CH 7 PARAMETER ESTIMATION

ENVIRONMENT:
- probability distribution $p(x, y)$
- unknown parameters
- data coming from training set D are i.i.d.
- D can be divided is subset D_i → classes
- for new example x → compute posterior probability

$$p(y_i | x, D) = \frac{p(x | y_i, D) p(y_i | D)}{p(x | D)}$$

↳ TRYING to find parameters that describes p from D

$p(X | D_i, y_i)$

MAXIMUM LIKELIHOOD

θ_i fixed but unknown

Analysis a posteriori

$\theta^* = \arg\max_\theta p(D | \theta)$

$= \arg\max_\theta \prod p(x_j | \theta)$

↳ Maximize the probability of the observed examples D_i of being part of D_i.

↓

LOG → $\nabla_\theta \sum_{j=1}^{n} \log(p(x_j | \theta)) = 0$

↳ Global / Local max

BAYESIAN

θ_i random, known prior distr.

↳ update values while getting new classified data

$$p(x | y_i, D_i) = \int_{\theta_i} p(x_i, \theta_i | y_i, D_i) \, d\theta_i$$

↳ $p(x | D) = \int_\theta p(x | \theta) p(\theta | D) \, d\theta$

for class y_i

⇓

Prediction of the new samples.
(integral over every possible par.)

- $p(x | \theta)$: easily computed, we have the type of distribution and its parameters (e.g. Gaussian)

$$- p(\theta | D) = \frac{p(D | \theta) \cdot p(\theta)}{p(D)}$$

$p(D)$ constant indip. from θ

If we want the most likely value for θ we compute $p(D | \theta) p(\theta)$.

If we want the final probability we have to consider also $p(D)$

$$p(D) = \int_\theta p(D | \theta) p(\theta) \, d\theta$$

Marginalization

la probabilità di D per tutti i valori di θ, pesate dalla probabilità a priori di θ

SUFFICIENT STATISTICS

Any function on a set of samples D is a SUFFICIENT STATISTIC: $S = \Phi(D)$
If $P(D|S,\theta) = P(D|S)$ → contiene tutte le info su D x ottenere la stessa prob.
If θ is a random variable, a sufficient stat. S contains all the relevant information about D:

$$P(\theta|D,S) = \frac{P(D|\theta,S)\,P(\theta|S)}{P(D|S)} = P(\theta|S)$$

For a gaussian S = mean, covariance

CONJUGATE PRIORS → ESEMPIO BERNOULLI and MULTINOMIAL (77-81)

Given likelihood $p(x|\theta)$ and prior $p(\theta)$, then $p(\theta)$ is a conjugate prior for $p(x|\theta)$ if the posterior distribution $p(\theta|x)$ is in the same family as the prior $p(\theta)$

CH.8 BAYESIAN NETWORKS

Bayesian Network → Directed Graphical model
- connections have direction
- node = variable
- edge = directed dependency

INDEPENDENCE ASSUMPTION: $I_{\ell}(G) = \{ \forall i \; x_i \perp NonDescendants_{x_i} \mid Parents_{x_i} \}$ (LOCAL)

Representation of probabilistic relationships: $I_{\ell}(G) \subseteq I(P)$ (set of ind. assertions holding in P)

THE reverse is not necessarily true

To model a joint probability with m variables, we have to consider 2^m possible configurations.
↳ Try to break down into pieces → P FACTORIZES according To G if:

$$P(x_1, \dots, x_m) = \prod_{i=1}^{m} P(x_i \mid Pa_{x_i})$$

(Pa_{x_i}: parents of x_i)

(1) If G is an I-map for P, then P factorizes according to G
(2) If P factorizes according to G, then G is an I-map for P.

