1. Basis

Linear Algebra

Definition: Inner Product

$$\vec{x}, \vec{z} \in \mathbb{R}^d: \langle \vec{x}, \vec{z} \rangle = \vec{x}^\top \vec{z} = \sum_{i=1}^{d} x_i z_i$$

Definition: Euclidean Norm

$$\vec{x} \in \mathbb{R}^d: \|\vec{x}\| = \sqrt{\langle \vec{x}, \vec{x} \rangle}$$

Definition: Posite Semi-definite Matrix

A symmetric $n \times n$ quadratic matrix A is positive semi-definite

$\Leftrightarrow \forall \vec{v} \in \mathbb{R}^n : \vec{v}^\top \mathbf{A} \vec{v} \geq 0$

$\Leftrightarrow$ For all eigenvalues λ_i : $\lambda_i \geq 0$

Theorem: Spectral Decomposition

Decompose symmetric matrix A into a diagonal matrix holding the eigenvalues and a matrix holding the corresponding eigenvectors.

Let: Symmetric $n \times n$ matrix A

Then: There exists an orthogonal matrix U and diagonal matrix Λ, s.t.:

$$A = U \Lambda U^T$$

Eigenvalues: $\lambda_i = \Lambda_{ii}$

Eigenvectors: $\vec{u}_i$: i-th column of U

Definition: Moore-Penrose Pseudoinverse

For a matrix $A \in \mathbb{R}^{n \times m}$, there exists a matrix $A^+ \in \mathbb{R}^{m \times n}$, that satisfies:

(i) $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$
(ii) $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$
(iii) $(\mathbf{A}\mathbf{A}^+)^\top = \mathbf{A}\mathbf{A}^+$
(iv) $(\mathbf{A}^+\mathbf{A})^\top = \mathbf{A}^+\mathbf{A}$

$$A^+ = (A^T A)^{-1} A^T$$

Definition: Inner Product (or Pre-Hilbert) Spaces

A vector space $\mathcal{H}$ is an inner product space, if it is equipped with a function:

$$\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$$

That satisfies, for all $\vec{x}, \vec{y} \in \mathcal{H}$ and scalars λ_1, λ_2:

1. $\langle \vec{x}, \vec{y} \rangle = \langle \vec{y}, \vec{x} \rangle$
2. $\langle \vec{x}, \vec{x} \rangle \geq 0$ with $\langle \vec{x}, \vec{x} \rangle = 0$ iff $\vec{x} = \vec{0}$
3. $\langle \lambda_1 \vec{x} + \lambda_2 \vec{y}, \vec{z} \rangle = \lambda_1 \langle \vec{x}, \vec{z} \rangle + \lambda_2 \langle \vec{y}, \vec{z} \rangle$

Definition: Hadamar Product

Let: $\mathbf{A} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{pmatrix}$

Hadamard Product: $\mathbf{A} \circ \mathbf{B} = \begin{pmatrix} a_{11}b_{11} & \dots & a_{1n}b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & \dots & a_{mn}b_{mn} \end{pmatrix}$

Definition: kronecker Product

Let: $\mathbf{A} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} b_{11} & \dots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \dots & b_{pq} \end{pmatrix}$

Kronecker Product: $\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \dots & a_{mn}\mathbf{B} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & \dots & a_{11}b_{1q} & \dots & a_{1n}b_{11} & \dots & a_{1n}b_{1q} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & \dots & a_{11}b_{pq} & \dots & a_{1n}b_{p1} & \dots & a_{1n}b_{pq} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1}b_{11} & \dots & a_{m1}b_{1q} & \dots & a_{mn}b_{11} & \dots & a_{mn}b_{1q} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & \dots & a_{m1}b_{pq} & \dots & a_{mn}b_{p1} & \dots & a_{mn}b_{pq} \end{pmatrix}$

Kronecker Matrix: $(mp) \times (nq)$ matrix with elements

$$(\mathbf{A} \otimes \mathbf{B})_{uv} = (\mathbf{A})_{ij}(\mathbf{B})_{kl}$$

with

$$\begin{aligned} u &= p(i-1)+k \\ v &= q(j-1)+l \end{aligned}$$

for all $i = 1, \dots, m, j = 1, \dots, n, k = 1, \dots, p, l = 1, \dots, q$

Properties of the Hadamard and kroneck Products

Given: $n \times n$ matrices A and B

1. Eigenvalues of $A \otimes B$ are the product of the eigenvalues of A and B
2. $A \circ B$ is a principal submatrix of $A \otimes B$

   $A \circ B$ can be obtained from $A \otimes B$ by deleting rows and columns

$\mathbf{A} \otimes \mathbf{B} =$ $\begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} & a_{13}b_{11} & a_{13}b_{12} & a_{13}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} & a_{13}b_{21} & a_{13}b_{22} & a_{13}b_{23} \\ a_{11}b_{31} & a_{11}b_{32} & a_{11}b_{33} & a_{12}b_{31} & a_{12}b_{32} & a_{12}b_{33} & a_{13}b_{31} & a_{13}b_{32} & a_{13}b_{33} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} & a_{23}b_{11} & a_{23}b_{12} & a_{23}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} & a_{23}b_{21} & a_{23}b_{22} & a_{23}b_{23} \\ a_{21}b_{31} & a_{21}b_{32} & a_{21}b_{33} & a_{22}b_{31} & a_{22}b_{32} & a_{22}b_{33} & a_{23}b_{31} & a_{23}b_{32} & a_{23}b_{33} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} & a_{32}b_{11} & a_{32}b_{12} & a_{32}b_{13} & a_{33}b_{11} & a_{33}b_{12} & a_{33}b_{13} \\ a_{31}b_{21} & a_{31}b_{22} & a_{31}b_{23} & a_{32}b_{21} & a_{32}b_{22} & a_{32}b_{23} & a_{33}b_{21} & a_{33}b_{22} & a_{33}b_{23} \\ a_{31}b_{31} & a_{31}b_{32} & a_{31}b_{33} & a_{32}b_{31} & a_{32}b_{32} & a_{32}b_{33} & a_{33}b_{31} & a_{33}b_{32} & a_{33}b_{33} \end{pmatrix}$

3. $A \otimes B$ is positive semi-definite $\Rightarrow A \circ B$ is positive semi-definite

Analysis

Definition: Pointwise Convergence

A sequence $\{f_n : \mathbb{R} \to \mathbb{R}\}$ of functions converges pointwise to a function to a function $f : \mathbb{R} \to \mathbb{R}$, if:

$$\lim_{n \to \infty} f_n(x) = f(x)$$

2. Computational Learning Theory

Predictive Learning: Function Approximation

Input: Instance space X with fixed distribution D_X
Output: Target space Y
Mapping: Function $f: X \to Y$
Hypothesis space: Set of allowed hypotheses $h \in \mathcal{H}$

Goal:

Given: A set of examples $(x, y) \in E \subseteq X \times Y$ with:
1. x drawn i.i.d according to D_X
2. $y = f(x)$

Find: A hypothesis $h \in \mathcal{H}$ such that the true error is minimized

$$\text{error}_{D_X}(h) := \mathbb{E}_{D_X}(\text{error}(f(x), h(x)))$$

→ expected average error between function and hypothesis w.r.t. the underlying distribution

⇒ Approximate function f as closely as possible with a hypothesis $h \in \mathcal{H}$

Error Measures:

Classification: $\text{error}(h(x), f(x)) = \begin{cases} 0 & \text{if } h(x) = f(x) \\ 1 & \text{o/w} \end{cases}$

Regression: $\text{error}(h(x), f(x)) = (h(x) - f(x))^2$

Model of Learning (Rivest):

1. **Learner:** Program/algorithm working in polynomial time and finite memory
   Find a consistent hypothesis $h \in \mathcal{H}$ from the hypothesis space $\mathcal{H}$
2. **Domain/Target:** Unknown concept/function/language intended to be learned
   Domain: X **Concept:** $c: X \to Y$
   Conjunctive concepts: $c : X \to \{0, 1\}$ with $X = \{0, 1\}^n$
3. **Information Source:** Information given to the learner to learn the domain (labeled example/oracle)
4. **Prior knowledge:** Initial knowledge about the domain
5. **Performance Criteria:** Assess the quality of the Learner

Version Spaces

A version space $VS_{H,E}$ w.r.t the hypothesis space H and the training examples E describes the set of consistent hypotheses in H:

$$\mathrm{VS}_{\mathcal{H},E} = \{h \in \mathcal{H} : h \text{ is consistent with } E\}$$

Representation: $\mathrm{VS}_{\mathcal{H},E} = \{h \in \mathcal{H} : \text{ there exist } g \in G \text{ and } s \in S \text{ such that } g \leq h \leq s\}$

A version space is defined through the general boundary and specific boundary.

General Boundary:

Maximal general hypotheses from H consistent with E

$g \in G \iff$
g is consistent with $E \wedge \forall g' \in \mathcal{H}$: if $g' < g$ then g' is inconsistent with E
$\Rightarrow$ "inconsistency": $E^- \cap g' \neq \emptyset$

Specific Boundary:

Maximal specific hypotheses from H consistent with E

$s \in S \iff$
s is consistent with $E \wedge \forall s' \in \mathcal{H}$: if $s < s'$ then s' is inconsistent with E
$\Rightarrow$ "inconsistency": $E^+ \not\subseteq s'$

Concept Learning (Binary Classification): $|Y| = 2$

f and h are binary valued: $f, h : X \to \{0, 1\}$

→ f and h represent subsets of X

Definition: Consistent Hypothesis Finding Problem

Concept: $c \subseteq X$

Domain: X

Hypothesis Space: $\mathcal{H} \subseteq 2^X$

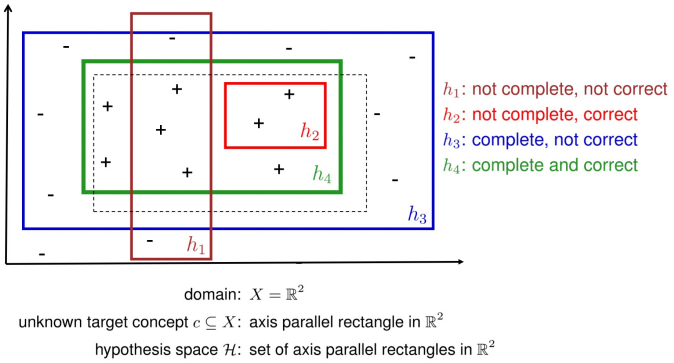
Training Examples: $E = E^+ \cup E^-$ with $E^+ \subseteq C$ and $E^- \subseteq X \setminus C$

Find a hypothesis $h \in \mathcal{H}$ consistent with E

Definition: Consistency

A hypothesis h is consistent with examples E, if:

1. h is complete: $E^+ \subseteq h$
2. h is correct: $E^- \cap h = \emptyset$



Definition: Cover

1. h covers x, if $x \in h$ $(h(x) = 1)$
2. h does not cover x, if $x \notin h$ $(h(x) = -1)$

Definition: Generality

$h_1 \in \mathcal{H}$ is more general than $h_2 \in \mathcal{H}$ if h_1 covers equal or more instances than h_2.

$$h_1 \leq h_2 \iff \{x \in X : h_1 \text{ covers } x\} \supseteq \{x \in X : h_2 \text{ covers } x\}$$

Definition: Strict Generality

$h_1 \in \mathcal{H}$ is strictly more general than $h_2 \in \mathcal{H}$ if h_1 covers more instances than h_2.

$$h_1 < h_2 \iff \{x \in X : h_1 \text{ covers } x\} \supsetneq \{x \in X : h_2 \text{ covers } x\}$$

Definition: Overfitting

Hypothesis $h \in \mathcal{H}$ overfits training data, if there is an alternative hypothesis $h' \in \mathcal{H}$ that has a larger empirical risk but a smaller true risk: $\text{error}_E(h) < \text{error}_E(h')$

$$\text{error}_{\mathcal{D}}(h) > \text{error}_{\mathcal{D}}(h')$$

Causes: noisy data, lack of representative data, lack of samples

Well- and ill-Posed Problems

Theorem 2.1:

A mathematical model of a physical phenomenon must satisfy the following conditions:

1. **existence**: the problem must have a solution
2. **uniqueness:** the solution must be unique
3. **stability:** the solution depends continuously on the data

Definition: Well-Posed Problem

A problem is well-posed if it satisfies all conditions of Theorem 1.1.

Else it is ill-posed.

Typically: Learning is an ill-posed problem

→ Regularization helps to restore well-posedness

Definition: Regularization

Methods to improve the stability of solutions for ill-posed inverse problems by limiting the capacity of the concept class.

→ also prevents overfitting

Theorem 2.2:

Let: $f \in \mathcal{H}$ for some hypothesis space $\mathcal{H}$,
S random sample of size m

Then with probability of at least $1-\delta$:

$$R[f] \leq R_{\text{emp}}[f] + \sqrt{\frac{\text{VC}_{\text{dim}}(\mathcal{H})\left(\ln \frac{2m}{\text{VC}_{\text{dim}}(\mathcal{H})} + 1\right) - \ln \frac{\delta}{4}}{m}}$$

⇒ higher capacity comes at the costs of a worse generalization.

⇒ Limiting the VC dimension leads to minimizing the structural risk and better generalization

Tikhonov Regularization:

Transform the original problem into a conditional-optimization problem that penalizes undesirable solutions by adding a term, called Regularizer

Given: Find function $f_S \in \mathcal{H}$, s.t. the error is minimized

Minimize: $R[f] = \int_{X \times Y} V(f(x), y) dP(x, y)$ i.e. $R_{\text{emp}}[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i)$

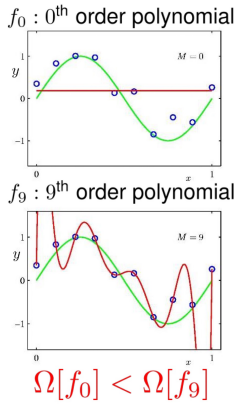
Regularization: $\underset{f \in \mathcal{H}}{\operatorname{argmin}} R_{\text{emp}}[f] + \lambda \Omega[f]$

Ω: Regularizer

Function monotonically increasing with the non-smoothness (complexity) of f

$\lambda \in \mathbb{R}$: Regularization parameter

Trade-Off: (I) Minimize Empirical Risk
(II) Minimize model complexity



PAC Learning Model

A mistake bound model for concept learning. The goal is not to properly learn a concept but introduces the technique of improper learning. The concept is intended to be learned probably approximately correct. It defines an upper bound for the expected error.

Definition: PAC-learnable

A concept class C is PAC-learnable, if there is an algorithm A, such that for any $\varepsilon > 0$ and $\delta > 0$, for all distributions D on X, the upper-bound holds:

$$P(\text{error}(h) \leq \varepsilon) \geq 1 - \delta$$

Definition: Efficient PAC-learnable

There exist a PAC-learning Algorithm that runs polynomial in $1/\varepsilon, 1/\delta, n$ (size of an instance of X) and size(c)

Finite Concept Classes

Theorem 2.3:

For any finite class C and for any finite hypothesis class $\mathcal{H}$, IF we draw m random examples and find a consistent hypothesis from $\mathcal{H}$ using algorithm A, with:

$$m \geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right) \Rightarrow C \text{ is PAC-learnable using } \mathcal{H}$$

Proof:

Let: D: distribution over X
$c \in C$: unknown target concept.
$e_1, \dots, e_m$: i.i.d. drawn examples w.r.t. D

1. Define events for all hypothesis $h \in \mathcal{H}$ and examples e_i

$E_{h,i}$: h is consistent with $e_i \Leftrightarrow c(e_i) = h(e_i)$
E_h: h is consistent with $e_1, \dots, e_m \Leftrightarrow \bigwedge_{i=1}^{m} E_{h,i}$

Probability of the events:

$$P(E_{h,i}) = P(h(e_i) = c(e_i)) = 1 - P(h(e_i) \neq c(e_i)) = 1 - error_{D,c}(h)$$

$$P(E_h) = P\left(\bigwedge_{i=1}^{m} E_{h,i}\right) = \prod_{i=1}^{m} P(E_{h,i}) = (1 - error_{D,c}(h))^m$$

(I) IF $error_{D,c} > \varepsilon \Rightarrow P(E_h) < (1-\varepsilon)^m$

2. Set an upper-bound for the error

Let: H: output of algorithm A
$\rightarrow$ consistent with $e_1, \dots, e_m$
$\mathcal{H}_{bad} = \{h \in \mathcal{H} : error_{D,c}(h) > \varepsilon\}$

$$P(error_{D,c}(H) > \varepsilon) \leq P\left(\bigvee_{h \in \mathcal{H}_{bad}} E_h\right) \leq \sum_{h \in \mathcal{H}_{bad}} P(E_h) < |\mathcal{H}|(1-\varepsilon)^m$$

(1)

prob of H belonging to $\mathcal{H}_{bad}$ — union bound — Bernoulli's inequality: $1 - x \leq e^{-x}$

(II) $P(error_{D,c}(H) > \varepsilon) < |\mathcal{H}|(1-\varepsilon)^m \leq |\mathcal{H}|\, e^{-\varepsilon m}$

3. Proof PAC-learnability

Set $m \geq \frac{1}{\varepsilon}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{1}{\delta}\right)\right)$ by choice

$$P(error_{D,c}(H) > \varepsilon) \leq |\mathcal{H}|\, e^{-\varepsilon m} = |\mathcal{H}| \exp\left(-\varepsilon \tfrac{1}{\varepsilon}\left(\ln(|\mathcal{H}|) + \ln\left(\tfrac{1}{\delta}\right)\right)\right) = \delta$$

$\Rightarrow$ If A finds a consistent hypothesis for at least m then A PAC-learns C using $\mathcal{H}$

Corollary 2.4:

If the previous theorem holds and:

1. Sample complexity m is polynomial in instance size n
2. A runs polynomial in the sample size and instance size n

Then:

A runs polynomial in $1/\varepsilon, 1/\delta, n, \text{size}(c)$

$\Rightarrow C$ is efficiently PAC-learnable

Infinite Concept Classes

Definition: Dichotomy

A partition of a set in two adjoint subsets

Linear separability:

Dichotomy (S^+, S^-) of $S \subseteq \mathbb{R}^d$ is linear separable
$\Leftrightarrow$ There exist a $(d-1)$-dimensional affine hyperplane separating S^+ and S^-

Definition: Family of subsets S realized by C

For any concept class C over X and for any finite subset $S \subseteq X$:

$$\Pi_C(S) = \{c \cap S : c \in C\}$$

$\rightarrow$ All subsets realizable by C

Definition: Shattering

$S \subseteq X$ is shattered by C, if $\Pi_C(S) = 2^S$
$\Leftrightarrow$ All dichotomies of S are realized by some concept in C

Definition: Vapnik-Chervonenkis Dimension

For a concept class $C \subseteq 2^X$ over instance space X the VC Dimension is the size of the largest subset $S \subseteq X$ that can be shattered by C.

