0 if h(x) = f(x)
else 1 (0-1 loss)

regression: Y is continuous, a numeric set (typically real numbers)

error(h(x), f(x)) = (h(x) − f(x))² (squared error)

training error: (经验风险 / train error)

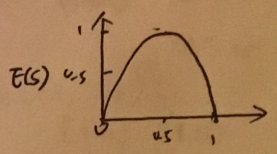
$error_E(h) := \frac{1}{|E|} * \sum_{e=(x,y)\in E} error(y, h(x))$

minimal training err does not entail minimal test err.

- TDIDT. Decision Tree

Entropy(S) = $-P_+ \log_2 P_+ - P_- \log_2 P_-$

$I(S)$ information measures the impurity of S



Gain(S, A) = expected reduction in entropy due to sorting on A

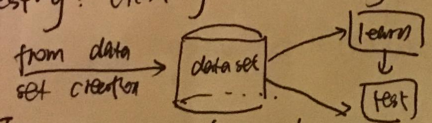
$= Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$

Handling numeric data by:

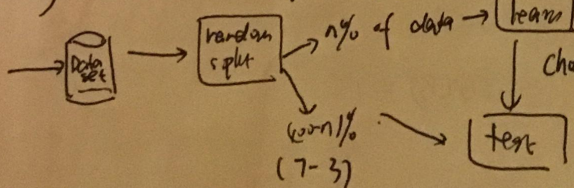
① expert preprocessing (专家预处理)
② automatic preclustering (把 cluster 看为 groups; 聚类预处理)
③ search out split time
(sort attributes, create n-1 binary attributes; compare and select best split using entropy/info Gain)

overfitting: errors is lower than it would be on independent test set

- Full set testing: (testing on training set, training error)

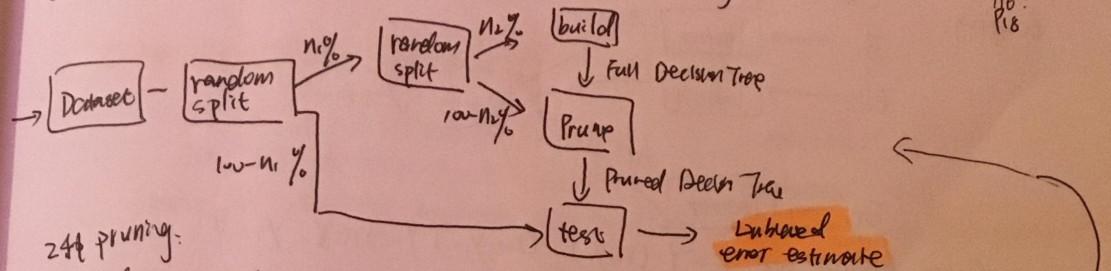


Estimating True error: out-of-sample test → unbiased estimator



Out-of-Sample Testing for Decision Tree Pruning

11-13 PPT P6? P18



24种 pruning:

① Reduced error pruning: { cutting a node, replaced by majority class; at each level, select tree with lowest sample error on the validation set; stop if next level doesn't reduce error anymore

② Rule post-pruning ← (C4.5)

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use.

Greedy algorithm
↙ $n \log n$ (n: #example)
computational complexity

Iterative Windowing: (example set E)

Sample := random_sample (E, N%)

T: TDIDT (sample), Err := errors (T, E)

while Err ≠ ∅, do "Progress"

- Sample := Sample ∪ random_sample (Err, M%)
- T := TDIDT (Sample)
- Err := errors (T, E)

return T

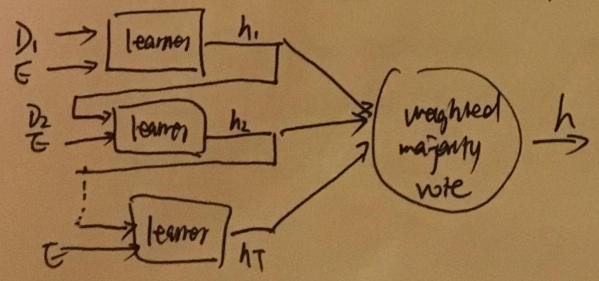
Boosting: — The basic algorithm

- Given training set $E = \{(x_1, y_1), \ldots (x_m, y_m)\}$
- $D_1(x_i) = 1/m$, $1 \le i \le m$
- Repeat for $t = 1, \ldots T$:
  - Learn hypothesis h_t on current weighted training set
  - Modify D_t so that incorrectly classified examples get emphasized
    - $\forall i$: $h_t(x_i) \neq y_i$: $D_{t+1}(x_i) > D_t(x_i)$ using a suitable update rule
    - $\forall i$: $h_t(x_i) = y_i$: $D_{t+1}(x_i) < D_t(x_i)$ using a su- - - - -
- Combine hypotheses by weighted majority vote into one single accurate rule

$h(x) := \text{argmax}_{y \in Y} \sum_{t=1 \ldots T} h_t(x) = y \; w_t(x)$

Bagging (Bootstrap aggregating): 随机选取 n 个训练样本，重复多次，预测函数为 $h_1, \ldots h_n$.
then. 最终 h 对分类采用投票方式.

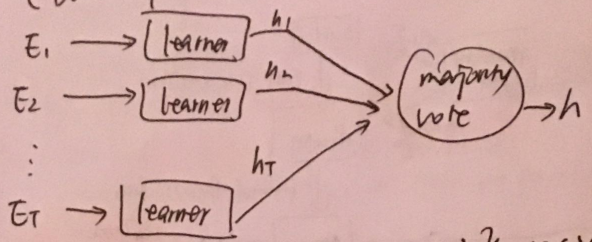
boosting: 初始时对每一个训练例 赋相等权重 $\frac{1}{n}$.
每次训练后，对训练失败的训练例赋于较大的权重，得到 $h_1, \ldots h_m$.
预测好的 权重大，反之则小. 最终 h 对分类采用 有权重投票.