PROOF (1) Imap ⇒ Factorization
1. If P is an Imap satisfies $\{ \forall i \; X_i \perp NonDescendants_{X_i} \mid Pa_{X_i} \}$
2. order variables in topological order: $X_i \to X_j \Rightarrow i < j$
3. Decompose joint prob. using CHAIN RULE: $P(x_1, \dots x_m) = \prod P(x_i \mid x_1, \dots x_{i-1})$
4. Local independences imply that for each x_i: $P(x_i \mid x_1, \dots x_{i-1}) = P(x_i \mid Pa_{x_i})$

PROOF (2) Factorization ⇒ Imap
1. If P factorizes according to G, the joint probability can be written as $\prod P(x_i \mid Pa_{x_i})$
2. Considering x_m, By product and sum rule: $P(x_m \mid x_1, \dots x_{m-1}) = \frac{P(x_1, \dots x_m)}{\sum_{x_m} P(x_1, \dots, x_m)}$
3. Applying factorization: $\frac{\prod P(x_i \mid Pa_{x_i})}{\sum_{x_m} \prod P(x_i \mid Pa_{x_i})} = P(x_m \mid Pa_{x_m})$

⇓

FACTORIZATION ⟺ IMAP

BAYESIAN NETWORK is a pair (G, P) where P factorizes over G and it is represented as a set of conditional probability distribution (CPD) associated with the nodes of G

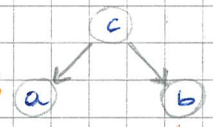
$$P(x_1, \dots, x_m) = \prod_{i=1}^{m} P(x_i \mid Pa_{x_i})$$

CONDITIONAL INDEPENDENCE: a, b are independent $(a \perp b \mid \emptyset)$ if: $P(a,b) = P(a)P(b)$
a, b are conditionally independent given c $(a \perp b \mid c)$ if: $P(a,b \mid c) = P(a \mid c) P(b \mid c)$

Independence assumptions can be verified by repeated applications of sum and product rules. Graphical models allow to directly verify them through the d-separation criterion.

D-SEPARATION

TAIL-TO-TAIL



c is tail-to-tail to the path a -> b

$P(a,b,c) = P(a|c)\,P(b|c)\,P(c)$

a, b are not independent: $P(a,b) = \sum_c P(a|c)\,P(b|c)\,P(c) \neq P(a)\,P(b)$

a, b are conditional independent given c: $P(a,b|c) = P(a|c)\,P(b|c)$

HEAD-TO-TAIL

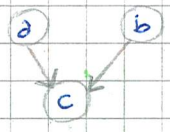
c is in the middle of a chain

$P(a,b,c) = P(b|c)\,P(c|a)\,P(a)$

a, b are not independent: $P(a,b) = \sum_c P(b|c)P(c|a) \neq P(a)P(b)$

a, b are cond. ind. given c: $P(a,b|c) = \frac{P(b|c)P(c|a)P(a)}{P(c)} = P(b|c)P(a|c)$ (BAYES)

HEAD-TO-HEAD



$P(a,b,c) = P(c|a,b)\,P(a)\,P(b)$

a, b are independent: $P(a,b) = \sum_c P(c|a,b)\,P(a)\,P(b) = P(a)\,P(b)$

a, b are NOT condit. ind. given c

REMARK: In real world applications, we tipically observe the effects rather than the causes. We have to apply the Bayes theorem in order to reason about the probability of the causes

GENERAL HEAD-TO-HEAD

+ di 3 variabili

- Let a descendant of a node x be any node which can be reached from x with a path following the direction of the arrows.
- A head-to-head node c unblocks the dependency path between its parents if either itself or any of its descendants receives evidence.

For every descendant, though probably the further the descendant the less influent is the indirect connection.

GENERAL D-SEPARATION

- Given a generic Bayesian network
- Given A, B, C arbitrary nonintersecting nodes

↳ - The sets A and B are d-separated by C if all paths from any node in A to any node in B are blocked.