Largest d with $|S| = d$, s.t. all dichotomies of S can be realized by a concept in C

IF there is no such set $VC_{dim}(C) = \infty$

$\Rightarrow$ Measure of expressive power of concept class C

Technique to show $VC_{dim}(C)$

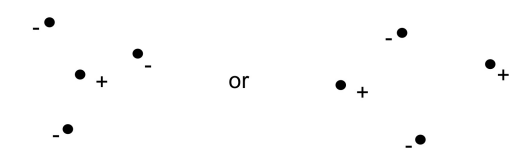
1. Show explicitly lower bound a, s.t. $a \leq VC_{dim}(C)$
   Give an example subset $S \subseteq X$ that can be shattered by C with $|S| = a$
2. Show upper bound b, s.t. $VC_{dim}(C) < b$
   Show that there exists now subset S of size b that can be shattered by C

Example:

$\mathcal{C}$: set of half-spaces of the plane

- samples of cardinalities 3 can be shattered
- however, no sample of cardinality 4 can be shattered

$\Rightarrow VC_{dim}(\mathcal{C}) = 3$



Theorem 2.5: Blumer, Ehrenfeucht, Haussler, Warmuth Theorem

For any class C and for any hypothesis class $\mathcal{H}$,
IF we draw m random examples and find a consistent hypothesis from $\mathcal{H}$ using algorithm A, with:

$$\begin{aligned} m &\geq \frac{4}{\epsilon}\log_2\frac{2}{\delta} + \frac{8\cdot \mathrm{VC_{dim}}(\mathcal{C})}{\epsilon}\log_2\frac{13}{\epsilon} \\ &= O\left(\frac{1}{\epsilon}\left(\log\frac{1}{\delta} + \mathrm{VC_{dim}}(\mathcal{C})\log\frac{1}{\epsilon}\right)\right) \end{aligned}$$

Proof: (sketch)

1. Polynomial upper bound for $|\Pi_C(S)|$

Perles-Shelah, Sauer, Vapnik-Chervonenkis theorem

Define: $\Pi_C(m) = \max\{|\Pi_C(S)| : S\subseteq X, |S|=m\}$, for $m\in\mathbb{N}$

→ Maximal number of dichotomies realizable by concept class C on a subset $S\subseteq X$ of size m

For $d = VC_{dim}(\mathcal{C}) > 0$:

(1) $\Pi_C(m) = 2^m$, if $m\leq d$

(2) $\Pi_C(m) \leq \left(\frac{me}{d}\right)^d$, else

⇒ For $m>d$: $|\Pi_C(S)| \leq \left(\frac{me}{d}\right)^d$

2. ε-nets

Notions:

1. hypothesis h is bad w.r.t. a concept $c\in C$, if $error(h)\geq\varepsilon$

→ $error(h) = Pr_{x\in D}[x\in c\Delta h]$

$c\Delta h$: $c\setminus h$

2. Error Regions:

For a fixed target concept $c\in\mathcal{C}$:

Regions not covered by h: $\Delta(c) = \{h\Delta c : h\in C\}$

Points with probability $>\varepsilon$ according to D:

$$\Delta_\epsilon(c) = \{r\in\Delta(c) : \mathrm{Pr}_{x\in D}[x\in r]\geq\epsilon\}$$

Definition: ε-Net

For any $\varepsilon>0$, a set $S\subseteq X$ is an ε-net, if every region in $\Delta_\varepsilon(c)$ is hit by a point:

$$S\cap r\neq\emptyset \text{ for every } r\in\Delta_\epsilon(c)$$

⇒ Any hypothesis consistent with an ε-net is good

⇒ An upper bound that a random sample fails to form an ε-net is also an upper bound for:

$P(\text{error}(h) \geq \varepsilon)$

Goal: Draw a random sample S that forms an ε-net with a high probability

Probability that S fails to form an ε-net:

$$\begin{aligned} &\Pr[S \text{ fails to form an } \epsilon\text{-net}] \\ &= \Pr[\exists c' \in \Delta_\epsilon(c) \text{ s.t. } S \cap c' = \emptyset] \\ &\leq \sum_{c' \in \Delta_\epsilon(c)} \Pr[S \cap c' = \emptyset] \end{aligned}$$

→ intractable to compute as $|\Delta_\varepsilon(c)|$ grows to infinity

⇒ Estimate through Double Sampling technique

3 Double Sampling Technique

First Sampling Run:

Step 1: Draw first random sample S_1 of size m

event A_m: S_1 *fails* to form an ϵ-net, i.e.,

$$A_m : \exists c' \in \Delta_\epsilon(c) \text{ such that } S_1 \cap c' = \emptyset$$

goal: $\Pr[A_m] \leq \delta$

Step 2: Draw second random sample S_2 of size m

event B_m: A_m happens and S_2 has at least $\frac{\epsilon m}{2}$ hits in some region of $\Delta_\epsilon(c)$ that has been missed by S_1, i.e.,

$$B_m : \exists c' \in \Delta_\epsilon(c) \text{ such that } S_1 \cap c' = \emptyset \text{ and } |S_2 \cap c'| \geq \frac{\epsilon m}{2}$$

⇒

$$\begin{aligned} A_m &: \exists c' \in \Delta_\epsilon(c) \text{ with } c' = c \Delta h' \text{ for some } h' \in \mathcal{C} \text{ s.t. } S_1 \cap c' = \emptyset \\ B_m &: \exists c' \in \Delta_\epsilon(c) \text{ with } c' = c \Delta h' \text{ for some } h' \in \mathcal{C} \text{ s.t. } S_1 \cap c' = \emptyset \wedge |S_2 \cap c'| \geq \tfrac{\epsilon m}{2} \end{aligned}$$

Lemma: If $m > \frac{8}{\epsilon}$ then $\Pr[A_m] \leq 2 \cdot \Pr[B_m]$

⇒ To show $\Pr[A_m] \leq \delta$ it suffices to show that $\Pr[B_m]$ is small, i.e. $\Pr[B_m] \leq \frac{\delta}{2}$

Second Sampling Run

Step 1: Draw sample S of size $2m$

Step 2: Pick one of the $\binom{2m}{m}$ m-subsets of S uniformly at random for S_1' and the rest for S_2'

event B_m': $\exists c' \in \Delta_\epsilon(c)$ such that $S_1' \cap c' = \emptyset$ and $|S_2' \cap c'| \geq \frac{\epsilon m}{2}$

Fact: $\Pr[B_m] = \Pr[B_m']$, as S_1, S_2 and S_1', S_2' are identically distributed

- i.i.d. sampling

goal: upper bound on $\Pr[B_m']$

Upper Bound of $\Pr[B_m] = \Pr[B_m']$:

$$\Pr[B_m']$$

$$= \Pr\left[\exists c' \in \Delta_\epsilon(c) \text{ s.t. } c' \cap S_1' = \emptyset \wedge |c' \cap S_2'| \geq \frac{\epsilon m}{2}\right]$$

$$= \Pr\left[\exists Y = c' \cap S \text{ for some } c' \in \Delta_\epsilon(c) \text{ s.t. } Y \cap S_1' = \emptyset \wedge |Y \cap S_2'| \geq \frac{\epsilon m}{2}\right]$$

$$\leq \sum_{Y \in \{c' \cap S : c' \in \Delta_\epsilon(c)\}} \Pr\left[Y \cap S_1' = \emptyset \wedge |Y \cap S_2'| \geq \frac{\epsilon m}{2}\right]$$

the set system $\{c' \cap S : c' \in \Delta_\epsilon(c)\}$ is **finite**

$\Rightarrow$ we can use the union bound!

$S_1', S_2' \subset S$

$\Rightarrow c' \cap S_1' = (c' \cap S) \cap S_1' = Y \cap S_1'$

$\Rightarrow c' \cap S_2' = (c' \cap S) \cap S_2' = Y \cap S_2'$

Claim A: Let $d = VC_{\text{dim}}(\mathcal{C})$. Then $|\{c' \cap S : c' \in \Delta_\epsilon(c)\}| \leq \left(\frac{2em}{d}\right)^d$

Proof:

$$\begin{aligned}
|\{c' \cap S : c' \in \Delta_\epsilon(c)\}| &= |\Pi_{\Delta_\epsilon(c)}(S)| && \text{// } \Delta_\epsilon(c) \text{ is a concept class} \\
&\leq |\Pi_{\Delta(c)}(S)| && \text{// as } \Delta_\epsilon(c) \subseteq \Delta(c) \\
&\leq \left(\frac{2em}{\text{VC}_{\text{dim}}(\Delta(c))}\right)^{\text{VC}_{\text{dim}}(\Delta(c))} && \text{// by Sauer's theorem} \\
&= \left(\frac{2em}{d}\right)^d && \text{// by Lemma 5}
\end{aligned}$$

Claim B: For any $Y \in \{c' \cap S : c' \in \Delta_\epsilon(c)\}$:

$$\Pr\left[Y \cap S_1' = \emptyset \text{ and } |Y \cap S_2'| \geq \frac{\epsilon m}{2}\right] \leq 2^{-\frac{\epsilon m}{2}}$$

Proof: let $Y \in \{c' \cap S : c' \in \Delta_\epsilon(c)\}$

(i) if $|Y| > m$ or $\frac{\epsilon m}{2} > |Y|$ then $\Pr\left[Y \cap S_1' = \emptyset \text{ and } |Y \cap S_2'| \geq \frac{\epsilon m}{2}\right] = 0$

(ii) if $\frac{\epsilon m}{2} \leq |Y| \leq m$ then

$$\begin{aligned}
&\Pr\left[Y \cap S_1' = \emptyset \text{ and } |Y \cap S_2'| \geq \frac{\epsilon m}{2}\right] \\
&= \Pr[Y \subseteq S_2'] \\
&= \frac{\binom{2m-|Y|}{m}}{\binom{2m}{m}} && \text{// nominator: number of "good" choices for } S_1' \\
&= \prod_{i=0}^{|Y|-1} \frac{m-i}{2m-i} \\
&\leq 2^{-|Y|} \\
&\leq 2^{-\frac{\epsilon m}{2}}
\end{aligned}$$

Putting together claim A and B.

$$\Pr[B'_m] \leq \sum_{Y \in \{c' \cap S : c' \in \Delta_\epsilon(c)\}} \Pr\left[Y \cap S'_1 = \emptyset \text{ and } |Y \cap S'_2| \geq \frac{\epsilon m}{2}\right]$$

$$\leq \left(\frac{2em}{d}\right)^d \cdot 2^{-\frac{\epsilon m}{2}}$$

$$\Rightarrow \Pr[A_m] \leq 2 \cdot \Pr[B_m] = 2 \cdot \Pr[B'_m] \leq 2 \cdot \left(\frac{2em}{d}\right)^d \cdot 2^{-\frac{\epsilon m}{2}}$$

$\Rightarrow$ $\Pr[\text{a random sample of size } m \text{ fails to form an } \epsilon\text{-net}] \leq \delta$ **whenever**

$$2 \cdot \left(\frac{2em}{d}\right)^d \cdot 2^{-\frac{\epsilon m}{2}} \leq \delta$$

For $m \geq \frac{4}{\epsilon}\log_2\frac{2}{\delta} + \frac{8 \cdot \text{VC}_{\text{dim}}(\mathcal{C})}{\epsilon}\log_2\frac{13}{\epsilon}$, the inequality holds

at $m = \max\left(\frac{4}{\epsilon}\log_2\frac{2}{\delta}, \frac{8d}{\epsilon}\log_2\frac{13}{\epsilon}\right)$

$\Rightarrow P(\text{error}(h) \geq \varepsilon) \leq \delta$

$\Rightarrow$ A PAC-learns $\mathcal{C}$ using H

Corollary 2.6:

Let: X instance space, $\mathcal{C}$ concept class

$\mathcal{C}$ is polynomial PAC-learnable, if:

1. $\text{VC}_{\text{dim}}(\mathcal{C})$ is bounded polynomial in instance size n
2. Consistent hypothesis finding provided a sample S runs polynomial in $|S|$ and n

Theorem 2.7: Radon's Theorem

Any set of $d+2$ points in $\mathbb{R}^d$ can always be partitioned in two subsets S_1 and S_2, s.t.:

$$\text{Conv}(S_1) \cap \text{Conv}(S_2) \neq \emptyset$$ Conv(S): convex hull

Theorem 2.8: Mangasarian Theorem

Given: two disjoint finite subsets S_1 and S_2 of $\mathbb{R}^d$ that are linearly separable

Then: One can find a separating hyperplane in time polynomial in d and $|S_1 \cup S_2|$

Proof: Problem solved through linear programming (P)

PAC-Learning Results

Theorem 2.9:

For a constant k, $k\text{-CNF}_n / k\text{-DNF}_n$ is efficiently PAC-learnable

Proof: Use corollary 2.2 (dual for k-DNF_n)

Let: $\mathcal{H} = \mathcal{C}$

1. Show $\ln|\mathcal{C}|$ is polynomial in $n \Leftrightarrow$ Show $|\mathcal{C}| = 2^{O(n^k)}$

   1.1 Number of clauses with at most k literals

   Number of clauses with i literals: $n_i = \binom{2n}{i}$

   Number of clauses with at most k literals:

   $$\sum_{i=1}^{k} \binom{2n}{i} \leq k \cdot (2n)^k$$

   $$\binom{2n}{i} = \frac{2n!}{i!(2n-i)!} \leq \frac{2n!}{(2n-k)!} \leq \prod_{j=0}^{k-1} 2n \leq (2n)^k$$

   1.2 Number of different k-CNFs

   Let: $\{C_1, \ldots, C_L\}$: denotes set of clases with at most k literals

   $\Rightarrow$ Every k-CNF ϕ can be written as:

   $$\bigwedge_{i \in I} C_i \text{, for some } I \subseteq [L]$$

   $\Rightarrow$ The number of k-CNFs is bounded by the number of subsets $I \subseteq [L]$

   $\Rightarrow$ Since $L \leq k(2n)^k \Rightarrow$ Number of k-CNFs: $2^{k(2n)^k}$

   $\Rightarrow \ln|\mathcal{C}|$ is polynomial in n

2. Show that the consistant hypothesis finding algorithm runs polynomial in the sample size and instance size n

   2.1 Transform k-CNF to a conjunction over $v_1', \ldots, v_L'$

   Each variable v_i' stands for clause C_i:

   $$v_i' = 1 \Leftrightarrow C_i = 1$$

   2.2 Use Find-S conjunctions to find a consistent hypothesis

   $\Rightarrow$ Find-S runs polynomial in sample size and instance size

$\Rightarrow$ k-CNFs are efficiently PAC-learnable $\square$

Definition. Consistency Problem for k-term DNF

Given: $S = \{\langle x_1, c(\vec{x}_1)\rangle, \ldots, \langle x_m, c(\vec{x}_m)\rangle\}$

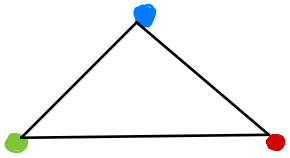
with unknown concept: $c : \{0,1\}^n \to \{0,1\}$.

Decide whether there exists a $h \in$ k-term-DNF that is consistant with S

Definition: 3-Colorability Problem for Graphs

Given: undirected graph $G = (V, E)$

Decide: Does a 3-coloring $\varphi: V \to \{R, G, B\}$ exist such that: $\varphi(u) \neq \varphi(v)$, if $(u,v) \in E$



3-Colorability is NP-complete

Theorem 2.10:

The consistancy problem for k-term DNF_n is NP-complete

Proof:

Step 1: Proof 3-term DNF_n is NP

Given an example solution v,
we can construct a Turing machine that verifies, if v is consistant on S
$\Rightarrow$ 3-term DNF is NP

Step 2: Proof 3-term DNF_n is NP-Hard

Polynomial reduction to 3-colorability problem for graphs

Let: $G = (V, E)$ undirected graph

w.l.o.g. $V = \{v_1, \ldots, v_n\}$

Construct 3-term DNF_n input:

$$\overline{\delta}_{ij} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{otherwise.} \end{cases} \quad (i,j \in [n]) \qquad (\text{“negative Kronecker delta”})$$

$$\omega : V \to \{0,1\}^n, \quad v_i \mapsto (1, \ldots, 1, \overset{i\text{-th entry}}{0}, 1, \ldots, 1) = (\overline{\delta}_{ij})_{j=1}^n$$

$$S^+ = \{\langle \omega(v_i), 1\rangle : i \in [n]\}$$

$$S^- = \{\langle \omega(v_i) \,\&\, \omega(v_j), 0\rangle : i \neq j, \{v_i, v_j\} \in E\} \qquad (\&: \text{bitwise and})$$

For every node: Create a positive example
For every edge: Create a negative example

Claim:

G has 3-coloring $\Leftrightarrow \exists$ 3-term DNF_n h: h is consistent with S

Proof "$\Rightarrow$": Suppose $\varphi: V \to \{R, G, B\}$ is a 3-coloring of G

Let $T_c = \bigwedge_{i \in [n]: \varphi(v_i) \neq c} l_i$, for all $c \in \{R, G, B\}$

Then $h = T_R \vee T_G \vee T_B$

h is complete w.r.t S^+: ($l_i = 0$, rest 1)

For $i \in [n]$: $\ell_i \notin T_{\varphi(v_i)} \overset{(\omega(v_i))_{j \neq i} = 1}{\Longrightarrow} T_{\varphi(v_i)}(\omega(v_i)) = 1 \implies h(\omega(v_i)) = 1$

(l_i not included)

h is correct w.r.t. S^-: ($l_i = l_j = 0$)

IF $i \neq j$ such that $\{v_i, v_j\} \in E$, then:

$$\varphi(v_i) \neq \varphi(v_j) \implies \ell_i \in T_{\varphi(v_j)}, \ell_j \in T_{\varphi(v_i)} \overset{(\star)}{\Longrightarrow} h(\omega(v_i) \,\&\, \omega(v_j)) = 0$$

since $(\star)$ $\ell_i, \ell_j \in T_c$ for $c \in \{\color{red}\bullet, \color{blue}\bullet, \color{green}\bullet\} \setminus \{\varphi(v_i), \varphi(v_j)\}$.