样本点

sb. $\left(y_i\left(\langle w, x\rangle + b\right) - 1\right) \le 0$

Bagging. (Bootstrap Aggregation)



Given training set $E = \{(x_1, y_1) .. (x_m, y_m)\}$ $x_i \in X$, $y_i \in Y$

Repeat for $i = 1 \dots T$:

- Produce bootstrap replicates E_i of training set E by sampling with replacement s.t. $|E_i| = |E|$
- learn classifier h_i from replicate E_i

Output majority vote hypothesis H.

$$H(x) = \text{argmax}_{y \in Y} |\{i \in \{1 \dots T\} | h_i(x) = y\}|$$

Predictive Learning:
Function Approximation

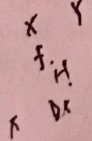
Assumed.: X → instance space (fixed distribution D_x)
Y → target space
function $f: X \to Y$
A set of allowed hypothesis H

Given (Input):
examples $E \subseteq X \times Y$ such that
- x drawn iid from D_x
- $\forall$ "labeled" $e(x,y) \in E$, $f(x) = y$

Find (Output):
A hypothesis $h \in H$ such that the true error of h

$$\text{error}_{D_x}(h) := E_{D_x}(\text{error}(f(x), h(x)))$$

expected (avg) classification error on instances drawn according to D_x is minimal



$$\text{error}_{D_x}(h) = E_{D_x}[\text{error}(f(x), h(x))]$$

Concept learning Generality

- **Consistent hypothesis finding problem**

  For all $e^+ = (x, y) \in E^+$, $h(x) = +1$ — "completeness" of h
  For all $e^- = (x, y) \in E^-$, $h(x) = -1$ — "correctness" of h
  Finding a consistent hypothesis means finding a hypothesis with **training error zero**

- Covers Relation

  For an example $e = (x, y)$, we say
  - h covers e iff $h(x) = +1$ (h classified e as positive)
    - if $y = +1$. — true positive
    - if $y = -1$, — false positive
  - h doesn't cover e iff $h(x) = -1$ (h classified e as negative)
    - if $y = -1$. — true neg
    - if $y = +1$, — false neg

- Generality.

  A hypothesis (concept) h_1 is more general (less specific) than hypothesis (concept) h_2 iff h_1 covers a superset of instances of h_2:

$$h_1 \leq h_2 \text{ iff } \{x \in X \mid h_1 \text{ covers } x\} \geq \{x \in X \mid h_2 \text{ covers } x\}$$

Most specific hypothesis $h_{ms} = \langle \phi, \phi, \phi, \phi \rangle$
Most general hypothesis $h_{mg} = \langle ?, ?, ?, ? \rangle$

$$h_1^{[func]} \leq h_2^{[func]} \text{ iff } h_1^{[rule]} \models h_2^{[rule]} \text{ iff } h_2^{[pred]} \models h_1^{[pred]}$$

P17.18 -11-20

Find-S Algorithm.

1. Initialize h to the most specific hypothesis in H.
2. For each positive training instance x.
   - For each attribute constraint a_i in h
     If the constraint a_i in h is satisfied by x
     Then do nothing
     Else replace a_i in h by the next more general constraint that is satisfied by x

   (minimal generalization)
3. Output hypothesis h

Find-S disadvantages:

- Can't tell whether it has learned concept
- Can't tell when training data inconsistent
- Picks a maximally specific h.
- Depending on H, there might be several

- Version Spaces

D: training examples target concept → c

Consistent(h, D) := h(x) = c(x) for all (x, c(x)) in D

Version Space $VS_{H,D}$.

$VS_{H,D} \equiv \{h \in H \mid Consistent(h, D)\}$

↳ with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with all training examples in D

- List-Then-Eliminate Algorithm

1. Version space ← a list containing hypotheses in H
2. For each training example, ⟨x, c(x)⟩
   remove from Version Space any hypothesis for which $h(x) \neq c(x)$
3. Output the list of hypotheses in Version Space

- General bound G. of $VS_{H,D}$ is the set of its maximally general members

">": general $G \equiv \{g \in H \mid Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$

Specific bound S, ... - - - - - specific ...

$S \equiv \{s \in H \mid consistent(s, D) \wedge (\neg \exists s \in H)[(s >_g s') \wedge Consistent(s', D)]\}$

Every member of $VS_{H,D}$ lies between these boundaries

$VS_{H,D} = \{h \in H \mid (\exists s \in S)(\exists g \in G).\ g \leq h \leq s\}$

- Candidate Elimination Algorithm.

G. S.

For each training example d, do

- if d is "+"
  - remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - remove s from S
    - Add to S all minimal generalization h of s such that
      1. h is consistent with d
      2. some members of G is more-general-than h
    - remove from S any hypothesis that is more general than another hypothesis in S
- if d is "−"
  - remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - remove g from G
    - Add to G all minimal specializations h of g such that
      1. h is consistent with d
      2. some members of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

Def. Let h_j, h_k be boolean-valued function defined over X.
Then, h_j is more_general_than_or_equal_to h_k ($h_j \geq_g h_k$) if and only if
$(\forall x \in X)[(h_k(x)=1) \to (h_j(x)=1)]$

P37?