- A path is blocked if it includes at least s.t. either:
  - the arrows on the path meet tail-to-tail or head-to-tail at the node and it is in C or → Nodo e' flusso di info
  - the arrows on the path meet head-to-head at the node and neither it nor any of its descendants is in C. → Nodo e' flusso di info una volta che si e' condizionato su C

D-SEPARATION IMPLIES CONDITIONAL INDEPENDENCE

↳ The sets A and B are ind. given C ($A \perp B \mid C$) if they're d-separated by C

BN INDEPENDENCE REVISITED

A BN structure G encodes a set of ~~LOCAL~~ independence assumption:

$I_\ell(G) = \{ \forall i \; x_i \perp NonDescendants_{x_i} \mid Parents_{x_i} \}$

↳ every node is cond. ind. by its non-descendants, given its parents

A BN structure G encodes a set of ~~GLOBAL~~ (Markov) independence assumptions:

$I(G) = \{ (A \perp B \mid C) : dsep(A; B \mid C) \}$

↳ "A e B sono d-separati da C"

BN EQUIVALENCE CLASSES

Different BN structures can encode the exact same set of independence assumptions.
Two BN structures G and G' are I-equivalent if $I(G) = I(G')$.
The space of BN structures over X is partitioned into a set of mutually exclusive and exhaustive I-equivalence classes.

I-MAPS VS DISTRIBUTIONS

For a structure G to be an I-map for P, it does not need to encode all its independences. As a consequence a fully connected graph is an I-map for any Pi defined over its variables.

MINIMAL I-MAP: a min. I-map for P is an I-map G which can't be "reduced" into a $G' \subset G$ that is also an I-map for P. In other words it is no more possible to remove edges from G without introducing independences which don't hold in P.

PERFECT MAPS P-MAPS: A structure G is a perfect map (P-map) for P it captures all its independences: $I(G) = I(P)$

PAG. 103: suggestion for building a BN

CH 10 LEARNING BN

We have a dataset of examples $D = \{x(1), \dots, x(N)\}$. Each example $x(i)$ is a configuration for all (complete data) or some (incomplete data) variables in the model (BN).

TASK: estimate parameters of the model (CPD) from data.

SIMPLEST APPROACH: learning parameters maximizing the likelihood of data:

$$\theta^{max} = \arg\max_\theta P(D|\theta) = \arg\max_\theta \mathcal{L}(D,\theta)$$

since data are i.i.d. given θ: $P(D|\theta) = \prod_{i=1}^{N} P(x(i)|\theta)$

Factorization of BN: $P(D|\theta) = \prod_{i=1}^{N}\prod_{j=1}^{m} P(x_j(i)|Pa(i),\theta)$

$P(x_j(i)|Pa_j(i))$ depends only on $\theta_{x_j|Pa_j}$ and not on other parameters.

$$P(D|\theta) = \prod_{i=1}^{N}\prod_{j=1}^{m} P(x_j(i)|Pa_j(i), \theta_{x_j|Pa_j})$$ ML ESTIMATION WITH COMPLETE DATA

Parameters for each CPD can be estimated independently

$$\theta^{max}_{x_j|Pa_j} = \arg\max_{\theta_{x_j|Pa_j}} \prod_{i=1}^{N} P(x_j(i)|Pa_j(i), \theta_{x_j|Pa_j}) = \mathcal{L}(\theta_{x_j|Pa_j}, D)$$

↳ we have to compute a maximization of this kind for every node.

A discrete CPD $P(X|U)$ (U: PARENTS) can be represented by a table, for example:

X_1 \ X_2	R	G	B
T	$\theta_{T|R}$	$\theta_{T|G}$	$\theta_{T|B}$
F	$\theta_{F|R}$	$\theta_{F|G}$	$\theta_{F|B}$

Replacing $P(x(i)|Pa(i))$ with $\theta_{x(i)|u(i)}$, the local likelihood of a single CPD becomes:

$$L(\theta_{X|Pa}, D) = \prod_{u \in Val(U)} \left[\prod_{x \in Val(X)} \theta_{x|u}^{N_{u,x}} \right]$$

(conf. of parents; conf. of X)

Parameters of every column can be estimated independently.