(l_i or l_j are included in every term)

Proof "$\Leftarrow$": Suppose $h = T_R \vee T_G \vee T_B$ is a 3-term DNF_n consistent with S

Define: $\varphi: V \to \{R, G, B\}$,

$$v_i \mapsto \min\{c \in \{R, G, B\} \mid T_c(\omega(v_i)) = 1\}$$ with $R < G < B$

Step 1: φ is well-defined ✓

Step 2: φ is a 3-coloring: $i \neq j, \{v_i, v_j\} \notin E \Longleftarrow \varphi(v_i) = \varphi(v_j)$

Suppose $i \neq j$. Then

$$\begin{aligned} c := \varphi(v_i) = \varphi(v_j) &\implies T_c(\omega(v_i)) = 1 = T_c(\omega(v_j)) && \text{(def. of } \varphi) \\ &\implies \ell_i, \ell_j \notin T_c \implies T_c(\omega(v_i) \,\&\, \omega(v_j)) = 1 \\ &\implies h(\omega(v_i) \,\&\, \omega(v_j)) = 1 \implies \{v_i, v_j\} \notin E && \text{(choice of } h) \end{aligned}$$

($\omega(v_i) \,\&\, \omega(v_j)$ is not a negative example)

Definition: Random Polynomial Time

Class of languages $L \subseteq \Sigma^*$ that can be decided by a randomized algorithm $\mathfrak{A}$ in polynomial time as follows:

if $w \notin L$, then $\mathfrak{A}$ always rejects w (i.e., with probability 1)

if $w \in L$, then $\mathfrak{A}$ accepts w with probability at least $\frac{1}{2}$

Remarks:

1. $RP \subseteq NP$ ($RP \subset NP$?)
2. $\mathfrak{A}$ is often referred as Monte Carlo algorithm

Theorem 2.11:

If 3-term DNF_n is efficiently PAC-learnable using 3-term-DNF_n, then $RP = NP$

Proof: Show that this implies that the consistency problem is in RP

Let: Algorithm A efficiently PAC-learns 3-term DNF_n using 3-term DNF_n

$\Rightarrow$ A's running time is bounded polynomial in:

$T = p(n, 1/\varepsilon, 1/\delta)$

Construct polynomial algorithm A':

Let: Set of examples $S = \{\langle x_1, c(\vec{x}_1)\rangle, \ldots, \langle x_m, c(\vec{x}_m)\rangle\}$,
Unknown target concept $c : \{0,1\}^n \to \{0,1\}$

Algorithm: A' runs A with: $\varepsilon = \frac{1}{m+1}$, $\delta = \frac{1}{2}$ and D over $\{0,1\}^n$ as a uniform distribution over S

→ simulates random oracle

Output:

1. (i) A does not terminate in T steps
   (ii) A terminates but does
   } "No"
2. A terminates and outputs hypothesis h on S:
   Test hypothesis h on S:
   (i) h is consistent with S ⟶ "Yes"
   (ii) h is not consistent with S ⟶ "No"

Claim: $h \in$ 3-term-DNF_n is consistent with $S \Leftrightarrow$ error $\leq \frac{1}{m+1}$

Verification:

$\mathfrak{A}$	output of $\mathfrak{A}'$	$\nexists$ consistent hyp.	$\exists$ consistent hyp.
doesn't terminate in time T	NO	correct	can't happen
terminates without output	NO	correct	wrong happens with prob. $< \delta = \frac{1}{2}$
returns an inconsistent hyp.	NO	correct	wrong happens with prob. $< \delta = \frac{1}{2}$
returns a consistent hyp.	YES	can't happen	correct

Putting together:

No consistent hypothesis h: A' always returns "No"
Consistent hypothesis: A' returns "Yes" with a probability of at least $\frac{1}{2}$

$\Rightarrow$ Consistency problem for 3-term-DNF_n is in RP

$\Rightarrow$ $RP = NP$

□

Theorem 2.12:

$3\text{-term-DNF}_n \subseteq 3\text{-CNF}_n$

Proof:

Let: $L = \{v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n\}$ set of literals and $\phi \in 3\text{-Term-DNF}_n$

Then: $\phi = T_1 \vee T_2 \vee T_3$ with $T_i = \bigwedge_{l \in L_i} l$ for $L_i \subseteq L$, $i \in [3]$

Construct 3-CNF_n:

$$\phi = \bigvee_{i=1}^{3} T_i = \left(\bigwedge_{l \in L_1} l\right) \vee \left(\bigwedge_{l \in L_2} l\right) \vee \left(\bigwedge_{l \in L_3} l\right) = \bigwedge_{(l_1, l_2, l_3) \in L_1 \times L_2 \times L_3} (l_1 \vee l_2 \vee l_3)$$

$\Rightarrow$ Every 3-term DNF$_n$ can be transformed to a 3-CNF$_n$

$\Rightarrow$ 3-term-DNF$_n \subseteq$ 3-CNF$_n$

Theorem 2.13:

3-term-DNF$_n$ is polynomial PAC-learnable using 3-CNF$_n$

Proof:

(I) From Theorem 2.12: 3-term-DNF $\subseteq$ 3-CNF$_n$

(II) From Theorem 2.9: k-CNF$_n$ is efficiently PAC-Learnable

From (II):

1. Size of 3-CNF$_n$ is bounded by a polynomial in n
2. Consistent 3-CNF$_n$ finding can efficiently be done

(II)
$\Rightarrow$

1. Size of 3-term DNF$_n$ is bounded by a polynomial in n
2. Consistent 3-term DNF$_n$ finding can efficiently be done

Corollary 1.2
$\Rightarrow$

3-term-DNF$_n$ is polynomial PAC-learnable using 3-CNF$_n$

Lemma 2.14:

Let: $\mathcal{C}$ be the set of all half-spaces of $\mathbb{R}^d$

Then: $VC_{\dim}(\mathcal{C}) = d + 1.$

Proof:

Step 1: Show $VC_{dim}(\mathcal{C}) \leq d+1$

Suppose: $\exists\ S \subseteq \mathbb{R}^d$ with $|S| = d + 2$: S can be shattered by $\mathcal{C}$

Radon's theorem: there exist S_1 and S_2 with $\text{Conv}(S_1) \cap \text{Conv}(S_2) \neq \emptyset$

$\Rightarrow$ for any half-space H:

$$\text{if } S_1 \subseteq H \text{ then } H \cap \text{Conv}(S_2) \neq \emptyset$$

$\Rightarrow$ $H \cap S_2 \neq \emptyset$

$\Rightarrow$ S_1 can't be realized by a half-space

$\Rightarrow$ S can't be shattered by $\mathcal{C}$; *contradiction*

Step 2: Show $VC_{dim}(\mathcal{C}) \geq d+1$

Explicitly define S with $|S| = d+1$ that can be shattered by $\mathcal{C}$

let $S = \{\vec{0}, \vec{e}_1, \ldots, \vec{e}_d\} \subseteq \mathbb{R}^d$ with

- $\vec{0}$: null vector
- $\vec{e}_i$: i-th standard basis vector

for any $T \subseteq S$, the half-space $\vec{w}^\top \vec{x} \geq \theta$ contains exactly the elements of T from S, where

$$w_i = \begin{cases} 1 & \text{if } \vec{e}_i \in T \\ -1 & \text{o/w} \end{cases}$$

$$\theta = \begin{cases} 0 & \text{if } \vec{0} \in T \\ 1 & \text{o/w} \end{cases}$$

q.e.d.

Theorem 2.15:

$MONOM_n$ is efficiently PAC-learnable

Proof:

Given Find-S algorithm:

1: Draw m examples by calling the oracle $Ex(c, D)$
2: SET $h = v_1 \bar{v}_1 v_2 \bar{v}_2 \ldots v_n \bar{v}_n$
3: FOR $i = 0, 1, \ldots, n$:
4: | IF $c(\vec{x}_i) \neq h(\vec{x}_i)$:
5: | | Remove all literals from h that are false in $\vec{x}_i$
6: RETURN h

Observations

- h is the most specific hypothesis consistent with the examples
- literals in the target concept are never deleted, but h may include some additional literals

$\Rightarrow$ some instances of c are excluded

Goal: Show that the combined probability is not greater than ε

Define $p(z) \longrightarrow$ probability that the oracle returns a positive example with literal z being false

$\rightarrow$ probability z gets removed

Claim 1: $error(h) = \sum_{z \in h} p(z)$

Definition: Literal z is bad, if: $p(z) \geq \frac{\varepsilon}{2n}$

Claim 2: If h has no bad literals: $error(h) < \varepsilon$

Probability that a bad literal remains in h after m examples:

Let z be a bad literal:

$$\begin{aligned} \Pr[z \text{ survives one example}] &= 1 - \Pr[z \text{ is deleted by one example}] \\ &= 1 - p(z) \\ &\leq 1 - \frac{\epsilon}{2n} \\ \Pr[z \text{ survives } m \text{ examples}] &\leq \left(1 - \frac{\epsilon}{2n}\right)^m \end{aligned}$$

$$\Rightarrow \Pr(\text{some bad literal survives } m \text{ examples}) \leq 2n \left(1 - \frac{\epsilon}{2n}\right)^m$$

For large enough m:

$$\Pr(\text{some bad literal survives } m \text{ examples}) \leq 2n \left(1 - \tfrac{\epsilon}{2n}\right)^m \leq \delta \qquad (*)$$

Applying the inequality $1 - x \leq e^{-x}$, any value of m satisfying

$$2ne^{-\frac{m\epsilon}{2n}} \leq \delta$$

also satisfies condition (*) above, and thus we have

$$m \geq \tfrac{2n}{\epsilon} \left(\ln 2n + \ln \tfrac{1}{\delta}\right)$$

Optimization Theory

Constrained Optimization Problem:

$$\begin{aligned} \min_{\vec{w}} \quad & f(\vec{w}) \\ \text{s.t.} \quad & g_i(\vec{w}) \leq 0 \quad (i = 1, \ldots, m) \end{aligned}$$

Definition: Generalized Lagrangian

Represents optimization function and constraints in one function

$$L(\vec{w}, \vec{\alpha}) = f(\vec{w}) + \sum_{i=1}^{m} \alpha_i g_i(\vec{w})$$

α_i's: *Lagrange multipliers*; $\vec{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$

Primal Optimization Problem.

$$\min_{\vec{w}} \underbrace{\max_{\vec{\alpha} \geq \vec{0}} L(\vec{w}, \vec{\alpha})}_{L_{\mathcal{P}}(\vec{w})}$$

$$L_{\mathcal{P}}(\vec{w}) = \begin{cases} f(\vec{w}) & \text{if } \vec{w} \text{ satisfies all the constraints} \\ \infty & \text{o/w} \end{cases}$$

$\Rightarrow \min_{\vec{w}} L_{\mathcal{P}}(\vec{w}) = \min_{\vec{w}} \max_{\vec{\alpha} \geq \vec{0}} L(\vec{w}, \vec{\alpha})$: The primal optimization problem is the same as the initial problem

Dual Optimization Problem:

$$\max_{\vec{\alpha} \geq \vec{0}} \underbrace{\min_{\vec{w}} L(\vec{w}, \vec{\alpha})}_{L_{\mathcal{D}}(\vec{\alpha})}$$

Theorem 2.16: Weak Duality

For $d^* \leq p^*$, where:

$$\begin{aligned} d^* &= \max_{\vec{\alpha} \geq \vec{0}} \min_{\vec{w}} L(\vec{w}, \vec{\alpha}) \\ p^* &= \min_{\vec{w}} \max_{\vec{\alpha} \geq \vec{0}} L(\vec{w}, \vec{\alpha}) \end{aligned}$$

Theorem 2.17: Strong Duality

If: (i) f and the g_i's are *convex* and

(ii) the g_i's are *strictly feasible* (i.e., $\forall i \; \exists \vec{w}$ s.t. $g_i(\vec{w}) < 0$),

Then:

1. $\exists$ optimal solution for $\vec{w}^*$ and $\vec{\alpha}^*$
2. $L_{\mathcal{P}}(\vec{w}^*) = L_{\mathcal{D}}(\vec{\alpha}^*) = L(\vec{w}^*, \vec{\alpha}^*)$
3. $\vec{w}^*, \vec{\alpha}^*$ satisfy the Karush-Kuhn-Tucker conditions

(1) $\frac{\partial}{\partial w_i} L(\vec{w}^*, \vec{\alpha}^*) = 0$

(2) $\alpha_i^* g_i(\vec{w}^*) = 0$

(3) $g_i(\vec{w}^*) \leq 0$

(4) $\alpha_i^* \geq 0$

$\Rightarrow$ Solution of the dual problem is also solution for the initial problem

3. Learning Algorithms

Example Tasks:

Sky	Temp	Humid	Wind	Water	Forecast	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Hypothesis Space: Conjunctions

Hypothesis h: a conjunction of constraints on attributes

Sky	AirTemp	Humid	Wind	Water	Forecst
$\langle Sunny$	$?$	$?$	$Strong$	$?$	$Same\rangle$

Constraints:

- Specific value: e.g. water = "warm"
- Arbitraby: e.g. water = ?
- No value: e.g. water = $\emptyset$

Most specific hypothesis: $h_{\text{ms}} = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

Most general hypothesis: $h_{\text{mg}} = \langle ?, ?, ?, ?, ?, ? \rangle$

Find-S Algorithm

The Find-S algorithm starts with a maximal specific hypothesis and iteratively loosens the constraints such that the hypothesis covers all positive examples. It returns a maximally specific hypothesis disregarding the negative examples.

Base Algorithm:

1. initialize h to the most specific hypothesis in $\mathcal{H}$
2. **forall** positive training examples x **do**
3. **forall** attribute constraints a_i in h **do**
4. **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 satisfied by x
5. **print** h

For Conjunctions:

Require: hypothesis space $\mathcal{H}_n$ of monomials
instance space $X = \{0,1\}^n$
Input: examples $E \subseteq \{0,1\}^n \times \{0,1\}$
Output: hypothesis $h \in \mathcal{H}_n$

1. $h \leftarrow x_1\overline{x_1}x_2\overline{x_2}\ldots x_n\overline{x_n}$
2. **For all** training examples $(x, \ell) \in E$
3. **If** $\ell \neq h(x)$
4. Remove all literals from h that are false by x
5. **Return** h

For Disjunctions:

Requires: hypothesis space $\mathcal{H}_n$ of 1-DNFs
instance space $X = \{0,1\}^n$
Input: examples $E = E^+ \cup E^- \subseteq \{0,1\}^n$
Output: hypothesis $h \in \mathcal{H}_n$

1. $h \leftarrow (\bigvee_{i=1}^n v_i) \vee (\bigvee_{i=1}^n \overline{v_i})$
2. **For all** $e \in E^-$ \\ order matters!
3. **If** $h(e) = 1$
4. Remove all literals from h that are true by e
5. **For all** $e \in E^+$
6. **If** $h(e) = 0$
7. **Return NO**
8. **Return** h

Example:

FIND_S: Example

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$h_1 = \langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$

$h_2 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$

$h_3 = h_2$

$h_4 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? \rangle$

Problems:

- Can't decide if an unknown concept is correctly learned
- Works only, if a consistent hypothesis exists in $\mathcal{H}$

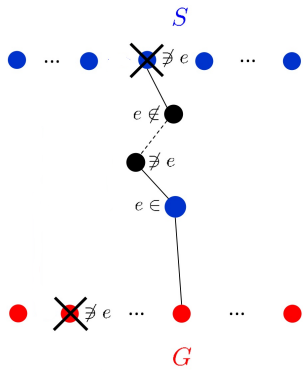
Candidate Elimination Algorithm

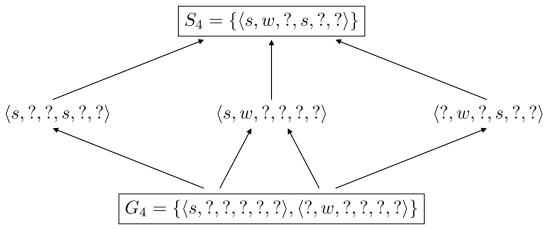
The algorithm computes the general and specific boundary for the version space spanned by $\mathcal{H}$ and E. Iteratively the general and specific boundaries are moved towards each other to define a version space that is consistent with the previous training examples.

Algorithm:

Assumed: hypothesis space $\mathcal{H} \subseteq 2^X$ over instance space X
Input : set $E = E^+ \cup E^- \subseteq X$ of examples
Output : general boundary G and specific boundary S of $\mathrm{VS}_{\mathcal{H},E}$

$G := \{g \in \mathcal{H} :$ there is no $g' \in \mathcal{H}$ with $g' < g\}$
$S := \{s \in \mathcal{H} :$ there is no $s' \in \mathcal{H}$ with $s < s'\}$
forall training examples $e \in E$ **do**
  if $e \in E^+$ **then**
    remove from G all hypotheses that do not cover e
    forall $s \in S$ that do not cover e **do**
      remove s from S
      add to S all generalizations $h \in \mathcal{H}$ of s that satisfy ← take the most specific more general h that covers e
        (i) h covers e,
        (ii) $\exists g \in G$ with $g \leq h$, and
        (iii) $\nexists h' \in \mathcal{H}$ with $h < h' < s$ that satisfies (i) and (ii)
    forall $h \in S$ **do** ← Take most specific h
      if $\exists h' \in S$ with $h < h'$ **then** $S := S \setminus \{h\}$
  if $e \in E^-$ **then**
    remove from S all hypotheses that cover e
    forall hypotheses $g \in G$ that cover e **do**
      remove g from G
      add to G all specializations $h \in \mathcal{H}$ of g that satisfy
        (i) h does not cover e,
        (ii) $\exists s \in S$ with $h \leq s$, and
        (iii) $\nexists h' \in \mathcal{H}$ with $g < h' < h$ that satisfies (i) and (ii)
    forall $h \in G$ **do**
      if $\exists h' \in G$ with $h' < h$ **then** $G := G \setminus \{h\}$





Classification of new instances

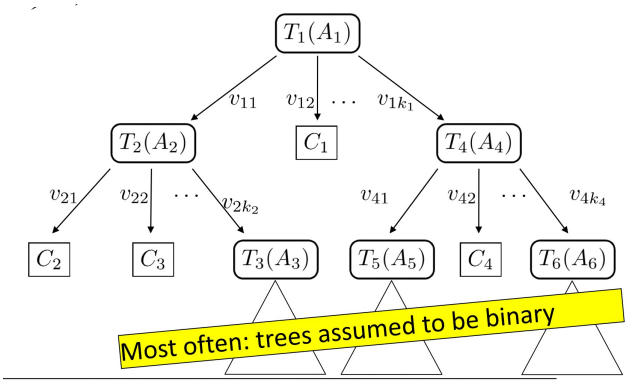
If all hypotheses in the version space agree:
Classify according to the hypotheses

Else:
Reject or Majority vote

Decision Tree Learning

Decision Tree

Internal Nodes: Test an attribute
Branches: values of the test
Leaf Nodes: Assign to classification



Top-Down Induction of Decision Trees

TDIDT is a recursive greedy algorithm. In each recursion the algorithm splits the data on basis of an attribute to classify the data. To find the attribute used in the test, the information gain from each attribute is calculated

Algorithm:

```
0:  TDIDT(E, Atts, T, Node):
1:    IF examples in E are perfectly classified with class c:
2:      Make node a leaf node in T with class c
3:      Return T
4:    IF no remaining test splits the data:
5:      Make node a leaf node in T with the class
        being the majority vote of the examples E
6:      return T
7:    Select a Test from the attributes that
      maximizes the Information Gain
8:    Assign Test to Node
9:    FOR EACH value of Test:
10:     Create a new node D and a new branch in T
11:     Split E, such that E_t corresponds to the
        examples which attributes match the value
12:     TDIDT(E_t, ATTs, T, D)
```

Selection of the Test

The test that maximizes the information gain is chosen.

$\Leftrightarrow$ Minimize the conditional impurity function

For numerical values: Test for specific values

For continuous values: Define a threshold to test for

Algorithm: Search at split time

Sort the examples according to A and test all threshold that are the mean between two subsequent examples that belong to different classes.

Find the best split for continuous attribute A:

1: sort examples covered by the current node according to the values of A
2: **forall** adjacent examples with different classes **do**
3: take the mean m of the values of A for the two examples
4: calculate the information gain for the split $A \leq m$ for the current node
5: **return** $A \leq c$ that maximizes the information gain

Definition: Information Gain

Gain of information, if we split the subset of examples S by attribute A

$$\text{Gain(S,A)} = H(S) - H(S|A)$$

Gives an indication of the impurity which is intended to be minimized during decision tree construction

$H(S)$: Entropy/Impurity function

$H(S|A)$: Conditional Entropy/Impurity function

Definition: Impurity Function

Let: $p_\oplus = \frac{\text{number of positive examples}}{m}$

$p_\ominus = \frac{\text{number of negative examples}}{m}$

An impurity function $f : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ satisfies:

(i) $f(p_\oplus, p_\ominus) = f(p_\ominus, p_\oplus)$ for all $p_\oplus, p_\ominus \geq 0$ with $p_\oplus + p_\ominus = 1$

(ii) $\underset{p_\oplus, p_\ominus \geq 0, p_\oplus + p_\ominus = 1}{\arg\max} f(p_\oplus, p_\ominus) = (1/2, 1/2)$

(iii) $\underset{p_\oplus, p_\ominus \geq 0, p_\oplus + p_\ominus = 1}{\arg\min} f(p_\oplus, p_\ominus) \in \{(0,1), (1,0)\}$

Definition: Entropy

Measure of mean uncertainty

$$H(S) = \sum_i p_i \log_2 \frac{1}{p_i}$$

Derivation: **Shannon information content:** Let the outcomes (i.e., elementary events) of a probabilistic experiment be $a_1, \ldots, a_k$ with respective probabilities $p_1, \ldots, p_k$ (i.e., $\sum p_i = 1$). Then the information amount supplied by a_k is

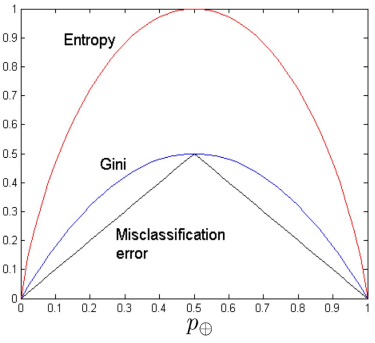
$$I(a_k) = \log_2 \frac{1}{p_k}$$

$\Rightarrow$ outcomes of low probabilities supply much more information than outcomes of high probabilities

Definition: Conditional Entropy

Measure of mean uncertainty given the result from A

$$H(S|A) = \sum_{j=1}^{m} P(\text{value of } A = v_j) H(S_j)$$



Theorem 3.1: $H(X) = \sum_{i=1}^{n} -p_i \log_2 p_i \leq \log_2 n$

Let $X : \Omega \to V$ be a discrete random variable over a finite set $V = \{v_1, v_2, \ldots, v_n\}$ with distribution $p_i := \mathbb{P}(X = v_i)$ for all $i \in [n]$. Then Hartley's formula yields an upper bound for the entropy of X: $H(X) \leq \log_2 n$.