Inductive Bias

concept learning algorithm L
instances X, target concept c.
training examples $D_c = \{<x, c(x)>\}$
$L(x_i, D_c)$ denotes the classification ~~of~~ assigned to the instance x_i by L training on data D_c

Def. the inductive bias of L is any minimal set of assertions B 断言 such that for any target concept c and corresponding training examples D_c
$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$

$A \vdash B$ means A logically entails B
B为A 演绎派生 (follow deductively, 故 B 可以由 A 证明得出)
$A \succ B$ 表示 B 从A 归纳推理得到

有偏程度 Rote learner < version space candidate.. < Finds

习题 44 Candidate 的归纳偏置: 目标概念 c 包含在给定的假设空间 H 中

TMY P13 记下
(1) 用出 伪代码

---

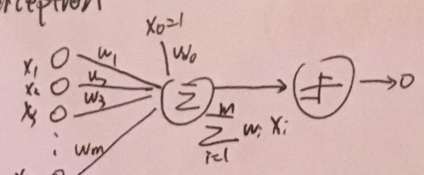
correctness: G, S. $VS_{H,D} == Output$
1) soundness $VS_{H,D} \subseteq Output$
2) completness $Output \in VS_{H,D}$
Proof $D = \{e_1, \dots e_n\}$
Math induction 数学归纳法
1) $D = \emptyset$. $G_0 = \{most\ general\}$ $S_0 = \{most\ specific\}$
$VS_{H,D} = H == H$
2) $D_i = \{e_1, \dots e_i\}$ $VS_{H,D} == output$
3) $D_i + \{e_{i+1}\}$
3. 1) e_{i+1} is positive
G_i and S_i
check G_i for consistency }
check S_i for consistency }
3.2) e_{i+1} is negative #
G_{i+1} and S_{i+1} == $VS_{H, D_{i+1}}$

suppose that are not consistent with e_{i+1}
for all $s \in S_i$, that don't cover e_{i+1},
remove s from S_i,
add to S_i all generalizations h. s.t
a) h cover e_{i+1}
b) $\exists g \in G_{i+1}$, $g < h$
c) $\nexists h'$ $h < h' < S$
not satisfy e_{i+1}

S
V ↑ h
V ↑
G

Neural Network

- Perceptron



$$o(x_1, \dots x_m) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_i x_i > 0 \\ -1 & \text{o.w} \end{cases}$$

perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta (t - o) x_i$$

(η: learning rate)

- Gradient Descent:

$$o = w_0 + w_1 x_1 + \dots + w_m x_m$$

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$ D: set of examples

Gradient: $\nabla E[\vec{w}] = \left[\frac{\partial w}{\partial w_0}, \frac{\partial w}{\partial w_1}, \dots \frac{\partial E}{\partial w_m} \right]$

training rule: $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$ $\left(\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \right)$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 = \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2 (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d})$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

- Sigma function: sigmoid unit

$$\sigma(x) = \frac{1}{1 + e^{-x}} \Rightarrow (0, 1)$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}} \Rightarrow (-1, 1)$$

$$\frac{d}{dx} \tanh = 1 - \tanh^2 x$$

with multiple layers: ① arbitary decision boundaries
② including non-convex, disconnected area

---

error gradient:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2 (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \left(- \frac{\partial o_d}{\partial w_i} \right)$$

$$= - \sum_d (t_d - o_d) \underbrace{\frac{\partial o_d}{\partial net_d}}_{①} \underbrace{\frac{\partial net_d}{\partial w_i}}_{②}$$

① $\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d (1 - o_d)$

② $\frac{\partial net_d}{\partial w_i} = x_{i,d}$

$$\therefore \frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

- Error function with multiple output units

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$$

反向传播算法的偏置
smooth interpolation 插值
between data points

Backpropagation Algorithm

Initialize all weights to small random numbers

Until satisfied, Do

- for each training example, do

1. Input the training example to the network and compute the network outputs
2. For each output unit k: → d'(sigmoid function) = $f'(x) = \frac{d}{dx}f(x) = f'(net)$

$$\delta_k \leftarrow O_k(1-O_k)(t_k - O_k)$$

3. For each hidden unit h:

$$\delta_h \leftarrow O_h(1-O_h)\sum_{k\in outputs} w_{h,k}\delta_k$$

$\frac{d}{dx}f(O_k) = f'(net)$

4. Update each network weight $w_{i,j}$:

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}, \quad \text{where } \Delta w_{i,j} = \eta\,\delta_j\,x_{i,j}$$

O_k: actual output of a node k

$x_{i,j}$: input from node i to node j

- Every boolean function can be ~~express~~ represented by network with a single hidden layer, but require exponential hidden units
- Every ~~bounded~~ continuous function can be approximated with arbitrary small error with one hidden layer.
- Any function can be approximately to arbitrary accuracy by a network with ~~two~~ hidden layer.

Random weight initialization ✓✓

- necessary
- reduce the risk of the network getting trapped in a local minimum
- not too large to avoid hitting the saturation region of sigma-function 饱和
  not too small to not reduce the network to a linear network

Gradient descent Algorithm

linear sigma network perceptron

Gradient-descent (training_examples, η)

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero
  - For each $\langle \vec{x}, t\rangle$ in training_examples, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t-o)x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

BP 对 output layer hidden layer 的对比

o output

Δw

Bayes Optimal Classifier: 给定训练数据，对新实例最可能的分类是什么?

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i) P(h_i|D)$$

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i) P(h_i|D)$$

V: 所有分类的集合

Naive Bayes Classifier:

Assume target function $f: X \rightarrow V$, where each instance x described by attributes $\langle a_1, a_2, \ldots a_n \rangle$.