For each multinomial distribution (column), zeroing the gradient of ML:

$$\theta^{max}_{x|u} = \frac{N_{u,x}}{\sum_x N_{u,x}}$$

ML parameters are the fraction of times in which the specific configuration was obs. in data.

ML estimation tends to overfit training set → configuration not appearing in the training set will receive zero probability.

↳ combining ML with prior probability → MAP: Maximum-a-posteriori estimate:

$\theta^{max} = \arg\max_\theta P(D|\theta)P(\theta)$ we want to maximize posterior, not likelih.

The conjugate prior for a multinomial distribution is a Dirichlet distribution with parameters $\alpha_{x|u}$ for each possible value of x. The resulting maximum-a-posteriori estimate is:

$$\theta^{max}_{x|u} = \frac{N_{u,x} + \alpha_{x|u}}{\sum_x (N_{u,x} + \alpha_{x|u})}$$

(conf. observed; → Prior → Parameters of Dirichlet distribution)

Prior is like having observed $\alpha_{x|u}$ imaginary samples with configuration $X=x$, $U=u$

ML ESTIMATION, INCOMPLETE DATA

Some of the examples miss evidence on some of the variables.
↳ use of EXPECTATION-MAXIMIZATION

- Initialize parameters
- Fill-in missing data inferring them using current parameters by solving inference problem to get expected counts.
- Compute parameters maximizing likelihood of expected counts.
  ↳ Iterate procedure to improve the quality of parameters (until converge)

FORMALIZING:

E-STEP $\bar{E}_{P(x|D,\theta)}[N_{ijk}] = \sum_{\ell=1}^{n} P(x_i(\ell) = x_k, Pa_i(\ell) = Pa_j \mid x_\ell, \theta)$ (n: number of training samples)

- x_ℓ: what we know about the ℓ^{TH} example (variabili note)
- N_{ijk}: count where example x_i takes value x_k and parents Pa_i of the example x_i takes value Pa_j.
- IF $x_i(\ell)$ and $Pa_i(\ell)$ are observed for x_ℓ, it's either 0 or 1.

M-STEP $\theta^* = \arg\max_\theta P(D_c|\theta)$, D_c is complete dataset
↳ "completato con i valori attesi"

$$\theta^*_{ijk} = \frac{\bar{E}_{P(x|D,\theta)}[N_{ijk}]}{\sum_{k=1}^{r} \bar{E}_{P(x|D,\theta)}[N_{ijk}]}$$

ML est. can be replaced by maximum a-posteriori (MAP) estimation giving:

$$\theta^*_{ijk} = \frac{d_{ijk} + E_{P(x|D,\theta)}[N_{ijk}]}{\sum_{k=1}^{r} (d_{ijk} + E_{P(x|D,\theta)}[N_{ijk}])}$$ → we have a new estimate of the par. θ.

LEARNING STRUCTURE OF GRAPHICAL MODELS

If we have domain knowledge we could try to construct the graphical model. This is not always possible. Typically, we know some relationships, but not everything.

1. CONSTRAINT-BASED: test conditional independences on the data and construct a model satisfying them. Based on results of tests, we add connection to network.
2. SCORE-BASED: assign a probabilistic score to each possible structure, define a search procedure looking for the structure maximize the score.
3. MODEL-AVERAGING: Treat structure as random variable. We assign a prior probability to each structure, and average prediction over all possible structures weighted by their probabilities.