Proof:

Define: $P = \{p_1, p_2, \ldots, p_n\}$ $\quad |P| < n$ possible!

$g : V \to P$ by $g(v_i) = p_i$ for all $i \in [n]$

$I_p = \{i \in [n] : p_i = p\}$ for all $p \in P$

Step 0: $\mathbb{P}\left(\frac{1}{g(X)} = \frac{1}{p}\right) = |I_p| p$ for all $p \in P$

Note for all $p \in P$: $\frac{1}{g(X)} = \frac{1}{p} \iff g(X) = p \iff X \in \{v_i : i \in I_p\}$

Hence for all $p \in P$:

$$\mathbb{P}\left(\frac{1}{g(X)} = \frac{1}{p}\right) = \mathbb{P}(X \in \{v_i : i \in I_p\}) = \sum_{i \in I_p} \underbrace{\mathbb{P}(X = v_i)}_{p} = |I_p| p$$

Step 1: $\mathbb{E}\left[\frac{1}{g(X)}\right] = n$

$$\mathbb{E}\left[\frac{1}{g(X)}\right] = \sum_{p \in P} \frac{1}{p} \mathbb{P}\left(\frac{1}{g(X)} = \frac{1}{p}\right) \qquad \text{def. of expectation}$$

$$= \sum_{p \in P} \frac{1}{p} |I_p| p = \sum_{p \in P} |I_p| = n \qquad \text{Step 0}$$

Theorem 3.2: $H(X|Y) \leq H(X)$ with equality if X and Y are independent

Definition: Gini Gain

$$\text{Gain}_{\text{Gini}}(S, A) = \text{Gini}(S) - \text{Gini}_{\text{split}}(S, A)$$

Definition: Gini Coefficient

Probability of incorrect labeling of a randomly chosen element, if the element was labeled randomly according to the class distributions

$$\begin{aligned} \text{Gini}(S) &= p_{\oplus}(1 - p_{\oplus}) + p_{\ominus}(1 - p_{\ominus}) \\ &= 1 - (p_{\oplus}^2 + p_{\ominus}^2) \end{aligned}$$

Definition: Gini Split

Weighted average Gini index of the children S according to A

$$\text{Gini}_{\text{split}}(S, A) = \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Gini}(S_v)$$

Prevent Overfitting

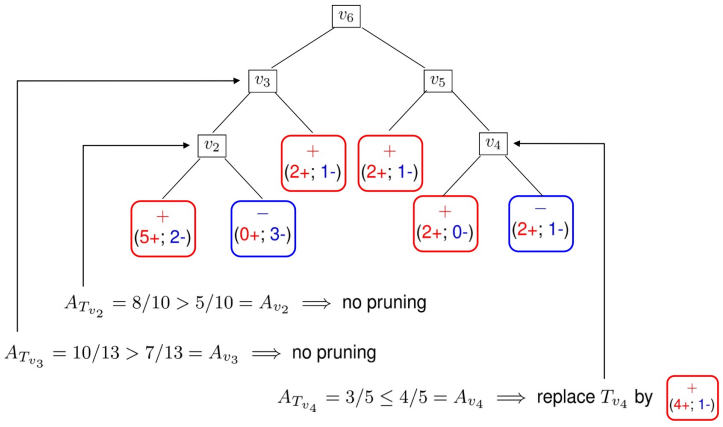
1. **Prepruning**: Stop growing when data split is not statistically significant

2. **Postpruning**: Grow full tree and then prune
   → Remove dependency on statistical noise or variation only related to the training data

Reduced Error Pruning:

After building the tree, prune the subtrees that have a lower accuracy than the tree pruned of the given subtree.

1: Split data in training set S_1 and validation set S_2
2: Build full tree T
3: FOR EACH internal node v in post-order traversal:
4: | Compute accuracy A_{T_v} of the subtree T_v rooted at v on S_2
5: | Compute accuracy A_T of the tree T with pruned subtree T_v on S_2
6: | IF $A_{T_v} \leq A_T$:
7: | | Replace T_v in T with a leaf that contains the majority vote of S_2



SLIQ: A Large-Scale Decision Tree Learning Algorithm

The SLIQ algorithm builds a decision tree in a breadth-first manner. It uses a data structure that divides the attributes in multiple attribute lists and the class labels in a class list. In one traversal over the attribute list the best split is found for each leaf node by maximizing the Gini gain.

Data Structure

1. **Class List** for the class labels

   Each entry consists of a class label and a leaf node

   Each leaf node has a class histogram

   For numeric attributes:

   list of ⟨class, frequency⟩

   For categorical attributes.

   list of ⟨value, class, frequency⟩

2. **Attribute List** for each attribute

   Each entry consists of a attribute value and a reference to the entry in the class list

   Numerical attributes are sorted

class list:

Class	Leaf
Yes	N_1
No	N_1
No	N_1
No	N_1
Yes	N_1
Yes	N_1
No	N_1
Yes	N_1
Yes	N_1
Yes	N_1
Yes	N_1
Yes	N_1
No	N_1
Yes	N_1

Attribute Lists:

Outlook	Class List Index
Rain	1
Rain	2
OVC	3
Sunny	4
OVC	5
Sunny	6
Sunny	7
OVC	8
Rain	9
Sunny	10
OVC	11
Rain	12
Rain	13
Sunny	14

TEMP	Class List Index
10	10
12	13
14	4
16	2
18	1
18	9
20	5
22	11
24	14
26	12
28	8
30	3
32	7
34	6

HUM	Class List Index
24	6
26	7
28	10
30	3
32	4
32	11
40	13
42	14
44	12
46	8
48	9
50	1
56	2
60	5

Algorithm:

0: UNTIL all leaf nodes are perfectly classified:

Split Evaluation:

1: FOR EACH attribute A:

2: FOR EACH value v in the attribute list of A:

3: Find the corresponding entry in the class list and leaf L

4: Update class histogram in L

5: IF A is a numeric attribute:

6: Compute splitting index for $A \leq v^*$ for leaf L

Option 1: $v^* = v$

→ initially used in the original

Option 2: $v^* = \begin{cases} v & \text{if } v \text{ is the maximum in the attribute list of } A \\ \frac{v+v'}{2} & \text{o/w} \end{cases}$

7: IF A is a categorical attribute.

8: FOR EACH leaf L of the tree:

9: Find best split by maximizing the Gini gain

Class List Update:

10: FOR EACH attribute A used in a split:

11: FOR EACH value v of attribute A.

12: Find corresponding class entry e

13: Find new node n which v belongs to according to the splitting criterion

14: Update entry e to n

4. Kernel Methods

Kernel Methods: Kernel methods are a class of pattern detecting algorithms

Motivation: Data is not linearly separable in a low-dimensional instance space. Thus embed the points in a high dimensional feature space in which the points are linearly separable
→ Cover's Theorem

Key Aspects:

Feature Space: High dimensional inner product space
→ patterns are linear in feature space

Algorithm: The algorithm only relies on the pairwise inner product of the points in the feature space.

Kernel: Without the knowledge about the actual embedding function, a kernel function computes the pairwise inner product of the points embedded in the feature space

Definition: General Position

A set $S \in \mathbb{R}^d$ of $n > 1$ points is in general position, if:

- $n \leq d$: S is not contained by a $(n-2)$-dimensional affine hyperplane
- $n > d$: S has no subset of $d+1$ points belonging to a $(d-1)$-dimensional affine hyperplane

→ e.g. for $\mathbb{R}^2$, points are not colinear

Definition: Kernel

A kernel is a function $k: X \times X \longrightarrow \mathbb{R}$, s.t. 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:

Calculating Φ might be prohibitively expensive.
Instead use kernel function to compute inner products by replacing: $k(x,y) = \langle \Phi(x), \Phi(y) \rangle$

Cosine Similarity in the feature space

Let: kernel function $k : X \times X \to \mathbb{R}$ with $k(x,y) = \langle \Phi(x), \Phi(y) \rangle$

Cosine Similarity = Normalized kernel: $\alpha(x,y) = \dfrac{k(x,y)}{\sqrt{k(x,x)}\sqrt{k(y,y)}}$

Proof:

$$\begin{aligned}
\alpha(x,y) &= \frac{k(x,y)}{\sqrt{k(x,x)}\sqrt{k(y,y)}} \\
&= \frac{\langle \Phi(x), \Phi(y) \rangle}{\sqrt{\langle \Phi(x), \Phi(x) \rangle}\sqrt{\langle \Phi(y), \Phi(y) \rangle}} \\
&= \frac{\langle \Phi(x), \Phi(y) \rangle}{\|\Phi(x)\| \|\Phi(y)\|} \\
&= \cos(\Phi(x), \Phi(y))
\end{aligned}$$

Characterization of Kernels

Definition: Gram/Kernel Matrix $K_{X,k}$

The Gram or kernel matrix $K_{X,k}$ of X w.r.t k is a $m \times n$ matrix with: $(\mathbf{K}_{X,k})_{ij} = k(x_i, x_j)$ for all $i, j = 1, \ldots, n$

Theorem 3.2: Finite Domain

Let: $k: X \times X \to \mathbb{R}$ symmetric function

k is a kernel $\iff$ $\mathbf{K}_{X,k}$ is positive semi-definite

Proof:

"$\Leftarrow$": Suppose $K_{X,k}$ is positive semi-definite

Spectral Decomposition: $\mathbf{K}_{X,k} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$: $\lambda_i = \mathbf{\Lambda}_{ii} \geq 0$ for all i

$\Rightarrow$ $\Phi : X \to \mathbb{R}^n$ with $\Phi : x_i \mapsto (\sqrt{\lambda_1} u_{i,1}, \ldots, \sqrt{\lambda_n} u_{i,n})^\top$ exists

$$\Rightarrow \langle \Phi(x_i), \Phi(x_j) \rangle = \sum_{l=1}^{n} \lambda_l u_{i,l} u_{j,l} = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)_{ij} = (\mathbf{K}_{X,k})_{ij} = k(x_i, x_j)$$

"⇒": Let: kernel k $k(x,y) = \langle \Phi(x), \Phi(y) \rangle$ for some $\Phi: X \to \mathcal{H}$

Assumption: $K_{X,k}$ is not positive semi-definite

$\Rightarrow$ $\mathbf{K}_{X,k} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ has a negative eigenvalue λ_i

For $\vec{v} = \sum_{j=1}^{n} u_{i,j}\Phi(x_j)$:

$$\begin{aligned}
\langle \vec{v}, \vec{v} \rangle &= \left\langle \sum_{j=1}^{n} u_{i,j}\Phi(x_j), \sum_{l=1}^{n} u_{i,l}\Phi(x_l) \right\rangle \\
&= \sum_{j,l=1}^{n} u_{i,j}\langle \Phi(x_j), \Phi(x_l) \rangle u_{i,l} \\
&= \sum_{j,l=1}^{n} u_{i,j} k(x_j, x_l) u_{i,l} \\
&= \vec{u}_i^\top \mathbf{K}_{X,k} \vec{u}_i \\
&= \lambda_i < 0 \qquad \text{contradiction}
\end{aligned}$$

Theorem 4.2: Infinite Domain (Mercer's Theorem)

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

k is a kernel $\iff$ for all **finite** $Y \subseteq X$: $\mathbf{K}_{Y,k}$ is positive semi-definite

Proof: omitted → follows directly from Mercer's Theorem

Theorem 4.3: Cover's (Function Counting) Theorem

Number of linearly separable dichotomies of n points in general position in $\mathbb{R}^d$:

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

Remark: If $\mathcal{H}$ is the set of half-spaces in $\mathbb{R}^d$, then $C(n,d) = \Pi_{\mathcal{H}_d}(n)$

Probability of Linear Separability

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

$$P(n,d) = \frac{1}{2^n}C(n,d)$$

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

if $n \gg d$: $P(n,d) \sim \frac{cn^d}{2^n}$ for some constant $c > 0$

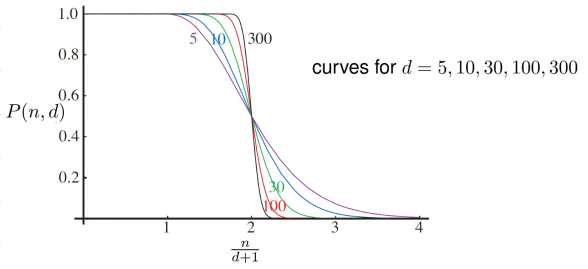
Phase Transition at: $n = 2(d+1)$

→ probability of linear separability is close to 1 at that point

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

$$P(2(d+1),d) = \frac{1}{2}$$

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



Implication:

If training data is not linearly separable, it is highly likely that it can be embedded in a higher dimensional feature space in which the data is linearly separable.

Kernel Functions

Proof techniques:

1. Construct underlying feature map ϕ corresponding to the kernel
2. Use Mercer's Theorem

1. $k(x,y) = f(x)f(y)$ kernel over $X \times X$ for all functions $f: X \to \mathbb{R}$

Proof:

Construct ϕ: $\phi: X \to \mathbb{R}$, $x \mapsto f(x)$ for all $x \in X$

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

2. $k(\vec{x},\vec{y}) = \vec{x}^\top \mathbf{A}\vec{y}$ for all $\vec{x},\vec{y} \in \mathbb{R}^n$ kernel over $\mathbb{R}^n \times \mathbb{R}^n$, $\mathbf{A}$ positive semi-definite matrix

Proof:

Spectral Theorem: $\exists_{\Lambda,U} : A = U^T \Lambda U$

$\Rightarrow$ All entries of Λ are positive | A positive semi-definite

Define: $B = \sqrt{\Lambda}\, U$ and $\phi: \vec{x} \mapsto B\vec{x}$

$\Rightarrow k(\vec{x},\vec{y}) = \vec{x}^\top \mathbf{A}\vec{y} = \vec{x}^\top \mathbf{U}^\top \mathbf{\Lambda} \mathbf{U}\vec{y} = \vec{x}^\top \mathbf{B}^\top \mathbf{B}\vec{y} = \langle \mathbf{B}\vec{x}, \mathbf{B}\vec{y} \rangle = \langle \Phi(\vec{x}), \Phi(\vec{y}) \rangle$

3. $k(x,y) = \alpha k_1(x,y) + \beta k_2(x,y)$ $\alpha, \beta \geq 0$ Kernel that is the linear combination of two kernels k_1, k_2 over $X \times X$

Proof:

For all finite $Y = \{x_1, \ldots, x_n\} \subseteq X$ and for all $\vec{a} \in \mathbb{R}^n$: | Mercer's Theorem

$\vec{a}^\top \mathbf{K}_{Y,k_1} \vec{a} \geq 0$ and $\vec{a}^\top \mathbf{K}_{Y,k_2} \vec{a} \geq 0$

Mercer's Theorem: it suffices to show $\vec{a}^\top(\alpha \mathbf{K}_{Y,k_1} + \beta \mathbf{K}_{Y,k_2})\vec{a} \geq 0$

$$\begin{aligned} \Rightarrow \vec{a}^\top(\alpha \mathbf{K}_{Y,k_1} + \beta \mathbf{K}_{Y,k_2})\vec{a} &= \vec{a}^\top(\alpha \mathbf{K}_{Y,k_1})\vec{a} + \vec{a}^\top(\beta \mathbf{K}_{Y,k_2})\vec{a} \\ &= \alpha \vec{a}^\top \mathbf{K}_{Y,k_1}\vec{a} + \beta \vec{a}^\top \mathbf{K}_{Y,k_2}\vec{a} \\ &\geq 0 \end{aligned}$$

4. $k(x,y) = k_1(x,y)k_2(x,y)$ For kernels k_1, k_2 over $X \times X$

Proof: Mercer's Theorem: Show that $K_{Y,k}$ is positive semi-definite

(i) $K_{Y,k}$ is the Hadamard product of K_{Y,k_1} and K_{Y,k_2}

(ii) K_{Y,k_1} and K_{Y,k_2} are positive semi-definite

$\Rightarrow$ Kronecker product of K_{Y,k_1} and K_{Y,k_2} is positive semi-definite

$K_{Y,k}$ is a principal matrix of $K_{Y,k_1} \otimes K_{Y,k_2}$

$\Rightarrow K_{Y,k}$ is positive semi-definite

5. $k(x,y) = p(k_1(x,y))$, for kernel k_1 over $X \times X$ and polynomial p without negative terms

Proof: Follows directly from 1., 3. and 4.

6. $k(x,y) = f(k_1(x,y))$, for kernel k_1 over $X \times X$ and let $f: \mathbb{R} \to \mathbb{R}$ be a function expressed as a power series

Proof:

$f = \lim_{n\to\infty} p_n$ for polynomials without negative terms

$\Rightarrow$ For any finite $Y \subseteq X$: $\mathbf{K}_{Y,f(k_1)} = \lim_{n\to\infty} \mathbf{K}_{Y,p_n(k_1)}$

From 5.: k_{Y,p_n} is positive semi-definite

$$\Rightarrow \quad \vec{v}^\top \mathbf{K}_{Y,f(k_1)} \vec{v} = \vec{v}^\top \left(\lim_{n\to\infty} \mathbf{K}_{Y,p_n(k_1)} \right) \vec{v} = \lim_{n\to\infty} \vec{v}^\top \mathbf{K}_{Y,p_n(k_1)} \vec{v} \geq 0$$

7. $k(x,y) = e^{k_1(x,y)}$, for kernel k_1 over $X \times X$

Proof: Follows directly from 6., with: $e^{k_1(x,y)} = \sum_{n=0}^{\infty} \frac{k_1(x,y)^n}{n!}$

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

Proof:

For $k'(\vec{x},\vec{y}) = \exp(\langle \vec{x},\vec{y} \rangle / \sigma^2)$:

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

$$= k(\vec{x},\vec{y})$$

follows from 1., 2., 7. — follows from 1 — closed under 4.

Common kernels for $\mathbb{R}^d \times \mathbb{R}^d$:

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

polynomial kernel: $k(\vec{x},\vec{y}) := (\vec{x}^\top \vec{y} + c)^m$

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

Ridge Regression

Least Squares optimization with an additional L2-Tikhonov regularization term that penalizes large weights.