Most probable value of $f(x)$ is:

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j | a_1, \ldots a_n)$$

$$= \arg\max_{v_j \in V} \frac{P(a_1, \ldots a_n | v_j) P(v_j)}{P(a_1, \ldots a_n)}$$

$$= \arg\max_{v_j \in V} P(a_1, \ldots a_n | v_j) P(v_j)$$

$$\prod_i P(a_i|v_j)$$

naive bayes classifier: $v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j)$ → attribute

[illegible] example 里面有

NB Algorithm:

Naive_Bayes_Learn (examples)

For each target value v_j

$\tilde{P}(v_j) \leftarrow$ estimate $P(v_j)$

For each attribute value a_i of each attribute a

$\tilde{P}(a_i|v_j) \leftarrow$ estimate $P(a_i|v_j)$

Classify_New_Instance (x)

$$v_{NB} = \arg\max_{v_j \in V} \tilde{P}(v_j) \prod_i \tilde{P}(a_i|v_j)$$

if none of training instances with target value v_j have attribute a_i?. Then $\tilde{P}(a_i|v_j) = 0 \ldots$ $P(v_j) \prod_i P(a_i|v_j) = 0$

so, we use

$$\tilde{P}(a_i|v_j) \leftarrow \frac{n_c + mp}{n + m}$$

n: number of training examples for which $v = v_j$

n_c: number of examples for which $v = v_j$ and $a = a_i$

p: prior estimate for $\tilde{P}(a_i|v_j)$ $= \frac{1}{m}$

m: weight given to prior (i.e. number of virtual examples)

Bayesian Belief Networks 拓扑一般表示一些概率分布

↳ describe conditional independence among subsets of variables

→ allow combining prior knowledge about independences among variables with observed training data

Def: X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given the value of Z: that is, if

$$(\forall x_i, y_j, z_k)\ P(X=x_i \mid Y=y_j, Z=z_k) = P(X=x_i \mid Z=z_k)$$

即

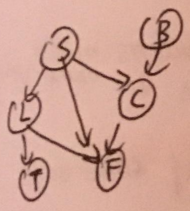
$$P(X \mid Y, Z) = P(X \mid Z)$$

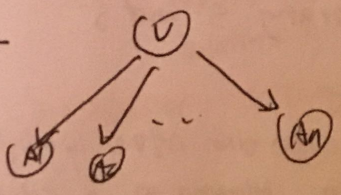
BNB use: $P(X, Y \mid Z) = P(X \mid Y, Z) \cdot P(Y \mid Z)$

证明: $= P(X \mid Z) \cdot P(Y \mid Z)$

$$P(y_1, y_2, \ldots y_n) = \prod_{i=1}^{n} P(y_i \mid \text{Parents}(Y_i))$$

where Parents(Y_i) denotes immediate predecessors of Y_i in graph.





Kernel Method: a class of pattern detecting algorithms

domain independence: generic algorithms parameterized by kernels

computational efficiency: runtime - polynomial

robustness: handle noise data, identify good approximations

statistical stability:

high-dimension ← (i) Feature space (data is embedded into an inner product space)

(ii) Patterns: linear functions in the feature space

(iii) pairwise inner product ← can be computed by a function → Kernel

(iv)

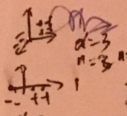
Kernel methods: a class of pattern detecting algorithms

Key Aspects

(i) data is embedded into a inner product space ⇒ feature space (high-dim)

(ii) Patterns: linear functions in the feature space (usually corresponed to non-linear functions in the original space)

Def. A dichotomy (S^+, S^-) of $S \subseteq \mathbb{R}^d$ is linearly separable if there exists a $(d-1)$-dim affine hyperplane separating S^+ and S^-.



Def. A set $S \subseteq \mathbb{R}^d$ of n points are in general position if

(i) $n \le d$ and S is not contained by an $(n-2)$-dim affine hyperplane

(ii) $n > d$ and S has no subset of $d+1$ points belonging to a $(d-1)$ dim affine hyperplane.

Cover's (Function Counting) Theorem:

The number of linearly separable dichotomies of n points in general position in $\mathbb{R}^d$ is

$$C(n,d) = 2\sum_{k=0}^{d} \binom{n-1}{k} \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Probability of Linear Separability:

$P(n,d)$: Prob of (A dichotomy selected uniformly at random from a set of n points in general position in $\mathbb{R}^d$ is linearly separable)

$$P(n,d) = \frac{1}{2^n} C(n,d) = \frac{1}{2^n} 2\sum_{k=0}^{d} \binom{n-1}{k}$$

Implication of Cover's Theorem:

If training data is not linearly separable, with high probability it can be embedded into a higher dimension space via some non-linear transformation in which it becomes linearly separable.

if $n \le d+1$: $C(n,d) = 2\sum_{k=0}^{n-1} \binom{n-1}{k} = 2^n \Rightarrow P(n,d) = 1$

if $n = 2(d+1)$, $P(n,d) = \frac{1}{2^{2(d+1)}} \cdot 2\sum_{k=0}^{d} \binom{2d+1}{k} = \frac{1}{2}$

if $n \gg d$, $P(n,d) \to \frac{Cn^d}{2^n}$ for some constant $C > 0$

$$P(n,d) = \begin{cases} 1 & \text{if } n \le d+1 \\ \frac{1}{2} & \text{if } n = 2(d+1) \\ 0\left(\frac{n^d}{2^n}\right) & \text{if } n \gg d \end{cases}$$

So, for any $\varepsilon > 0$

$\lim_{d\to\infty} P(2(d+1)(1-\varepsilon), d) = 1$

$P(2(d+1), d) = \frac{1}{2}$ phase transition at $n = 2(d+1)$

$\lim_{d\to\infty} P(2(d+1)(1+\varepsilon), d) = 0$

the learning problem:

Given: • a set of training data $S = \{(x_1, y_1), \dots (x_n, y_n)\} \subseteq X \times Y$ generated by a probability distribution $P(x, y)$, and
• a set of $H = \{f \mid f: X \to Y\}$ of functions, → hypothesis space.

Find a function $f_S \in H$ minimizing the true error.

$$R[f] = \int_{X \times Y} V(f(x), y)\, dP(x, y)$$

loss function

Empirical Risk Minimization:

empirical error: $R_{emp}[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i)$

empirical risk minimization: $\arg\min_{f \in H} R_{emp}[f]$ (select a h that minimizes the training error)

• A problem is well-posed if it satisfies:
1. existence: the problem has a solution
2. uniqueness: the solution must be unique
3. stability: the s.t. depends continuously on the data

o.w. is ill-posed

deblur ⇒ unstable

• Regularization theory:
↓ method improving stability of solutions of ill-posed inverse problems

Tikhonov regularization: transforms the original problem into a conditional-optimization problem ~~into a constrained optimization problem~~ that penalizes undesirable solutions by adding a term, called regularizer.

regularizer: enforce the solution to be smooth.

↓

Ω

$$\arg\min_{f \in H} R_{emp}[f] + \lambda \Omega[f]$$

$\lambda \in \mathbb{R}$: regularization parameter controlling trade-off between
(i) minimizing the empirical risk
(ii) minimizing the model complexity

• Ridge Regression:

Given a sample $S = \{(\vec{x}_1, y_1) \dots (\vec{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$, find a linear function $f(\vec{x}) = \vec{x}^T \vec{w} = \sum_{i=1}^{d} x_i w_i$ that best interpolates S.

equals: find a vector $\vec{w} \in \mathbb{R}^d$ $f(\vec{x}) = \underbrace{\langle \vec{x}, \vec{w} \rangle}_{\text{inner product}} = \vec{x}^T \vec{w} = \sum_{i=1}^{d} x_i w_i$

"best interpolates" for $(\vec{x}, y)$, $V(f(\vec{x}), y) = (f(\vec{x}) - y)^2$

empirical risk: $R_{emp}[f] = \frac{1}{n} \sum_{i=1}^{n} (f(\vec{x}_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (\langle \vec{x}_i, \vec{w} \rangle - y_i)^2$

$$= \frac{1}{n} (X\vec{w} - \vec{y})^T (X\vec{w} - \vec{y}) = \frac{1}{n} \|X\vec{w} - \vec{y}\|^2$$

$$\min_{\vec{w}} \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 + \lambda \|\vec{w}\|^2$$

$$= \min_{\vec{w}} R_\lambda(\vec{w}, S) = \min_{\vec{w}} \|X\vec{w} - \vec{y}\|^2 + \lambda \|\vec{w}\|^2 \text{ is convex in } \vec{w}$$

Curse of dimensionality:
(i) potentially enormous time complexity of computing inner products
(ii) overfitting

Kernel methods solve:
(i) Kernel trick (inner product can be calculated without computing the embedding)
(ii) Regularization (automatic control of overfitting)

Ridge Regression

dual solution $\vec{\alpha} = \frac{1}{\lambda}(\vec{y} - X\vec{w})$
$\vec{w} = X^T\vec{\alpha}$

primal	dual
$\vec{w} = (X^TX + \lambda I_d)^{-1}X^T\vec{y}$	$\vec{\alpha} = (XX^T + \lambda I_n)^{-1}\vec{y}$
$f(\vec{x}) = \vec{x}^T\vec{w}$	$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \langle \vec{x}, \vec{x}_i \rangle$

Complexity:
- computation of $\vec{w}$: $O(d^3)$
- evaluation of f: $O(d)$
- computation of $\vec{\alpha}$: $O(n^3)$
- evaluation of f: $O(nd)$

dual is faster for the first step if $d > n$, but evaluation is always slower.

crucial ~~obs~~

About the dual, matrix $G = XX^T \Rightarrow$ Gram matrix, is given by inner products of the training points,

$$G_{i,j} = \langle \vec{x}_i, \vec{x}_j \rangle$$

Kernel Trick:

Learning in Feature Space

worked example: dual of ridge regression
(i) $\vec{\alpha} = (G + \lambda I_n)^{-1}\vec{y}$ with $G_{ij} = \langle \vec{x}_i, \vec{x}_j \rangle$
(ii) $f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \langle \vec{x}, \vec{x}_i \rangle$

transformation: $\phi: \mathbb{R}^d \to \mathbb{R}^D$

then, work automatically in feature space:
$\Rightarrow$ taking algorithm $\mathcal{A}$ with para $G, \lambda, \& \vec{y}$
(1) replace all entries $G_{ij} = \langle \vec{x}_i, \vec{x}_j \rangle$ by $\langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$ and solve (i) by running $\mathcal{A}$ with paras the new Gram matrix λ and $\vec{y}$
(2) for (ii), use $f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \langle \phi(\vec{x}), \phi(\vec{x}_i) \rangle$