Problem:

Given: Instance space $\mathcal{X} = \mathbb{R}^d$, Target space $\mathcal{Y} = \mathbb{R}$,
Samples $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$,
Target vector $\vec{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$,

Data matrix $X = \begin{bmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix} \in \mathbb{R}^{n \times d}$

Find: a linear function:

$$f(\vec{x}) = \vec{x}^\top \vec{w} = \sum_{i=1}^{d} x_i w_i = \langle \vec{x}, \vec{w} \rangle$$

, that best interpolates S

Loss Function: $V(f(\vec{x}), y) = (f(\vec{x}) - y)^2$
→ Squared loss

Empirical Risk: $R_{\text{emp}}[f] = \frac{1}{n} \|\mathbf{X}\vec{w} - \vec{y}\|^2$

Proof:

$$\begin{aligned} R_{\text{emp}}[f_{\vec{w}}] &= \frac{1}{n} \sum_{i=1}^{n} V(f_{\vec{w}}(\vec{x}_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^{n} (f_{\vec{w}}(\vec{x}_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (\langle \vec{x}_i, \vec{w} \rangle - y_i)^2 \\ &= \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \end{aligned}$$

Optimization Function: $\min_{\vec{w}} R_\lambda(\vec{w}, S) = \min_{\vec{w}} \|\mathbf{X}\vec{w} - \vec{y}\|^2 + \lambda\|\vec{w}\|^2$

Convex optimization function in $\vec{w}$

Gradient: $\nabla_{\vec{w}} R_\lambda(\vec{w}, S) = 2\mathbf{X}^\top\mathbf{X}\vec{w} - 2\mathbf{X}^\top\vec{y} + 2\lambda\vec{w}$

Proof:

1. Expand $R_\lambda(\vec{w}, S)$

$$\begin{aligned}
R_\lambda(\vec{w}, S) &= \|X\vec{w} - \vec{y}\|^2 + \lambda\|\vec{w}\|^2 && \text{(def. of } R_\lambda(\vec{w}, S)\text{)} \\
&= \langle X\vec{w} - \vec{y}, X\vec{w} - \vec{y}\rangle + \lambda\|\vec{w}\|^2 && \text{(prop. of } \|.\|\text{)} \\
&= \langle X\vec{w}, X\vec{w}\rangle - 2\langle X\vec{w}, \vec{y}\rangle + \|\vec{y}\|^2 + \lambda\|\vec{w}\|^2 && \text{(bilinearity of } \langle .,.\rangle\text{)} \\
&= \underbrace{(X\vec{w})^\top(X\vec{w})}_{①} - \underbrace{2\vec{y}^\top(X\vec{w})}_{②} + \|\vec{y}\|^2 + \underbrace{\lambda\|\vec{w}\|^2}_{③} && (\langle\vec{a}, \vec{b}\rangle = \vec{b}^\top\vec{a})
\end{aligned}$$

2. Solve terms separately (∇ is linear)

Term 1: $\nabla_{\vec{w}}((X\vec{w})^\top(X\vec{w})) = \nabla_{\vec{w}}\left(\sum_{i=1}^{n}\langle\vec{x}_i, \vec{w}\rangle^2\right) \overset{(*)}{=} 2X^\top X\vec{w}$

(*) $\frac{\partial}{\partial w_k}\left(\sum_{i=1}^{n}\langle\vec{x}_i, \vec{w}\rangle^2\right) = \sum_{i=1}^{n} 2\langle\vec{x}_i, \vec{w}\rangle\frac{\partial}{\partial w_k}\left(\sum_{j=1}^{d} x_{ij}w_j\right) = 2\sum_{i=1}^{n}\langle\vec{x}_i, \vec{w}\rangle x_{ik} = 2\sum_{i=1}^{n}(X\vec{w})_i x_{ik}$

Term 2: $\nabla_{\vec{w}}(2\vec{y}^\top(X\vec{w})) = 2\nabla_{\vec{w}}((X^\top\vec{y})\vec{w}) = 2\nabla_{\vec{w}}\left(\sum_{i=1}^{d}(X^\top\vec{y})_i w_i\right) = 2X^\top\vec{y}$

Term 3: $\nabla_{\vec{w}}(\lambda\|\vec{w}\|^2) = \lambda\nabla_{\vec{w}}\left(\sum_{i=1}^{d} w_i^2\right) = \lambda(2w_1, 2w_2, \ldots, 2w_d) = 2\lambda\vec{w}$

3. Put together: $\nabla_{\vec{w}} R_\lambda(\vec{w}, S) = 2\mathbf{X}^\top\mathbf{X}\vec{w} - 2\mathbf{X}^\top\vec{y} + 2\lambda\vec{w}$

Solutions

Primal Solution: $\vec{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \vec{y}$

Proof:

Given: $\min_{\vec{w}} R_\lambda(\vec{w}, S) = \min_{\vec{w}} \|\mathbf{X}\vec{w} - \vec{y}\|^2 + \lambda \|\vec{w}\|^2$

1. Calculate gradients: $\nabla_{\vec{w}} R_\lambda(\vec{w}, S) = 2\mathbf{X}^\top \mathbf{X}\vec{w} - 2\mathbf{X}^\top \vec{y} + 2\lambda \vec{w}$
2. Set to zero: $\mathbf{X}^\top \mathbf{X}\vec{w} + \lambda \vec{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)\vec{w} = \mathbf{X}^\top \vec{y}$
3. Build linear equation system: $\vec{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \vec{y}$

Regression Function: $f(\vec{x}) = \vec{x}^\top \vec{w} = \vec{x}^\top (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \vec{y}$

Complexity: computation of $\vec{w}$: $O(d^3)$

evaluation of f: $O(d)$

Dual Solution: $\vec{\alpha} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_n)^{-1} \vec{y}$

Proof:

From primal solution: $\mathbf{X}^\top \mathbf{X}\vec{w} + \lambda \vec{w} = \mathbf{X}^\top \vec{y}$

$$\Rightarrow \vec{w} = \frac{1}{\lambda} (\mathbf{X}^\top \vec{y} - \mathbf{X}^\top \mathbf{X}\vec{w}) = \mathbf{X}^\top \frac{1}{\lambda} (\vec{y} - \mathbf{X}\vec{w}) = \mathbf{X}^\top \vec{\alpha}$$

Dual solution: $\vec{\alpha} = \frac{1}{\lambda} (\vec{y} - \mathbf{X}\vec{w})$

$$\begin{aligned} \Rightarrow \quad \lambda \vec{\alpha} &= \vec{y} - \mathbf{X}\mathbf{X}^\top \vec{\alpha} \\ (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_n)\vec{\alpha} &= \vec{y} \\ \vec{\alpha} &= (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_n)^{-1} \vec{y} \end{aligned}$$

Regression Function: $f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \langle \vec{x}, \vec{x}_i \rangle$

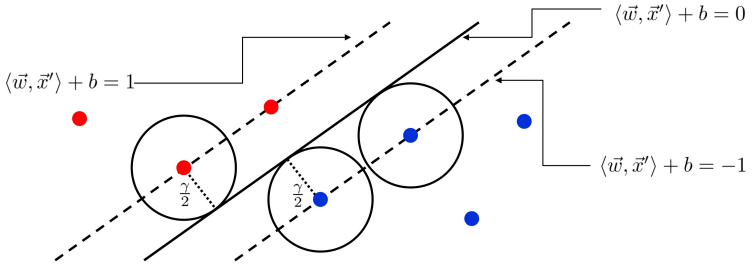
Complexity: computation of $\vec{\alpha}$: $O(n^3)$

evaluation of f: $O(nd)$

→ Application of the kernel trick

Support Vector Machines

Maximum Margin Hyperplane



Hyperplane: $\langle \vec{w}, \vec{x} \rangle + b = 0$

→ $\vec{w}$ orthogonal to hyperplane

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

Signed distance to the hyperplane: $d = \frac{f(\vec{x})}{\|\vec{w}\|}$

Prediction: $\vec{x}$: $\text{sign}(\langle \vec{w}, \vec{x} \rangle + b)$

Motivation of Maximum Margin Hyperplane

1. Robust against noise
2. Excellent predictive performance in practice
3. Separating hyperplane becomes unique
4. Structural risk minimization

→ Margin between risk and empirical risk is minimized

Definition: γ-Shattering

A finite set $S \subseteq \mathbb{R}^d$ is γ-shattered by hyperplanes, if all dichotomies can be realized throug a hyperplane with margin at least γ

Theorem 4.4:

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

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

⇒ bounds the VC dimension with margin γ

⇒ larger margin implies smaller VC dimension

⇒ maximum margin minimizes structural risk

Hard Margin SVM

Hard constraint on the correct classification of data points

For $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\} \subset \mathbb{R}^d \times \{+1, -1\}$, s.t. S is linearly separable

Find a separating hyperplane

Primal Form:

$$\begin{array}{lll} \max\limits_{\vec{w},b} & \frac{2}{\|\vec{w}\|} & \\ \text{subject to} & y_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 \quad \text{for } i = 1, \ldots, n \end{array} \quad \Leftrightarrow \quad \begin{array}{lll} \min\limits_{\vec{w},b} & \frac{1}{2}\|\vec{w}\|^2 & \\ \text{s.t.} & -(y_i(\langle \vec{w}, \vec{x}_i \rangle + b) - 1) \leq 0 & \text{for } i = 1, \ldots, n \end{array}$$

Hard margin constraints:

$$\begin{aligned} \langle \vec{w}, \vec{x}_i \rangle + b &\geq +1 \text{ if } y_i = +1 \\ \langle \vec{w}, \vec{x}_i \rangle + b &\leq -1 \text{ if } y_i = -1 \end{aligned}$$

→ convert to dual form to make use of kernel methods

Generalized Lagrangian:

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w}^\top \vec{x}_i + b) - 1)$$

Dual Form:

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and} \\ & \alpha_i \geq 0, i = 1, \ldots, n \end{aligned}$$

Optimization Function:

$$L_D(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle$$

Derivation: Derive solution for $\min\limits_{\vec{w},b} L(\vec{w}, b, \vec{\alpha})$

1. Define Lagrangian:

$$\begin{array}{lll} \max\limits_{\vec{\alpha}} & \underbrace{\min\limits_{\vec{w},b} L(\vec{w}, b, \vec{\alpha})}_{L_D} & \\ \text{s.t.} & \alpha_i \geq 0 & \text{for } i = 1, \ldots, n \end{array}$$

2. Take partial derivatives w.r.t. $\vec{w}$ and b

$$\frac{\partial L(\vec{w}, b, \vec{\alpha})}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i = 0 \quad \Rightarrow \quad \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$$

$$\frac{\partial L(\vec{w}, b, \vec{\alpha})}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Rightarrow \quad 0 = \sum_{i=1}^{n} \alpha_i y_i$$

3. Plug into generalized Lagrangian

$$L_D(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle$$

Maximum Margin Hyperplane:

$$f(\vec{x}) = \sum_{i=1}^{n} y_i \alpha_i \langle \vec{x}_i, \vec{x} \rangle + b$$

With:

$$b = -\frac{1}{2} \left(\max_{j:y_j=-1} \left(\sum_{i=1}^{n} y_i \alpha_i \langle \vec{x}_i, \vec{x}_j \rangle \right) + \min_{j:y_j=1} \left(\sum_{i=1}^{n} y_i \alpha_i \langle \vec{x}_i, \vec{x}_j \rangle \right) \right), \quad \vec{w} = \sum_{i=1}^{n} y_i \alpha_i \vec{x}_i$$

Support Vectors: $\alpha_i > 0$

Remarks:

1. Corresponds to the regularized empirical risk minimization:

$$\min_{\vec{w},b} \frac{1}{n} \sum_{i=1}^{n} V(f(\vec{x}_i), y_i) + \lambda \|\vec{w}\|^2$$

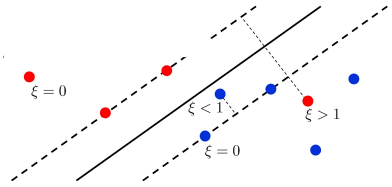
loss function: $V(f(\vec{x}), y) = \begin{cases} 0 & \text{if } y \cdot f(\vec{x}) \geq +1 \\ \infty & \text{o/w} \end{cases}$

regularization parameter: any $0 < \lambda < \infty$ results in the same solution

2. For prediction only the inner product with the support vectors is required

Soft Margin SVM

Relaxes the constraints and allows false classifications, if the data is not linearly separable



Soft Constraints:

Violations:

(I) Example in margin region, but on correct side
(II) Example on the wrong side

Constraints: $y_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{for} \quad \xi_i \geq 0 \quad (i = 1, \ldots, n)$

Introduce slack variable to loosen constraints

Slack variable ξ_i:

$\xi_i = 0$: correct classification

$0 < \xi_i \leq 1$: within margin, but on correct side

$\xi_i > 1$: lies on the wrong side

Primal Form:

$$\begin{aligned} \min_{\vec{w}, b, \vec{\xi}} \quad & C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\vec{w}\|^2 \\ \text{s.t.} \quad & y_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad i = 1, \ldots, n \end{aligned}$$

, for $C \geq 0$

Dual Form:

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad i = 1, \ldots, n \end{aligned}$$

Derivation:

1. Define Lagrangian:

$$L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\lambda}) = C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\langle \vec{x}_i, \vec{w} \rangle + b) - (1 - \xi_i)) - \sum_{i=1}^{n} \lambda_i \xi_i$$

2. Rewrite Lagrangian

$$\begin{aligned} L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\lambda}) &= C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\langle \vec{x}_i, \vec{w} \rangle + b) - (1 - \xi_i)) - \sum_{i=1}^{n} \lambda_i \xi_i \\ &= \sum_{i=1}^{n} C\xi_i - \sum_{i=1}^{n} \alpha_i (-(-\xi_i)) - \sum_{i=1}^{n} \lambda_i \xi_i \\ &\quad - \sum_{i=1}^{n} \alpha_i y_i b - \sum_{i=1}^{n} \alpha_i (-1) \\ &\quad + \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \langle \vec{x}_i, \vec{w} \rangle \\ &= \sum_{i=1}^{n} (C - \alpha_i - \lambda_i) \xi_i - b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i + \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \langle \vec{x}_i, \vec{w} \rangle \end{aligned}$$

3. Take partial derivatives

$$L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\lambda}) = \sum_{i=1}^{n} (C - \alpha_i - \lambda_i)\xi_i - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i + \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \langle \vec{x_i}, \vec{w} \rangle$$

b:

$$\implies \frac{\partial L}{\partial b} = \frac{\partial}{\partial b}\left(-b\sum_{i=1}^{n} \alpha_i y_i\right) = -\sum_{i=1}^{n} \alpha_i y_i$$

Hence: $\frac{\partial L}{\partial b} \stackrel{!}{=} 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0$

$\vec{w}$:

$$\implies \frac{\partial L}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \langle \vec{x_i}, \vec{w} \rangle\right)$$

$$= \frac{\partial}{\partial w_k}\left(\frac{1}{2}\sum_{j=1}^{d} w_j^2 - \sum_{i=1}^{n}\sum_{j=1}^{d} \alpha_i y_i x_{ij} w_j\right) = w_k - \sum_{i=1}^{n} \alpha_i y_i x_{ik}$$

Hence: $\frac{\partial L}{\partial \vec{w}} \stackrel{!}{=} 0 \implies \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}$

ξ_i

$$\implies \frac{\partial L}{\partial \xi_k} = \frac{\partial}{\partial \xi_k}\left(\sum_{i=1}^{n} (C - \alpha_i - \lambda_i)\xi_i\right) = C - \alpha_k - \lambda_k$$

Hence: $\frac{\partial L}{\partial \xi_k} \stackrel{!}{=} 0 \implies C = \alpha_k + \lambda_k \quad (\stackrel{\alpha_k, \lambda_k \geq 0}{\implies} 0 \leq \alpha_i \leq C)$

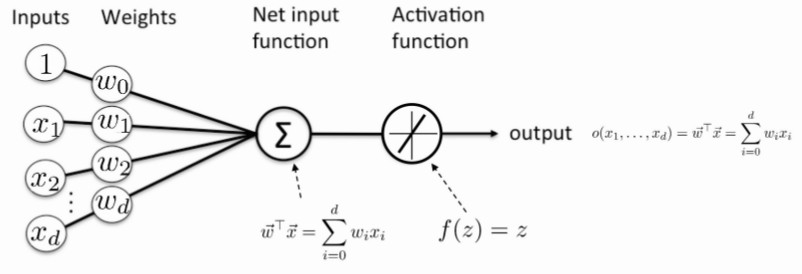
4. Putting together

$$\frac{\partial L}{\partial (\vec{w}, b, \vec{\xi})} \stackrel{!}{=} 0 \implies \textcircled{1}\ \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i} \quad \textcircled{2}\ \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \textcircled{3}\ \forall i \in [n] : C = \alpha_i + \lambda_i$$

$$\begin{aligned} L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\lambda}) &= \sum_{i=1}^{n} (C - \alpha_i - \lambda_i)\xi_i - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i + \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \langle \vec{x_i}, \vec{w} \rangle \\ &= \sum_{i=1}^{n} \alpha_i + \frac{1}{2}\left\langle \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}, \sum_{j=1}^{n} \alpha_j y_j \vec{x_j} \right\rangle - \sum_{i=1}^{n} \alpha_i y_i \left\langle \vec{x_i}, \sum_{j=1}^{n} \alpha_j y_j \vec{x_j} \right\rangle \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \vec{x_i}, \vec{x_j} \rangle \end{aligned}$$

5. Artificial Neural Networks

Perceptron:



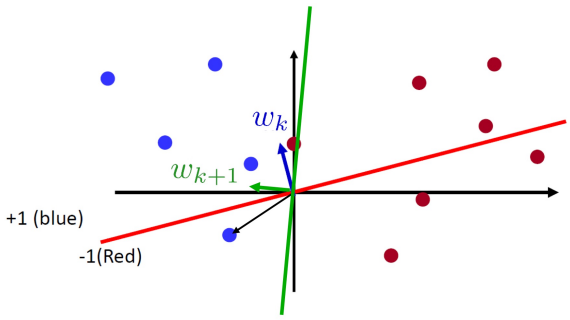
Perceptron Learning Rule:

Iteratively adjust weights to build a separating hyperplane

input: training sample $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \{-1, +1\}$,
number of epochs $T \in \mathbb{N}$

output: $\vec{w} \in R^d$

1. set $k := 0$, $\vec{w}_k := \vec{0}$
2. **repeat**
3. **forall** $(\vec{x}, y) \in D$ do
4. **if** $y(\vec{w_k}^\top \vec{x}) \leq 0$ **then**
5. $\vec{w}_{k+1} := \vec{w}_k + y\vec{x}$ ← perceptron rule
6. $k := k + 1$
7. **endif**
8. **endfor**
9. **until** T iterations reached
10. **return** $\vec{w}_k$



Activation Functions:

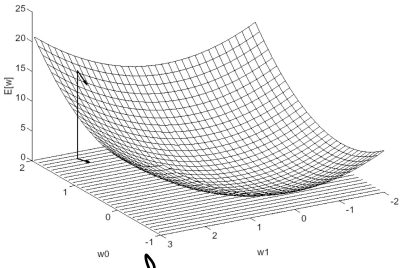
Linear: $f(z) = z$

Sigmoid: $sig(x) = 1/(1 + \exp(-x))$

Rectified linear unit: $ReLU(x) = \max(x, 0.0)$

Hyperbolic tangens: $\tanh(x)$

Gradient Descent:



Delta Rule: $\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$, with: $\Delta\vec{w} = -\eta \nabla E[\vec{w}]$

Gradients: $\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_d}\right]$

Goal: Minimize the error function, e.g. mean squared error

Linear Unit:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_{(\vec{x},y)\in D} \frac{\partial}{\partial w_i}(y - \vec{w}^\top \vec{x})^2 \\
&= \sum_{(\vec{x},y)\in D} (y - \vec{w}^\top \vec{x}) \frac{\partial}{\partial w_i}(y - \vec{w}^\top \vec{x}) \\
&= -\sum_{(\vec{x},y)\in D} (y - \vec{w}^\top \vec{x}) x_i
\end{aligned}$$

Sigmoid Unit:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_{(\vec{x},y)\in D} \frac{\partial}{\partial w_i}(y - o(\vec{x}))^2 \qquad //o(\vec{x}) = \sigma(\text{net}) = \sigma(\vec{w}^\top \vec{x}) \\
&= \sum_{(\vec{x},y)\in D} (y - o(\vec{x})) \frac{\partial}{\partial w_i}(y - o(\vec{x})) \\
&= \sum_{(\vec{x},y)\in D} (y - o(\vec{x})) \left(-\frac{\partial o(\vec{x})}{\partial w_i}\right) \\
&= -\sum_{(\vec{x},y)\in D} (y - o(\vec{x})) \frac{\partial o(\vec{x})}{\partial \text{net}} \frac{\partial \text{net}}{\partial w_i}
\end{aligned}$$

$$\frac{\partial o(\vec{x})}{\partial \text{net}} = \frac{\partial \sigma(\text{net})}{\partial \text{net}} \qquad \frac{\partial \text{net}}{\partial w_i} = \frac{\partial \vec{w}^\top \vec{x}}{\partial w_i}$$

$$= o(\vec{x})(1 - o(\vec{x})) \qquad = x_i$$

$$\Rightarrow \frac{\partial E}{\partial w_i} = -\sum_{(\vec{x},y)\in D} (y - o(\vec{x})) o(\vec{x})(1 - o(\vec{x})) x_i$$

Algorithm: Gradient Descent

Calculates the weight change w.r.t all examples and updates the weights at once

Algorithm GRADIENT-DESCENT(D, η)

- each training example in D is a pair of the form $(\vec{x}, y)$
- η is the learning rate (e.g., .05).