Inner Product (Pre-Hilbert) Spaces

Def. a vector space H is an inner product space if it is equipped with a function $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{R}$

satisfying
$\langle \vec{x}, \vec{y} \rangle = \langle \vec{y}, \vec{x} \rangle$
$\langle \vec{x}, \vec{x} \rangle \geq 0$ with $\langle \vec{x}, \vec{x} \rangle = 0$ iff $\vec{x} = \vec{0}$
$\langle \lambda_1 \vec{x} + \lambda_2 \vec{y}, \vec{z} \rangle = \lambda_1 \langle \vec{x}, \vec{z} \rangle + \lambda_2 \langle \vec{y}, \vec{z} \rangle$
for all $\vec{x}, \vec{y} \in H$ and scalars λ_1, λ_2

the standard inner product in $\mathbb{R}^d$, i.e

$$\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{d} x_i y_i$$

$$\langle \lambda_1 \vec{x} + \lambda_2 \vec{y}, \vec{z} \rangle = \sum_{i=1}^{d} (\lambda_1 x_i + \lambda_2 y_i) z_i$$
$$= \sum_{i=1}^{d} \lambda_1 x_i z_i + \sum_{i=1}^{d} \lambda_2 y_i z_i$$
$$= \lambda_1 \langle x, z \rangle +$$

The Kernel Trick.

Def. a **kernel** is a function $X \times X \to \mathbb{R}$ such that for all $x, y \in X$

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

for some function ϕ mapping X to an inner product feature space $\mathcal{H}$.

Kernel trick: substitute all occurrences of $\langle \cdot, \cdot \rangle$ by a kernel k with

$$k(x,y) = \langle \phi(x), \phi(y) \rangle$$

where ϕ is the underlying function mapping the input space into the feature space.

example. let $k: X \times X \to \mathbb{R}$ with $X \subseteq \mathbb{R}^d$ be defined by

$$k(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle^2 \text{ for all } \vec{x}, \vec{y} \in X$$

claim: k is a kernel corresponding to the feature map $\phi: \mathbb{R}^d \to \mathbb{R}^{d^2}$ defined by

$$\phi: \vec{z} \longmapsto (z_i z_j)_{i,j=1}^{d} \text{ for all } \vec{z} \in \mathbb{R}^d$$

Proof

$$\langle \phi(\vec{x}), \phi(\vec{y}) \rangle = \langle (x_i x_j)_{i,j=1}^{d}, (y_i y_j)_{i,j=1}^{d} \rangle$$

$$= \sum_{i,j=1}^{d} x_i x_j y_i y_j = \sum_{i=1}^{d} x_i y_i \sum_{j=1}^{d} x_j y_j$$

$$= \langle \vec{x}, \vec{y} \rangle^2$$

Normed linear space:

Def: a vector space $\mathcal{H}$ is a **normed linear space**, if there exists a function $\|\cdot\| : \mathcal{H} \to \mathbb{R}^+$ satisfying

$$\|\vec{x}\| \geq 0 \text{ with } \|\vec{x}\| = 0 \text{ iff } \vec{x} = \vec{0}$$

$$\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$$

$$\|\lambda \vec{x}\| = |\lambda| \cdot \|\vec{x}\| \text{ for all } \vec{x}, \vec{y} \in \mathcal{H} \text{ and scalar } \lambda$$

Def. for a normed linear space $\mathcal{H}$, the function $d(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|$ defines a distance on $\mathcal{H}$.

eigenvalue 不等, 不用同已知

Properties of Inner Products

$$\|\vec{x}\| = \sqrt{\langle \vec{x}, \vec{x} \rangle}$$ (inner product space $\mathcal{H}$ is a normed linear space)

Cauchy-Schwarz inequality: for any $\vec{x}, \vec{y} \in \mathcal{H}$.

$$|\langle x, y \rangle| \leq \|\vec{x}\| \cdot \|\vec{y}\|$$

$$\Rightarrow \begin{cases} -1 \leq \frac{\langle \vec{x}, \vec{y} \rangle}{\|\vec{x}\| \|\vec{y}\|} \leq 1 \\ \cos\theta = \frac{\langle \vec{x}, \vec{y} \rangle}{\|\vec{x}\| \|\vec{y}\|} \end{cases}$$

Properties of Kernels: Let $k: X \times X \to \mathbb{R}$ be a kernel function with $k(x,y) = \langle \phi(x), \phi(y) \rangle$, for ϕ, and ref

$$\alpha(x, y) = \frac{k(x,y)}{\sqrt{k(x,x)}\sqrt{k(y,y)}}$$

then $$\alpha(x,y) = \frac{\langle \phi(x), \phi(y) \rangle}{\sqrt{\langle \phi(x), \phi(x) \rangle}\sqrt{\langle \phi(y), \phi(y) \rangle}} = \cos(\phi(x), \phi(y))$$

$\Rightarrow$ normalized kernels = cosine similarity in the feature space

Characterization of Kernels: Finite Domain

($A^{(n\times n)}$ is positive semi-definite, iff all of its eigenvalues are non-negative)

Gram / Kernel matrix $K_{X,k}$ of X, w.r.t k is $n\times n$ matrix, $(K_{X,k})_{ij} = k(x_i, x_j)$