1. initialize each w_i to some *small* random value
2. **until** the termination condition is met **do**
3. **forall** $i = 0, 1, \ldots, d$ **do** $\Delta w_i \leftarrow 0$
4. **forall** $(\vec{x}, y) \in D$ **do**
5. $o := \vec{w}^\top \vec{x}$
6. **forall** $i = 0, 1, \ldots, d$ **do** $\Delta w_i := \Delta w_i + \eta(y - o)x_i$ ---------- delta rule
7. **forall** linear unit weights w_i **do** $w_i := w_i + \Delta w_i$

Algorithm: Stochastic Gradient Descent

Update the weights for each example separately

Algorithm STOCHASTIC GRADIENT-DESCENT(D, η)

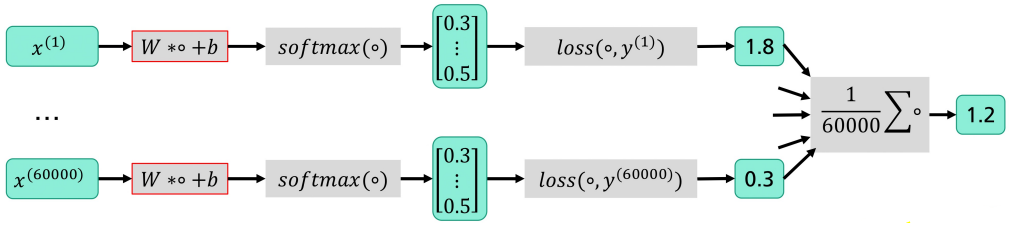
- each training example in D is a pair of the form $(\vec{x}, y)$
- η is the learning rate (e.g., .05).

1. initialize each w_i to some *small* random value
2. **until** the termination condition is met **do**
3. **forall** $(\vec{x}, y) \in D$ **do**
4. let $o = \vec{w}^\top \vec{x}$ be the output of the linear unit on $\vec{x}$
5. **forall** $i = 0, 1, \ldots, d$ **do** $w_i := w_i + \eta(y - o)x_i$

→ can also be used with batches

Logistic Regression Classifier

Given input images of handwritten digits, classify the input to one out of ten class labels



Loss Function: Negative Log-Likelihood

$$L(w) = -\log\big(p(Train|w)\big) = -\sum_{(x^{(1)},y^{(i)})\in Train} \log \hat{p}(y^{(i)}|x^{(i)};w)$$

Algorithm: Batch Gradient Descent

Algorithm Batch Gradient Descent($Train, \eta$)

- Trn $= \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$
- η is the learning rate, e.g. $\eta = 0.1$.
- d_0 is a threshold, e.g. $d_0 = 0.01$

1. initialize $w^{[0]}$ to small random values, set $t = 0$
2. Compute $\nabla L(w^{[t]})$ for current parameter $w^{[t]}$
3. Update $w^{[t+1]} = w^{[t]} - \eta * \nabla L(w^{[t]})$
4. Compute $d_{max} = \max_i \left| w_i^{[t+1]} - w_i^{[t]} \right|$
5. if $d_{max} > d_0$ the set $t \leftarrow t + 1$
   and continue with step 2
6. return $w^{[t+1]}$

https://emiliendupont.g

Theorem 5.1: Convergence (Block-Novikoff)

If: 1. Euclidean norm of examples in D is bounded by R

$$\forall i \in [n] : \|\vec{x}_i\| \leq R \text{ for some } R \in \mathbb{R}$$

2. Positive and negative examples are linearly separable with finite margin $\gamma > 0$

$$\exists \vec{u} \in \mathbb{R}^d, \gamma > 0 : \quad \forall i \in [n] : \quad y_i(\vec{u}^\intercal \vec{x}_i) \geq \gamma$$

Then: For number k of updates: $k \leq (R/\gamma)^2$

Proof:

Step 1: Show $\vec{w}_k^\intercal \vec{u} \geq k\gamma$

$$\begin{aligned}
\vec{w}_k^\intercal \vec{u} &= (\vec{w}_{k-1} + y\vec{x})^\intercal \vec{u} = \vec{w}_{k-1}^\intercal \vec{u} + y\vec{x}^\intercal \vec{u} && \text{(perceptron rule)} \\
&\overset{(B)}{\geq} \vec{w}_{k-1}^\intercal \vec{u} + \gamma && \text{(linear separability)} \\
&\geq \vec{w}_{k-2}^\intercal \vec{u} + 2\gamma && \text{(repeat the argument)} \\
&\geq \cdots \geq \ldots && \text{(repeat the argument ☺)} \\
&\geq \vec{w}_0^\intercal \vec{u} + k\gamma = k\gamma && (\vec{w}_0 = 0)
\end{aligned}$$

Step 2: Show $\|\vec{w}_k\|^2 \leq kR^2$

$$\begin{aligned}
\|\vec{w}_k\|^2 &= \vec{w}_k^\intercal \vec{w}_k && \text{(def. of } \|\cdot\|) \\
&= (\vec{w}_{k-1} + y\vec{x})^\intercal (\vec{w}_{k-1} + y\vec{x}) && \text{(perceptron rule)} \\
&= \vec{w}_{k-1}^\intercal \vec{w}_{k-1} + 2y\vec{w}_{k-1}^\intercal \vec{x} + y^2 \vec{x}^\intercal \vec{x} \\
&\leq \|\vec{w}_{k-1}\|^2 + \|\vec{x}\|^2 && ((\vec{x}, y) \text{ prediction mistake}) \\
&\overset{(A)}{\leq} \|\vec{w}_{k-1}\|^2 + R^2 && \text{(boundedness)} \\
&\leq \|\vec{w}_{k-2}\|^2 + 2R^2 \leq \cdots \leq \ldots && \text{(repeat the argument ☺)} \\
&\leq \|\vec{w}_0\|^2 + kR^2 = kR^2 && (\vec{w}_0 = \vec{0})
\end{aligned}$$

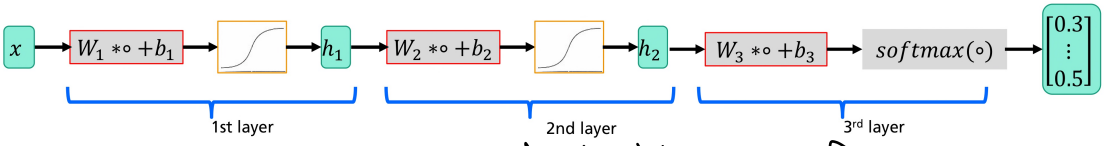
Step 3: Show $k \leq \left(\frac{R}{\gamma}\right)^2$

$$\begin{aligned}
(k\gamma)^2 &\leq (\vec{w}_k^\intercal \vec{u})^2 && \text{(Step ①)} \\
&\leq \|\vec{w}_k\|^2 \|\vec{u}\|^2 = \|\vec{w}_k\|^2 && \text{(Cauchy-Schwarz ineq., } \|\vec{u}\| = 1) \\
&\leq kR^2 && \text{(Step ②)}
\end{aligned}$$

$$\implies k \leq \frac{R^2}{\gamma^2} = \left(\frac{R}{\gamma}\right)^2$$

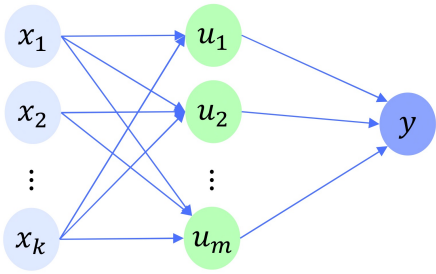
Deep Neural Networks

Neural networks with additional layers:



→ higher capacity and better performance

Backpropagation:



Model: $u_1 = w_1 x;\ h_1 = f_1(u_1);\ u_2 = w_2 h_1;\ \hat{y} = f_2(u_2);\ L = loss(\hat{y}, y)$

A Multi-Layer model can be viewed as a nested function:

$$L(w_1, w_2) = loss(f_2(w_2 f_1(w_1 x)), y)$$

Derivative of the loss w.r.t. the weights:

chain rule: $\frac{\partial f(g(x))}{\partial x}(x_0) = \frac{\partial f(u)}{\partial u}(g(x_0)) * \frac{\partial g(x)}{\partial x}(x_0)$

full derivative of w_1:

$$\frac{\partial loss(f_2(w_2 f_1(w_1 x))), y)}{\partial w_1} = \frac{\partial loss(\hat{y}, y)}{\partial \hat{y}}(\hat{y}) * \frac{\partial f_2(u_2)}{\partial u_2}(u_2) * w_2 * \frac{\partial f_1(u_1)}{\partial u_1}(u_1) * x$$

backpropagate derivative

$$= \frac{\partial loss(\hat{y}, y)}{\partial u_2}(u_2) * w_2 * \frac{\partial f_1(u_1)}{\partial u_1}(u_1) * x$$

full derivative of w_2:

$$\frac{\partial loss(f_2(w_2 f_1(w_1 x))), y)}{\partial w_2} = \frac{\partial loss(\hat{y}, y)}{\partial \hat{y}}(\hat{y}) * \frac{\partial f_2(u_2)}{\partial u_2}(u_2) * u_1$$

→ Calculate partial derivatives from the end

Chain rule for vector functions: $\frac{\partial f(g(x))}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial f(u)}{\partial u_j} * \frac{\partial g_j(x)}{\partial x_i}$

with: $y = f(g(x))$ $\quad y = f(u_1, \ldots, u_m)$ $\quad \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} = g \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}$

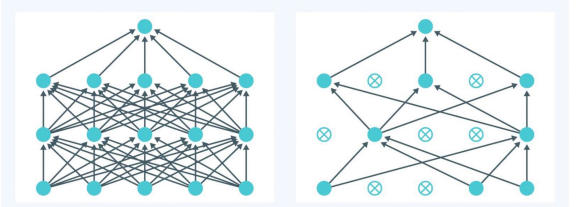
Regularization

1. **Tikhonov Regularization**

L2-Regularization: $\min_w L(w) + \lambda \sum_i w_i^2$

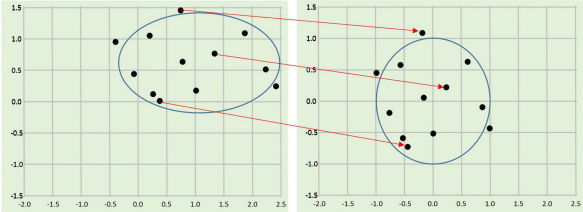
L1-Regularization: $\min_w L(w) + \lambda \sum_i |w_i|$

2. **Dropout Layers**



In each iteration a number of neurons is set to 0 (~50%)

3. **Batch Normalization**



Normalize values in hidden units to mean 0 and variance 1

6. Bayesian Learning

Bayes Formula: $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$

- $P(h|D)$: Posterior probability. Prob. that h is the target hypothesis given data D
- $P(D|h)$: Likelihood: Prob that D is observed given hypothesis h
- $P(h)$: Prior Probability: Belief about the target hypothesis before observing data D
- $P(D)$: Prior Probability. Probability that data D is observed

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

Goal: Find the best hypothesis h given data D, i.e. the most probable hypothesis

Definition: Maximum a Posteriori (MAP)

Maximize the posteriori probability

$$\begin{aligned} h_{\text{MAP}} &= \arg\max_{h \in \mathcal{H}} P(h|D) \\ &= \arg\max_{h \in \mathcal{H}} \frac{P(D|h)P(h)}{P(D)} \\ &= \arg\max_{h \in \mathcal{H}} P(D|h)P(h) \end{aligned}$$

| $P(D)$ constant

Definition: Maximum Likelihood

Assumption: Prior belief about the hypothesis is the same for all hypothesis: $\forall h_i, h_j \in \mathcal{H} : P(h_i) = P(h_j)$

Likelihood of data given h: $L(h) = P(D|h) = \prod_{\langle x,y \rangle \in D} P(\langle x,y \rangle | h) = \prod_{\langle x,y \rangle \in D} P(y|x,h)P(x)$

Maximum Likelihood hypothesis: $h_{\text{ML}} = \arg\max_{h \in \mathcal{H}} \sum_{\langle x,y \rangle \in D} \log P(y|x,h)$

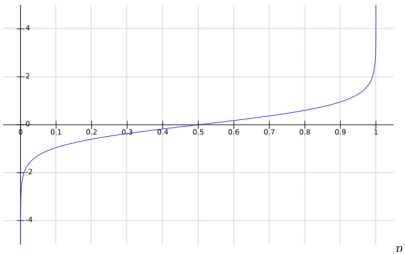
Derivation:

$$\begin{aligned} h_{\text{ML}} &= \arg\max_{h \in \mathcal{H}} L(h) \\ &= \arg\max_{h \in \mathcal{H}} \log L(h) \quad \text{| Log-Trick to replace product with sum} \\ &= \arg\max_{h \in \mathcal{H}} \sum_{\langle x,y \rangle \in D} \log \left(P(y|x,h)P(x) \right) \\ &= \arg\max_{h \in \mathcal{H}} \left(\sum_{\langle x,y \rangle \in D} \log P(y|x,h) + \sum_{\langle x,y \rangle \in D} \log P(x) \right) \\ &= \arg\max_{h \in \mathcal{H}} \sum_{\langle x,y \rangle \in D} \log P(y|x,h) \end{aligned}$$

Logistic Regression

Probabilistic view of classification with the learning of a separating hyperplane modeled as a regression process of interpolating the points.

Logit Function: $\text{logit}(p) = \log \frac{p}{1-p}$



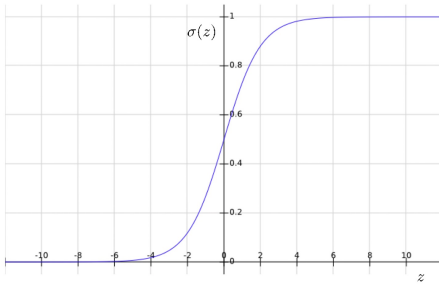
Logarithm of odds that solves the asymmetry of the odds of an event.

Odds of an event e with probability p: $\frac{\Pr(e)}{\Pr(\neg e)} = \frac{p}{1-p}$

Transforms:

probability → logit: $z = \log \frac{p}{1-p}$

logit → probability: $p = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$



Logistic function $\sigma(z)$

Hypothesis for linear regression:

For parameters $\theta = (\theta_0, \theta_1, \ldots, \theta_d) \in \mathbb{R}^{d+1}$

Hypothesis: $h_\theta : \mathbb{R}^d \to \mathbb{R}, \; h_\theta : x \mapsto \theta^\top x'$

Hypothesis for logistic regression:

For parameters $\theta = (\theta_0, \theta_1, \ldots, \theta_d) \in \mathbb{R}^{d+1}$

Hypothesis: $h_\theta : \mathbb{R}^d \to [0,1], \; h_\theta : x \mapsto \sigma(\theta^\top x')$

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

Interpretation:

$$\left.\begin{aligned} \Pr(y=1|x;\theta) &= h_\theta(x) \\ \Pr(y=0|x;\theta) &= 1 - h_\theta(x) \end{aligned}\right\} \text{ Bernoulli process: } \Pr(y|x;\theta) = h_\theta(x)^y (1-h_\theta(x))^{1-y}$$

Equivalence to linear regression:

Logistic Regression:		Linear Regression:	
Predict: $y=1$	if $h_\theta(x) \geq 0.5$	Predict: $y=1$	if $\theta^\top x' \geq 0$
$y=0$	if $h_\theta(x) < 0.5$	$y=0$	if $\theta^\top x' < 0$

with decision boundary: **hyperplane** $\theta^\top x' = 0$

Determining Parameters θ

Given: $D = \{\langle x_1, y_1\rangle \ldots, \langle x_m, y_m\rangle\}$,

Find: θ that minimizes the negative log-likelihood

Minimize: $\boxed{\ell(\theta) = -\sum_{\langle x,y\rangle \in D} y \log h_\theta(x) + (1-y)\log(1-h_\theta(x))}$

→ $L(\theta)$ is convex, thus has a global minimum

Derivation:

1. Maximize log-likelihood

$$\begin{aligned} h_{\text{ML}} &= \arg\max_{\theta\in\mathbb{R}^{d+1}} \sum_{\langle x,y\rangle\in D} \log P(y|x;\theta) \\ &= \arg\max_{\theta\in\mathbb{R}^{d+1}} \sum_{\langle x,y\rangle\in D} \log\left(h_\theta(x)^y(1-h_\theta(x))^{1-y}\right) \\ &= \arg\max_{\theta\in\mathbb{R}^{d+1}} \sum_{\langle x,y\rangle\in D} y\log h_\theta(x) + (1-y)\log(1-h_\theta(x)) \end{aligned}$$

2. Equivalent to minimizing the negative log-likelihood

$$h_{\text{ML}} = \arg\min_{\theta\in\mathbb{R}^{d+1}} -\left(\sum_{\langle x,y\rangle\in D} y\log h_\theta(x) + (1-y)\log(1-h_\theta(x))\right)$$

Partial derivatives:

$$\begin{aligned} \frac{\partial}{\partial\theta_i}\ell(\theta) &= \frac{\partial}{\partial\theta_i}\sum_{\langle x,y\rangle\in D}\left(\log\left(1+\exp\left(\theta^\top x'\right)\right) - y\theta^\top x'\right) \\ &= \sum_{\langle x,y\rangle\in D}(h_\theta(x)-y)x_i \\ &= \sum_{\langle x,y\rangle\in D}\left(\frac{1}{1+\exp(-\theta^\top x')} - y\right)x_i \end{aligned}$$