Thm. (finite case): Let $k: X\times X \to \mathbb{R}$ be a symmetric function ($k(x,y)=k(y,x)$). Then

k is a kernel $\Leftrightarrow$ $K_{X,k}$ is positive semi-definite

example: $k: \{0,1,2\}\times\{0,1,2\}\to\mathbb{R}$ defined by $K_{X,k} = U\Lambda U^T$

$$k(x,y) = \begin{cases} 1 & \text{if } |x-y|\le 1 \\ 0 & \text{o.w} \end{cases}$$

is not a kernel. because:

$$K_{X,k} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

it has negative eig. $\Rightarrow$ is not positive semi-definite

Characterization of Kernels: Infinite Domain

Let X be a compact subset of $\mathbb{R}^n$ and let $k: X\times X \to \mathbb{R}$ be a symmetric and continuous function, then:

k is a kernel $\Leftrightarrow$ for all finite $Y\subseteq X$: $K_{Y,k}$ is positive semi-definite

$k(x,y) = \langle \phi(x), \phi(y)\rangle$

Kernel Construction

(i) **$k(x,y) = f(x)f(y)$** is a kernel over $X\times X$ for all functions $f: X\to\mathbb{R}$

Proof, let $\phi: X\to\mathbb{R}$ defined by $\phi: x \mapsto f(x)$ for all $x\in X$

$$\Rightarrow k(x,y) = f(x)f(y) = \langle \phi(x), \phi(y)\rangle$$

(ii) Let A be a symmetric positive semi-definite $n\times n$ matrix, then

$k(\vec{x},\vec{y}) = \vec{x}^T A \vec{y}$ for all $\vec{x},\vec{y}\in\mathbb{R}^n$ is a kernel over $\mathbb{R}^n\times\mathbb{R}^n$

Proof. by spectral theorem, $A = U^T\Lambda U$ (U: orthogonal matrix, Λ: diagonal matrix)

- all entries in Λ are non-negative, as A is positive semi-definite

$\Rightarrow$ we can take $\sqrt{\Lambda}$ (matrix with the square roots of the eigenvalues)

$\Rightarrow$ for $B = \sqrt{\Lambda}U$, $\phi: \vec{x}\mapsto B\vec{x}$,

$$k(\vec{x},\vec{y}) = \vec{x}^T A\vec{y} = \vec{x}^T U^T\Lambda U\vec{y} = \vec{x}^T B^T B\vec{y} = \langle B\vec{x}, B\vec{y}\rangle = \langle\phi(\vec{x}),\phi(\vec{y})\rangle$$

(iii) Let k_1, k_2 be kernels over $X\times X$. Then for all $\alpha,\beta\ge 0$

$k(x,y) = \alpha k_1(x,y) + \beta k_2(x,y)$ is a kernel

Proof. for all finite $Y = \{x_1,\dots x_n\}\subseteq X$ and for all $\vec{a}\in\mathbb{R}^n$

$\vec{a}^T K_{Y,k_1}\vec{a}\ge 0$ and $\vec{a}^T K_{Y,k_2}\vec{a}\ge 0$

Mercer's theorem ↗ it suffices to show that

$$\vec{a}^T(\alpha K_{Y,k_1} + \beta K_{Y,k_2})\vec{a} = \vec{a}^T(\alpha K_{Y,k_1})\vec{a} + \vec{a}^T(\beta K_{Y,k_2})\vec{a}$$

$$= \alpha\underbrace{\vec{a}^T K_{Y,k_1}\vec{a}}_{\ge 0} + \beta\underbrace{\vec{a}^T K_{Y,k_2}\vec{a}}_{\ge 0} \ge 0$$

~~(iv) let k_1, k_2 be k~~

(iv). Let k_1, k_2 be kernels over $X \times X$, then

$$k(x,y) = k_1(x,y)\, k_2(x,y)$$ is a kernel

Prof: $A \otimes B$: 点乘

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nn}B \end{pmatrix} \quad n^2 \times n^2$$

$A \otimes B$ is positive semi-definite $\Rightarrow$ $A \circ B$ is also 正 semi-definite

Let $Y = \{x_1, \ldots x_n\} \subseteq X$ be a finite set

$\Rightarrow$ Mercer's theorem: It suffices to show $K_{Y,k}$ is positive definite

- $K_{Y,k} = K_{Y,k_1} \circ K_{Y,k_2}$
- k_1, k_2 are kernels, $K_{Y,k_1} \otimes K_{Y,k_2}$ is positive semi-definite
- $K_{Y,k}$ is a principle matrix of $K_{Y,k} \otimes K_{Y,k_2}$ $= K_{Y,k_1} \circ K_{Y,k_2}$
- $K_{Y,k}$ is positive semi-definite.

(v) Let k_1 be a kernel over $X \times X$, and p be a polynomial with positive coefficients. Then

$$k(x,y) = p(k_1(x,y))$$ is a kernel.

Proof. It follows from (i), (iii)-(iv)

need (i) to cover the constant term in the polynomial

(vi). Let $k_1 : X \times X \to \mathbb{R}$ be a kernel and let $f : \mathbb{R} \to \mathbb{R}$ be a function expressed as power series with non-negative coefficients only. Then

$$k(x,y) = f(k_1(x,y))$$ is a kernel.

Proof. $f = \lim_{n\to\infty} p_n$ for polynomials with non-negative coefficients

$\Rightarrow$ for any finite $Y \subseteq X$: $K_{Y,f(k_1)} = \lim_{n\to\infty} K_{Y,p_n(k_1)}$

by (v) $K_{Y,p_n(k_1)}$ is positive definite for all n, and hence, for all $\vec{v}$:

$$\vec{v}^T K_{Y,f(k_1)} \vec{v} = \vec{v}^T \left(\lim_{n\to\infty} K_{Y,p_n(k_1)}\right) \vec{v} = \lim_{n\to\infty} \vec{v}^T K_{Y,p_n(k_1)} \vec{v} \geq 0$$

(vii) Let $k_1 : X \times X \to \mathbb{R}$ be a kernel. Then

$$k(x,y) = e^{k_1(x,y)}$$ is a kernel

Proof: Immediate fr. (vi), as all coefficients of $e^{k_1(x,y)} = \sum_{n=0}^{\infty} \frac{k_1(x,y)^n}{n!}$ are non-negative

(viii) The function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defined by

$$k(\vec{x}, \vec{y}) = \exp\left(-\frac{\|\vec{x}-\vec{y}\|_2^2}{2\sigma^2}\right)$$ for all $\vec{x}, \vec{y} \in \mathbb{R}^d$

is a kernel for every d, and for any $\sigma \in \mathbb{R}^+$. It is called Gaussian or radial basis function (RBF) kernel.