→ Does not have a closed form solution
→ Minimize iteratively

Loss function:

$$-y\log h_\theta(x) - (1-y)\log(1-h_\theta(x)) \longrightarrow V(h_\theta(x), y) = \begin{cases} -\log h_\theta(x) & \text{if } y = 1 \\ -\log(1-h_\theta(x)) & \text{o/w (i.e., if } y = 0) \end{cases}$$

Bayes Optimal Classifiers

Given: hypothesis space $\mathcal{H} = \{h | h : X \to Y\}$ with $|\mathcal{H}| < \infty$ and $|Y| < \infty$

Find: Most probable classification of new instance x

→ $h_{MAP}(x)$ is not the most probable classification

Definition: Bayes Optimal Classification

$$y_{\text{BayesOpt}} = \underset{y \in Y}{\text{argmax}} \sum_{h \in \mathcal{H}} P(y|h)P(h|x)$$

→ weighted combination of the probabilities predicting y for all hypotheses

Example: suppose for $Y = \{+, -\}$, $\mathcal{H} = \{h_1, h_2, h_3\}$, and instance x we have

$$\begin{array}{lllllllll} P(h_1|x) & = & 0.4 & P(-|h_1) & = & 0 & P(+|h_1) & = & 1 \\ P(h_2|x) & = & 0.3 & P(-|h_2) & = & 1 & P(+|h_2) & = & 0 \\ P(h_3|x) & = & 0.3 & P(-|h_3) & = & 1 & P(+|h_3) & = & 0 \end{array}$$

therefore

- for $y = +$: $\sum_{i=1}^{3} P(+|h_i)P(h_i|x) = 0.4$
- for $y = -$: $\sum_{i=1}^{3} P(-|h_i)P(h_i|x) = 0.6$

and hence

$$y_{\text{BayesOpt}} = \arg\max_{y \in \{+,-\}} \sum_{h \in \mathcal{H}} P(y|h)P(h|x) = -$$

Naive Bayes Classifier

Assumption: Target function $f : X \to Y$ with $X \subseteq A_1 \times \ldots \times A_n$ for sets $A_1, \ldots, A_n$ (attributes)

Naive Bayes assumption:

The attributes are conditionally independent given the target value:

$$P(a_1, a_2, \ldots, a_n | y) = \prod_{i=1}^{n} P(a_i | y)$$

Find: Most probable value $f(x)$ for $x = (a_1, \ldots, a_n)$

Definition: Naive Bayes Classifier

$$y_{\text{NB}} = \underset{y \in Y}{\operatorname{argmax}} P(y) \prod_{i=1}^{n} P(a_i | y)$$

→ Only requires search through all elements of Y and all $a_i \in A_i$

→ No explicit search through the hypothesis space

Derivation:

$$\begin{aligned}
\underset{y \in Y}{\operatorname{argmax}} P(y | a_1, a_2, \ldots, a_n) &= \underset{y \in Y}{\operatorname{argmax}} \frac{P(a_1, a_2, \ldots, a_n | y) P(y)}{P(a_1, a_2 \ldots a_n)} \\
&= \underset{y \in Y}{\operatorname{argmax}} P(a_1, a_2, \ldots, a_n | y) P(y) \\
&= \underset{y \in Y}{\operatorname{argmax}} P(y) \prod_{i}^{n} P(a_i | y)
\end{aligned}$$

Algorithm:

Learning:

algorithm NAIVE_BAYES_LEARN

input: training data $D \subseteq A_1 \times \ldots \times A_n \times Y$

output: estimates of $P(y)$, $P(a|y)$ for all $y \in Y$ and for all $a \in A_i$, for every $i = 1, \ldots, n$

1. **forall** target value $y \in Y$ **do**
2. $\quad \hat{P}(y) \leftarrow$ estimate $P(y)$ from training data D
3. $\quad$ **for** $i = 1, \ldots, n$ **do**
4. $\quad\quad$ **for** all attribute values $a \in A_i$ **do**
5. $\quad\quad\quad \hat{P}(a|y) \leftarrow$ estimate $P(a|y)$ from training data D

Classify new instance: CLASSIFY_NEW_INSTANCE(x)

$$y_{\text{NB}} = \underset{y \in Y}{\operatorname{argmax}} \hat{P}(y) \prod_{a_i \in x} \hat{P}(a_i | y)$$

Problems:

1. Conditional independence is often violated

   Still, often causes no problem, since the estimated posteriors do not need to be correct, only:

$$\operatorname*{argmax}_{y \in Y} \hat{P}(y) \prod_i \hat{P}(a_i|y) = \operatorname*{argmax}_{y \in Y} P(y) P(a_1, \ldots, a_n|y)$$

2. None of the training instances with target value y have attribute value a_i:

$$\Rightarrow \hat{P}(a_i|y) = 0 \implies \hat{P}(y) \prod_i \hat{P}(a_i|y) = 0$$

Solution: m-estimate smoothing

Assume that each attribute is given a prior probability p that have been previously observed in a virtual sample of size m

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

- n_c: number of training examples with target value y
- n: number of examples with target value y and $a = a_i$
- p: prior estimate for $P(a_i|y)$ (i.e. uniform)
- m: weight given to prior (virtual sample size)

Laplace smoothing: $m = |A_i|$ $p = \frac{1}{m}$

Classifying Text

Instances: Text documents

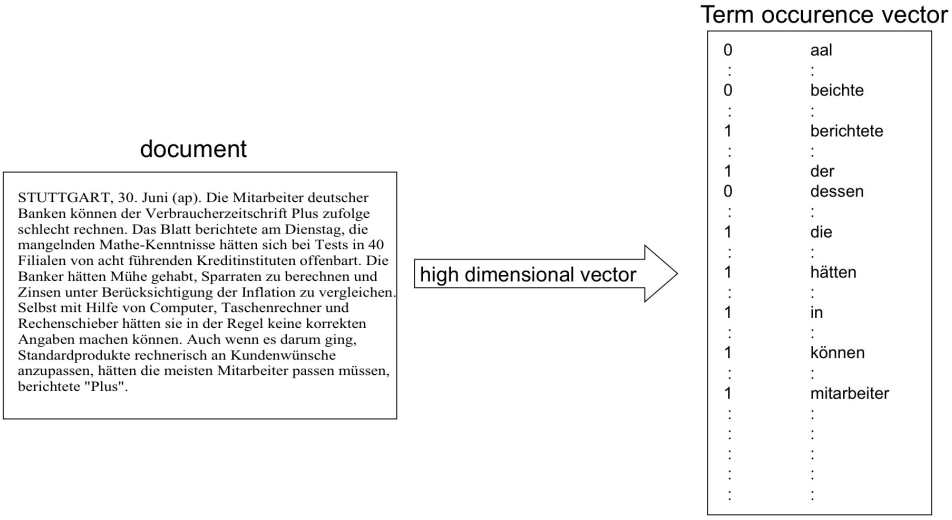
Target values: Document categories

} document $\mapsto \{c_1, \ldots, c_k\}$

Document representation: set-of-words

1. fix a set $V = \{v_1, \ldots, v_n\}$ of vocabulary
   - e.g., which are neither too frequent nor too infrequent
2. represent each document d by a **binary** vector $(f_1, \ldots, f_n)$ with

$$f_i = \begin{cases} 1 & \text{if } v_i \text{ occurs in the document} \\ 0 & \text{o/w} \end{cases}$$



Naive Bayes Classifier

Priors:

$$\hat{P}(c) = \frac{n_c}{n}$$

- n_c: number of documents in class c
- n: total number of documents

Posterior:

$$\hat{P}(v|c) = \frac{n_{c,v} + 1}{\sum_{v' \in V} (n_{c,v'} + 1)} \quad \text{| 1-smoothing}$$

$$= \frac{n_{c,v} + 1}{\sum_{v' \in V} n_{c,v'} + |V|}$$

7. Comparing Learning Algorithms

Comparing hypotheses

Definition: True Error

Error of hypothesis h w.r.t the target function $f(x)$ and distribution D

$$\text{error}_{\mathcal{D}}(h) = \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

Definition: Sample Error

Error of hypothesis h w.r.t the target function $f(x)$ and data sample S

$$error_S(h) = \frac{1}{|S|} \sum_{x \in S} \delta(f(x), h(x)), \text{ with } \delta(f(x), h(x)) = 1 \text{ if } f(x) \neq h(x)$$

→Estimator of the true error
→modeled as a random variable

Definition: Estimator

A random variable Y estimating a parameter p of an underlying distribution

Estimator Bias: $\text{Bias}(Y) = E[Y] - p$

Unbiased Estimator: $\text{Bias}(Y) = 0$

⇒ For n experiments the average of Y converges towards p

Mean Squared Error (MSE): $\text{MSE}(Y) = E[(Y - p)^2]$

Variance: $\text{Var}(Y) = E[(Y - E[Y])^2]$

Properties:

1. Estimators return varying estimates in every run
2. Ideally, estimators should be unbiased
3. Ideally, estimators should have a low as possible mean square error

Theorem 7.1:

For any estimator Y of p, it holds: $\text{MSE}(Y) = \text{Var}(Y) + \text{Bias}(Y)^2$

Proof:

$$\begin{aligned}
\text{MSE}(Y) &= E[(Y - p)^2] \\
&= E[Y^2] + E[p]^2 - 2pE[Y] \\
&= E[(Y - E[Y])^2] + E[Y]^2 + p^2 - 2pE[Y] \\
&= E[(Y - E[Y])^2] + (E[Y] - p)^2 \\
&= \text{Var}(Y) + \text{Bias}(Y)^2
\end{aligned}$$

Corollary 7.2:

For any unbiased estimator Y of p: $\text{MSE}(Y) = \text{Var}(Y)$

→Minimize the variance in the estimation

Estimating the True Error

The true error is typically not obtainable and thus must be estimated

Estimator: Sample error $error_{S_i}(h) = \frac{r_i}{n}$

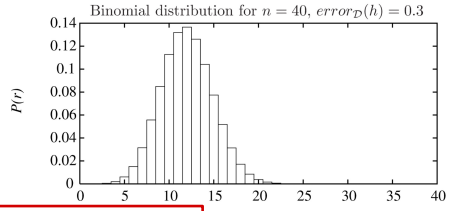
S_i: sample
n: sample size
r_i: number of missclassified instances

Modeling as random variable:

Run the experiment with randomly drawn S_i

Binomial Distribution: $P(r) = \binom{n}{r} \text{error}_{\mathcal{D}}(h)^r (1 - \text{error}_{\mathcal{D}}(h))^{n-r}$

→ each run is modeled as binomial distribution



Mean value: $E[X] = \sum_{i=0}^{n} iP(i) = np$

Variance: $\text{Var}(X) = E[(X - E[X])^2] = np(1-p)$

Standard deviation: $\sigma_X = \sqrt{E[(X - E[X])^2]} = \sqrt{np(1-p)}$

Theorem 7.3:

The sample error is an unbiased estimator of the true error

Proof:

$$\begin{aligned} E[error_S(h)] - error_{\mathcal{D}}(h) &= \frac{E[r]}{n} - error_{\mathcal{D}}(h) \\ &= \frac{n \cdot error_{\mathcal{D}}(h)}{n} - error_{\mathcal{D}}(h) \\ &= 0 \end{aligned}$$

Evaluating the quality of an estimate of the true error

Idea: Use confidence intervals to give a probability on the range the true error lays within given the estimate.

Theorem 7.4: Central Limit Theorem

Let: $Y_1 \dots Y_n$ random variables identically distributed with mean μ and variance σ^2

Sample mean: $\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$

For $n \to \infty$: Normal distribution, with: $\overline{Y} \to \mu$ and $\sigma_{\overline{Y}} \to \frac{\sigma}{n}$

Theorem 7.5: De Moivre-Laplace Theorem

Special case of the central limit theorem.

Binomial distribution of n independent Bernoulli trials is for a large n approximately a normal distribution, with:

$$\mu = np \quad \text{and} \quad \sigma = \sqrt{np(1-p)}$$

Normal distribution approximation

Given:

$error_S(h)$ follows a *binomial* distribution, with

- mean $\mu_{error_S(h)} = error_{\mathcal{D}}(h)$
- standard deviation $\sigma_{error_S(h)}$

$$\sigma_{error_S(h)} = \sqrt{\frac{error_{\mathcal{D}}(h)(1-error_{\mathcal{D}}(h))}{n}}$$

→

Approximation:

approximate this by a *normal* distribution with

- mean $\mu_{error_S(h)} = error_{\mathcal{D}}(h)$
- standard deviation $\sigma_{error_S(h)}$

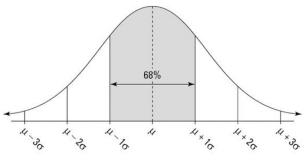
$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

Definition: z-scores

The z-score gives a confidence interval for the distribution of values drawn from a normal distribution.

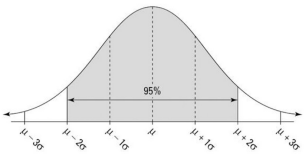
z-scores:

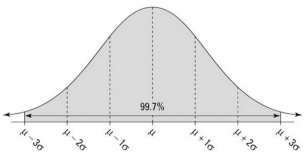
$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58



68-95-99.7 rule:

68.27% of values lie in $[\mu-\sigma, \mu+\sigma]$
95.45% of values lie in $[\mu-2\sigma, \mu+2\sigma]$
99.73% of values lie in $[\mu-3\sigma, \mu+3\sigma]$





z-score Transformation

For large n:

Estimator is distributed with $\mu_{error_S(h)}$ and $\sigma_{error_S(h)}$

Thus: $$\frac{error_S(h) - \mu_{error_S(h)}}{\sigma_{error_S(h)}} = \frac{error_S(h) - error_{\mathcal{D}}(h)}{\sigma_{error_S(h)}}$$

is a standard normal distribution:

$\mathcal{N}(0,1)$ (i.e., $\mu = 0$ and $\sigma = 1$)

Proof:
$$\mathbb{E}\left[\frac{error_S(h) - \mu_{error_S(h)}}{\sigma_{error_S(h)}}\right] = \frac{\mathbb{E}\left[error_S(h)\right] - \mu_{error_S(h)}}{\sigma_{error_S(h)}} = 0$$

and

$$\mathrm{Var}\left[\frac{error_S(h) - \mu_{error_S(h)}}{\sigma_{error_S(h)}}\right] = \frac{1}{\sigma^2_{error_S(h)}} \mathrm{Var}\left[error_S(h)\right] = 1$$

For Standard Normal distribution: $\frac{error_S(h) - error_{\mathcal{D}}(h)}{\sigma_{error_S(h)}} \in [-z_N, z_N]$

→ can be read directly from the z-table

Confidence Interval for the true error:

$$error_{\mathcal{D}}(h) \in [error_S(h) - z_N \sigma_{error_S(h)}, error_S(h) + z_N \sigma_{error_S(h)}]$$

Difference in Error between Hypotheses

Goal: Estimate the difference of the true error of hypotheses h_1 and h_2:

$$d = error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

Suppose. Hypotheses were tested:

hypothesis h_1 on a random sample S_1 with $|S_1| = n_1$ and

hypothesis h_2 on a random sample S_2 with $|S_2| = n_2$,

, with samples S_1, S_2 distributed according to D

Technique:

Estimate: $d = error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$

1. Define estimator: $\hat{d} = error_{S_1}(h_1) - error_{S_2}(h_2)$
   → unbiased
2. Approximate with normal distribution:

$$\mu = d \quad , \quad \sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}$$

3. Find interval such that N% of probability mass falls in the interval:

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}}$$

Comparing Learning Algorithms

Goal: Estimate $E_{S \subset \mathcal{D}}[\text{error}_{\mathcal{D}}(L_A(S)) - \text{error}_{\mathcal{D}}(L_B(S))]$

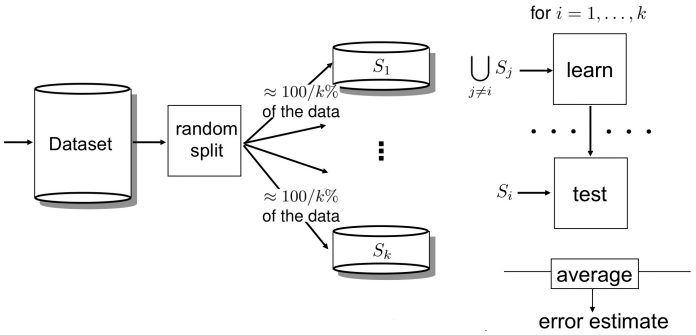
$L(S)$: Hypothesis output by learner L given training set S

Estimator: Split S in training and test set and estimate error on the test set

k-fold Cross Validation:

Split S in k partitions (typically 5, 10) and leave one partition out each run and test on it.