Proof. $k'(\vec{x}, \vec{y}) = \exp\left(\frac{\langle \vec{x}, \vec{y} \rangle}{\sigma^2}\right)$

$$\frac{k'(\vec{x},\vec{y})}{\sqrt{k'(\vec{x},\vec{x})}\sqrt{k'(\vec{y},\vec{y})}} = \underbrace{\exp\left(\frac{\langle \vec{x}, \vec{y} \rangle}{\sigma^2}\right)}_{= k(\vec{x},\vec{y})} \cdot \underbrace{\left(\exp\left(-\frac{\langle \vec{x}, \vec{x} \rangle}{2\sigma^2}\right) \cdot \exp\left(-\frac{\langle \vec{y}, \vec{y} \rangle}{2\sigma^2}\right)\right)}_{(i)}$$

(vii) (iv)

common Kernel functions over $\mathbb{R}^d \times \mathbb{R}^d$

linear kernel: $k(\vec{x}, \vec{y}) := \vec{x}^T \vec{y}$

poly... kernel $k(\vec{x}, \vec{y}) := (\vec{x}^T \vec{y} + c)^k$

Gaussian or RBF: $k(\vec{x}, \vec{y}) = \exp\left(-\frac{\|\vec{x}-\vec{y}\|_2^2}{2\sigma^2}\right)$

Support Vector Machine

- Choose hyperplane with the ~~largest~~ maximum margin γ:

  (1) Robust against noise
  (2) Excellent predictive performance in practice
  (3) separating hyperplane becomes unique
  (4) Good generalization is at odds with capacity
  minimize the structural risk (high dim ⇒ high structural risk)

  For any $\gamma > 0$, and for any $S \subset \{\vec{x} \in \mathbb{R}^d : \|\vec{x}\| \le R\}$ that can be γ-shattered by the hyperplanes:

  $$|S| \le \min\left(\left(\frac{R}{\gamma}\right)^2, d\right) + 1$$

  ⇒ bound on the VC-dim of hyperplanes separating with margin $\ge \gamma$
  - VC-dim of half-spaces $\mathbb{R}^d$ for "ordinary" shattering: $d+1$

  ⇒ Larger margin implies smaller VC-dim.
  ⇒ Max margin minimizes the structural risk of separating hyperplanes.

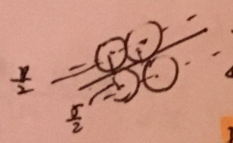
- Hard Margin SVM

  hyperplane for $f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b = 0$

  (1) $\vec{w}$ is orthogonal to the hyperplane
  (2) signed distance of a point $\vec{x}'$ from the hyperplane
  $$\lambda = \frac{f(\vec{x}')}{\|\vec{w}\|}$$

- The Maximum Margin hyperplane

  hyperplane $\langle \vec{w}, \vec{x} \rangle + b = 0$: scale $\vec{w}$ and b such that $|\langle \vec{w}, \vec{x}' \rangle + b| = 1$ of all points on the dashed hyperplanes

  margin: $\gamma = \frac{2}{\|\vec{w}\|}$

  prediction of an unseen instance $\vec{x}'$: $\text{sign}(\langle \vec{w}, \vec{x} \rangle + b)$

- non-convex optimization problem

  $$\max_{\vec{w}, b} \frac{2}{\|\vec{w}\|}$$

  subject to $|\langle \vec{w}, \vec{x}_i \rangle + b| \ge 1$ for $i = 1, \dots n$

equivalent formulation ($\max \frac{2}{\|\vec{w}\|} = \min \frac{1}{2}\|\vec{w}\|^2$)

Primal optimization problem: $\min_{\vec{w}, b} \frac{1}{2}\|\vec{w}\|^2$

s.t $-[y_i(\langle \vec{w}, \vec{x}_i \rangle + b) - 1] \le 0$ for $i = 1, \dots n$

Dual optimization Problem

$$\max_{\vec{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \vec{x_i}, \vec{x_j} \rangle$$

$$s.t \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0 \quad i = 1, \dots n$$

. Soft Margin SVM. (data is not linearly separable
idea: allow violations, but penalize them)

① Optimization problem with $C > 0$

$$\min_{\vec{w}, b, \vec{\xi}} \quad C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\vec{w}\|^2$$

$$s.t \quad y_i (\langle \vec{w}, \vec{x_i} \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad i = 1, \dots n$$

($\xi_i = 0$: correct classification
$0 < \xi_i \leq 1$: lies inside the margin, but on correct side
$\xi_i > 1$: lies on the wrong side)

$\sum_{i=1}^{n} \xi_i$ is an upper bound on the # of misclassified points

② Dual form.

$$\max_{\vec{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_i \langle \vec{x_i}, \vec{x_j} \rangle$$

$$s.t \quad 0 \leq \alpha_i \leq C \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad i = 1, \dots n$$