→ leaving data out might result in worse performance



Algorithm: **Algorithm** k-FOLD CROSS VALIDATION

1. partition examples S into k disjoint sets $S_1, \ldots, S_k$ of (almost) equal size
2. **for** $i = 1, \ldots, k$
3. $\text{Train}_i := S \setminus S_i$
4. $\text{Test}_i := S_i$
5. $h_i := \text{LEARN}(\text{Train}_i)$
6. $\text{error}_i := \text{error}_{\text{Test}_i}(h_i)$
7. **return** $\text{error} = \frac{1}{k} \sum_{i=1}^{k} \text{error}_i$

Paired t-Test

Estimate: $\bar{\delta}$: $d = E_{S' \subset S}[\text{error}_{\mathcal{D}}(L_A(S')) - \text{error}_{\mathcal{D}}(L_B(S'))]$, for S' with size $\frac{k-1}{k}|S|$

Confidence Interval: $\bar{\delta} \pm t_{N,k-1} s_{\bar{\delta}}$, with $s_{\bar{\delta}} = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2}$

$t_{N,k-1}$: t-distribution with confidence level N and with $k-1$ degrees of freedom

Algorithm:

1. partition data S into k disjoint test sets $S_1, \ldots, S_k$ of equal size, where this size is at least 30
2. **for** $i = 1, \ldots, k$ **do**

   use S_i for the test set, and the remaining data $S \setminus S_i$ for training set, i.e.,

   - $h_A \leftarrow L_A(S \setminus S_i)$
   - $h_B \leftarrow L_B(S \setminus S_i)$
   - $\delta_i \leftarrow \text{error}_{S_i}(h_A) - \text{error}_{S_i}(h_B)$
3. **return** $\bar{\delta} = \frac{1}{k} \sum_{i=1}^{k} \delta_i$

k ↓

cum. prob	$t_{.50}$	$t_{.75}$	$t_{.80}$	$t_{.85}$	$t_{.90}$	$t_{.95}$	$t_{.975}$	$t_{.99}$	$t_{.995}$	$t_{.999}$	$t_{.9995}$
one-tail	0.50	0.25	0.20	0.15	0.10	0.05	0.025	0.01	0.005	0.001	0.0005
two-tails	1.00	0.50	0.40	0.30	0.20	0.10	0.05	0.02	0.01	0.002	0.001
df											
1	0.000	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	318.31	636.62
2	0.000	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	22.327	31.599
3	0.000	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	10.215	12.924
4	0.000	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	0.000	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	5.893	6.869
6	0.000	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	0.000	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.785	5.408
8	0.000	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	0.000	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	0.000	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	0.000	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	0.000	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	0.000	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	0.000	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	0.000	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	0.000	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	0.000	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	0.000	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.610	3.922
19	0.000	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	0.000	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	0.000	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	0.000	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	0.000	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.485	3.768
24	0.000	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	0.000	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	0.000	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	0.000	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	0.000	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	0.000	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	0.000	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.385	3.646
40	0.000	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	3.307	3.551
60	0.000	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	3.232	3.460
80	0.000	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	3.195	3.416
100	0.000	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	3.174	3.390
1000	0.000	0.675	0.842	1.037	1.282	1.646	1.962	2.330	2.581	3.098	3.300
z	0.000	0.674	0.842	1.036	1.282	1.645	1.960	2.326	2.576	3.090	3.291
	0%	50%	60%	70%	80%	90%	95%	98%	99%	99.8%	99.9%
	Confidence Level										

Incorrectness

Estimation of $E_{S' \subset S}[\text{error}_{\mathcal{D}}(L_A(S')) - \text{error}_{\mathcal{D}}(L_B(S'))]$

not $E_{S \subset \mathcal{D}}[\text{error}_{\mathcal{D}}(L_A(S)) - \text{error}_{\mathcal{D}}(L_B(S))]$

→ Approximation sufficient for comparison

Concept Learning

Confusion Matrix:

Actual Class \ Predicted Class	+	-
+	**TP** true positives	**FN** false negatives
-	**FP** false positives	**TN** true negatives

Pos = TP + FN

Neg = FP + TN

Accuracy:

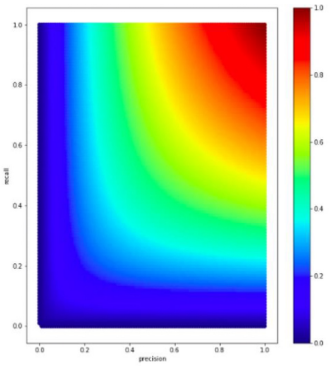
$$\text{accuracy} = \frac{\text{\# correctly classified}}{\text{\# test examples}} = \frac{TP + TN}{TP + FN + FP + TN}$$

Precision:

$$\textbf{precision} = \frac{TP}{TP + FP}$$

Recall:

$$\textbf{recall} = \frac{TP}{TP + FN}$$



F-Score:

$$F_1 = \frac{1}{\frac{1}{2} \cdot \left(\frac{1}{\text{precision}} + \frac{1}{\text{recall}} \right)} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

→ harmonic mean of precision and recall

F-Beta Score:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

β: Weighting between precision and recall

$0 < \beta < 1$: more sensitivity to precision

$\beta = 1$: Equal sensitivity

$\beta > 1$: more sensitivity to recall

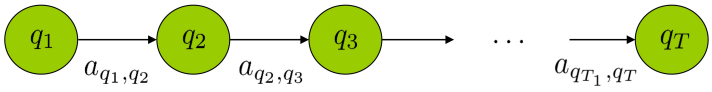
8. Hidden Markov Models

Definition: Markov Model

System with discrete states and state transitions that happen by a given probability

States: $S = \{S_1, S_2, \ldots, S_N\}$

State at time t: $q_t \in S$



State Change: $\Pr(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \ldots, q_1 = S_l)$

Transition Probability Matrix: $N \times N$ matrix A with elements:

$$a_{ij} = \Pr(q_{t+1} = S_j | q_t = S_i) \geq 0 \text{ , s.t. } \sum_{j=1}^{N} a_{ij} = 1$$

Initial Probabilities π_i: Probability that S_i is the first state

$\mathbf{\Pi} = (\pi_1, \ldots, \pi_N)^\top$ with

$$\pi_i = \Pr(q_1 = S_i)$$

Assumptions:

1. **First-order Markov Model (Markov Assumption)**

   q_{t+1} only depends on the last state

   $$\Pr(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \ldots, q_1 = S_l) = \Pr(q_{t+1} = S_j | q_t = S_i)$$

2. **Transition probabilities are independent of time**

   For all t and t' and S_i, S_j:

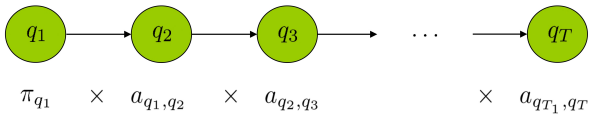
   $$\Pr(q_{t+1} = S_j | q_t = S_i) = \Pr(q_{t'+1} = S_j | q_{t'} = S_i)$$

Definition: Observable Markov Model

States are observable and q_t is known at any time t.

Probability of observed sequence

Let: sequence $\bar{Q} = \langle q_1, q_2, \ldots, q_T \rangle \in S^T$



$$\Pr(\bar{Q} | \mathbf{A}, \mathbf{\Pi}) = \pi_{q_1} \prod_{t=2}^{T} \Pr(q_t | q_{t-1}) = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1},q_t}$$

Estimate transition / initial probabilities

Given: observations $\bar{O}_1, \ldots, \bar{O}_K \in S^T$ with $\bar{O}_k = \langle q_1^k, \ldots, q_T^k \rangle$

Solution: Estimate π_l and a_{ij}

$$\hat{\pi}_l = \frac{\text{\#sequences starting with } S_l}{\text{\#sequences}} = \frac{|\{k \in [K] : q_1^k = S_l\}|}{K}$$

$$\hat{a}_{ij} = \frac{\text{\#transitions from } S_i \text{ to } S_j}{\text{\#transitions from } S_i} = \frac{|\{(k,t) \in [K] \times [T-1] : q_t^k = S_i \wedge q_{t+1}^k = S_j\}|}{|\{(k,t) \in [K] \times [T-1] : q_t^k = S_i\}|}$$

Definition: Hidden Markov Model

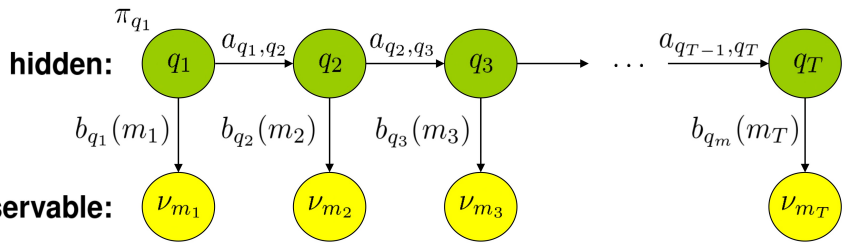
States are not observable

Observable: In each state we can observe an element from the alphabet: $\Sigma = \{\nu_1, \ldots, \nu_M\}$

Observation Probability:

$N \times M$ Matrix $\mathbf{B}$, with: $\mathbf{B}[j,m] = b_j(m) = \Pr(O_t = \nu_m | q_t = S_j) \geq 0$,

with $\sum_{m=1}^{M} b_j(m) = 1$



Formal Definition:

- $S = \{S_1, \ldots, S_N\}$: set of states
- $\Sigma = \{\nu_1, \ldots, \nu_M\}$: set of emission symbols
- $\mathbf{\Pi}$: **initial probability distribution** on S
  - π_i: probability that the initial state is S_i
- $\mathbf{A}$: **transition probability matrix** of size $N \times N$
  - a_{ij}: probability of moving from state S_i to state S_j
- $\mathbf{B}$: **emission probability matrix** of size $N \times M$
  - $b_j(m)$: probability of observing ν_m in state S_j

Parameters: $\lambda = (\mathbf{\Pi}, \mathbf{A}, \mathbf{B})$

Basic Problems of HMMs

1. **Given:** model $\lambda = (\mathbf{\Pi}, \mathbf{A}, \mathbf{B})$ and $\bar{O} = \langle O_1, O_2, \ldots, O_T \rangle \in \Sigma^T$

Find: Probability of observables $\Pr(\bar{O}|\lambda)$

Naive Approach: Marginalize over all possible sequences

For: $\lambda = (\mathbf{\Pi}, \mathbf{A}, \mathbf{B})$, $\bar{O}$, **and** $\bar{Q} = \langle q_1, \ldots, q_T \rangle \in S^T$

$$\Pr(\bar{O}|\lambda) = \sum_{\bar{Q} \in S^T} \Pr(\bar{O}, \bar{Q}|\lambda)$$

, with

$$\Pr(\bar{O}, \bar{Q}|\lambda) = \Pr(q_1) \prod_{t=2}^{T} \Pr(q_t|q_{t-1}) \prod_{t=1}^{T} \Pr(O_t|q_t, \lambda)$$

$$= \pi_{q_1} \prod_{t=2}^{T} a_{q_t q_{t-1}} \prod_{t=1}^{T} b_{q_t}(O_t)$$

→computational intractable

Forward Procedure

Forward variable $\alpha_t(i)$:

$$\alpha_t(i) = \Pr(O_1, \ldots, O_t, q_t = S_i|\lambda)$$

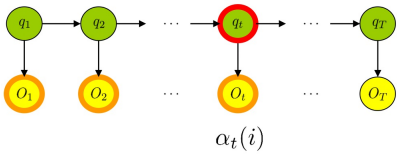
Probability of observing $\langle O_1, \ldots, O_t \rangle$ until time t and being in state S_i at time t, given λ

→Calculation may cause underflow, Normalize: $\alpha_t(i)$ by $\sum_{j=1}^{N} \alpha_t(j)$

Dynamic Programming Algorithm:

Iteratively calculate $\alpha_j(i)$ $j \in [1,t]$ for each timestep to build up the probability of observing $\bar{O}$ in t. by including all forward variables from the last timestamp

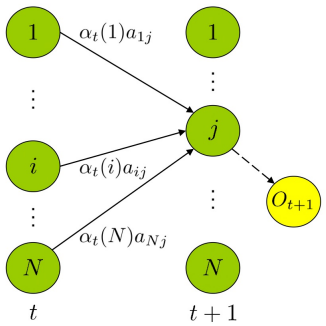
$$\begin{aligned}
\alpha_1(i) &= \Pr(O_1, q_1 = S_i|\lambda) \\
&= \pi_i b_i(O_1) \\
\alpha_{t+1}(j) &= \Pr(O_1, \ldots, O_{t+1}, q_{t+1} = S_j|\lambda) \\
&= \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right) b_j(O_{t+1})
\end{aligned}$$



$\alpha_t(i)$

Proof:

$$\begin{aligned}
\alpha_{t+1}(j) &= \Pr(O_1, \ldots, O_{t+1}, q_{t+1} = S_j|\lambda) \\
&= \Pr(O_1, \ldots, O_{t+1}|q_{t+1} = S_j, \lambda) \Pr(q_{t+1} = S_j|\lambda) \\
&= \Pr(O_1, \ldots, O_t|q_{t+1} = S_j, \lambda) \Pr(O_{t+1}|q_{t+1} = S_j, \lambda) \Pr(q_{t+1} = S_j|\lambda) \\
&= \Pr(O_1, \ldots, O_t, q_{t+1} = S_j|\lambda) \Pr(O_{t+1}|q_{t+1} = S_j, \lambda) \\
&= b_j(O_{t+1}) \sum_{i=1}^{N} \Pr(O_1, \ldots, O_t, q_t = S_i, q_{t+1} = S_j|\lambda) \\
&= b_j(O_{t+1}) \sum_{i=1}^{N} \Pr(O_1, \ldots, O_t, q_{t+1} = S_j|q_t = S_i, \lambda) \Pr(q_t = S_i|\lambda)) \\
&= b_j(O_{t+1}) \sum_{i=1}^{N} \Pr(O_1, \ldots, O_t, q_{t+1} = S_j|q_t = S_i, \lambda) \Pr(q_t = S_i|\lambda)) \\
&= b_j(O_{t+1}) \sum_{i=1}^{N} \Pr(O_1, \ldots, O_t|q_t = S_i, \lambda) \Pr(q_{t+1} = S_j|q_t = S_i, \lambda) \Pr(q_t = S_i|\lambda)) \\
&= b_j(O_{t+1}) \sum_{i=1}^{N} \Pr(O_1, \ldots, O_t, q_t = S_i|\lambda) a_{ij} \\
&= b_j(O_{t+1}) \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right)
\end{aligned}$$



Complexity: $O(TN^2)$

2. **Given:** model $\lambda = (\mathbf{\Pi}, \mathbf{A}, \mathbf{B})$ and $\bar{O} = \langle O_1, O_2, \ldots, O_T \rangle \in \Sigma^T$

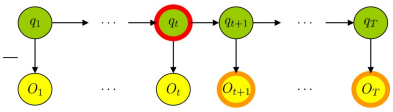
Find: state sequence $\bar{Q} = \langle q_1, q_2, \ldots, q_T \rangle \in S^T$ which has the highest probability: $\bar{Q}^* = \underset{\bar{Q} \in S^T}{\operatorname{argmax}} \Pr(\bar{Q}|\bar{O}, \lambda)$

→ Brute-force exponential in T

Viterbi Algorithm

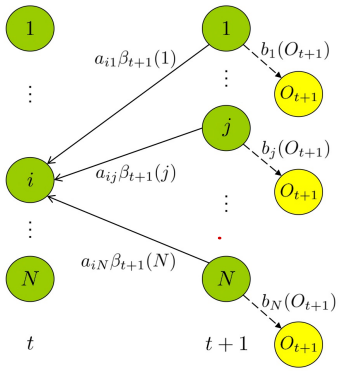
Backward variable $\beta_t(i)$: $\beta_t(i) = \Pr(O_{t+1}, \ldots, O_T | q_t = S_i, \lambda)$

Probability of the partially observed sequence after time t, given state S_i at time t



$$\begin{aligned} \beta_T(i) &= 1 \\ \beta_t(i) &= \Pr(O_{t+1}, \ldots, O_T | q_t = S_i, \lambda) \\ &= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \end{aligned}$$

for all $t = T-1, T-2, \ldots, 1$



Proof:

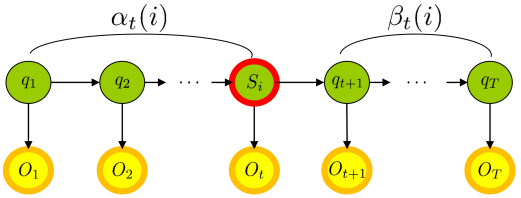
$$\begin{aligned} \beta_t(i) &= \Pr(O_{t+1}, \ldots, O_T | q_t = S_i, \lambda) \\ &= \sum_{j=1}^{N} P(O_{t+1}, \ldots, O_T, q_{t+1} = S_j | q_t = S_i, \lambda) \\ &= \sum_{j=1}^{N} P(O_{t+1}, \ldots, O_T | q_{t+1} = S_j, q_t = S_i, \lambda) \Pr(q_{t+1} = S_j | q_t = S_i, \lambda) \\ &= \sum_{j=1}^{N} \Pr(O_{t+1} | q_{t+1} = S_j, q_t = S_i, \lambda) \Pr(O_{t+2}, \ldots, O_T | q_{t+1} = S_j, q_t = S_i, \lambda) a_{ij} \\ &= \sum_{j=1}^{N} \Pr(O_{t+1} | q_{t+1} = S_j, \lambda) \Pr(O_{t+2}, \ldots, O_T | q_{t+1} = S_j, \lambda) a_{ij} \\ &= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \end{aligned}$$

Variable $\gamma_t(i)$: $\gamma_t(i) = \Pr(q_t = S_i | \bar{O}, \lambda)$, with: $\gamma_t(i) = \dfrac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$

Probability of being in state S_i at time t, given $\bar{O}$ and λ

Proof:

$$\begin{aligned} \gamma_t(i) &= \Pr(q_t = S_i | \bar{O}, \lambda) \\ &= \frac{\Pr(\bar{O} | q_t = S_i, \lambda) \Pr(q_t = S_i | \lambda)}{\Pr(\bar{O} | \lambda)} \\ &= \frac{\Pr(O_1, \ldots, O_t | q_t = S_i, \lambda) \Pr(O_{t+1}, \ldots, O_T | q_t = S_i, \lambda) \Pr(q_t = S_i | \lambda)}{\sum_{j=1}^{N} \Pr(\bar{O}, q_t = S_j | \lambda)} \\ &= \frac{\Pr(O_1, \ldots, O_t, q_t = S_i | \lambda) \Pr(O_{t+1}, \ldots, O_T | q_t = S_i, \lambda)}{\sum_{j=1}^{N} \Pr(\bar{O} | q_t = S_j, \lambda) \Pr(q_t = S_j | \lambda)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \end{aligned}$$



Observation:

$\alpha_t(i)$: Explains the prefix of the sequence up to time t and being in state S_i

$\beta_t(i)$: Explains the suffix from time t+1, given state S_i at time t

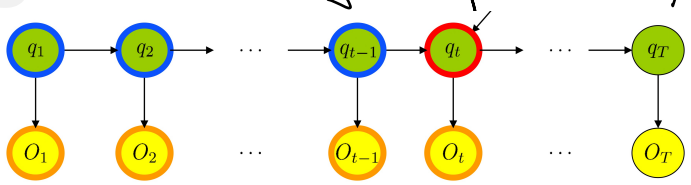
Idea: For each t, choose state q_t^* maximizing the probability: $q_t^* = \underset{i}{\operatorname{argmax}}\, \gamma_t(i)$

Problem: May select $q_{t-1}=S_i$ and $q_t=S_j$, with $a_{ij}=0$

Variable $\delta_t(i)$: $\delta_t(i) = \max_{\langle q_1,\ldots,q_{t-1}\rangle} \Pr(q_1, q_2, \ldots, q_{t-1}, q_t = S_i, O_1, \ldots, O_t|\lambda)$

Probability of a "best" path until time t, that

1. Accounts for the first t observations
2. Ends in S_i
3. Has the highest probability



Algorithm:

MAIN:

input: λ and $\bar{O} \in \Sigma^T$

output: $\bar{Q}^* = \arg\max_{\bar{Q}\in S^T} \Pr(\bar{Q}|\bar{O}, \lambda)$

1. $(p^*, q_T^*) \leftarrow$ FORWARD$(\lambda, \bar{O})$ ← p^*: Prob. of an opt. path, q_T^*: Last state
2. $\bar{Q}^* \leftarrow$ BACKTRACKING(q_T^*)
3. **return** $\bar{Q}^*$

FORWARD$(\lambda, \langle O_1, \ldots, O_T\rangle)$

1. **for** $i = 1, \ldots, N$ **do**
2. $\delta_1(i) = \pi_i b_i(O_1)$ } initialize states for $t=1$
3. $\psi_1(i) = \perp$
4. **for** $t = 1, \ldots, T-1$ **do** ← iterate through timesteps
5. **for** $j = 1, \ldots, N$ **do** ← for each timestep iterate over all states
6. $\delta_{t+1}(j) = \max_i \delta_t(i) a_{ij} b_j(O_{t+1})$ ← Take the highest probability for q_{t+1} when in S_i and observation $O_{t+1:T}$
7. $\psi_{t+1}(j) = \underset{i}{\operatorname{argmax}}\, \delta_t(i) a_{ij}$ ← Most probable state S_i for $t+1$ when in S_j
8. **return** $\left(\max_i \delta_T(i), \underset{i}{\operatorname{argmax}}\, \delta_T(i)\right)$ ← return probability of most probable path and last state

BACKTRACKING(λ, q_T^*)

1. **for** $t = T-1$ **to** 1 **do**
2. $q_t^* = \psi_{t+1}(q_{t+1}^*)$ ← Trace back by taking most probable state
3. **return** $\langle q_1^*, q_2^*, \ldots, q_T^*\rangle$

3. Given: Set of observation sequences $\mathcal{O} = \{\bar{O}^{(1)}, \ldots, \bar{O}^{(K)}\}$

Find: Model λ that maximizes the probability of generating $\mathcal{O}$: $\lambda^* = \underset{\lambda=(\mathbf{\Pi},\mathbf{A},\mathbf{B})}{\operatorname{argmax}} \Pr(\mathcal{O}|\lambda)$