Quando si costruisce una BN MAI usare mai prob. zero ad una conf. Altr. molti qualsiasi futuro esempio con quella configurazione avrà probabilità zero

CH. 11 NAIVE BAYES

PROBLEM: CLASSIFICATION → we have a set of inputs, compute the output } supervised learning

IDEA: Probabilistic algorithm for classification.
↳ compute argmax given data: $y^* = \text{argmax}_{y \in Y} P(y|\vec{x}) = \text{argmax}_{y \in Y} \frac{P(\vec{x}|y)P(y)}{P(\vec{x})}$

$P(\vec{x})$ is the input (a constant) so we can consider only $P(\vec{x}|y)P(y)$

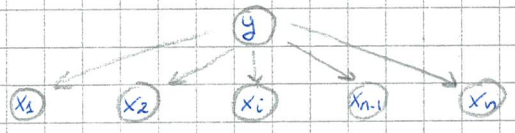
PROBLEM → To compute $P(\vec{x}|y) = P(x_1, \dots x_n | y)$ we need 2^n configurations

SOLUTION → Naive Bayes: assume that the different features $(x_1, \dots, x_n)$ of the input are independent given the class.

$$P(y|\vec{x}) = \prod_{i=1}^{n} P(x_i|y)P(y)$$

↳ consider separate table for each feature → not always accurate

No connection between features



connection from class y to each feature

TEXT CLASSIFICATION

Let's consider having a piece of text x and we consider words as features
↳ $\vec{x} = (w_1, \dots w_n)$. Probability to find a word or another is the same → ind.
To model a naive Bayes classifier is by saying that each word is modelled as independent items belonging to a greater set of words V (vocabulary). Each word can take a number of values corresponding at most with the size of the vocabulary. The Naive classifier will be:

$$\prod_{i=1}^{|\vec{x}|} P(w_i|y)P(y)$$

↳ Multinomial distribution that is specific for every types of topic y, so we can get all possible words for all possible class.
Each column is a multinomial distribution over size of $|V|$ possible values.

CH. 12 LINEAR DISCRIMINANT FUNCTIONS

GENERATIVE LEARNING
model the distribution governing the data. → (e.g. BN)
With a generative model we can generate new data (samples) according to the distribution.

DISCRIMINATIVE LEARNING
model the discriminant function.
The purpose is to predict output given input. In classification, models the boundaries between possible classes.
- PRO: useful with complex data; learning only parameters which are interesting
- CONS: less flexible; not allow to performe inference; not possible to generate new data

A linear discriminative model is a discriminative model which use a linear function to make predictions.
The DISCRIMINANT FUNCTION is a linear combination of example features

$$f(x) = w^T x + w_0$$

w_0 → bias / treshold

In BINARY CLASSIFICATION the predicted class is obtained taking the sign of the linear function: $f(x) = \text{sign}(w^T x + w_0)$.
$f(x) = 0$ is the DECISION BOUNDARY. Weight vector w is orthogonal to the decision hyperplane. → see PROOF

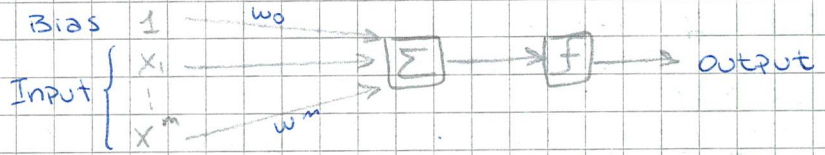
Functional Margin: the value of $f(x)$ for a certain point x is called Functional margin. It can be sees as a CONFIDENCE in the prediction. → The closer $f(x)$ is to zero, the smaller the confidence of the prediction.

Geometric Margin: The distance from x to the hyperplane: $r^x = \frac{f(x)}{\|w\|}$
Normalized version of functional margin.
Given an arbitrary point x, it can be represented by its projection on H (x^p) plus its distance to H: $x = x^p + r^x \frac{w}{\|w\|}$

PERCEPTRON

BIOLOGICAL MOTIVATION: electrochemical reactions → synapses (excite or inhibit) → neurons accumulates signals → exceeds treshold → signal

PERCEPTRON: Matematical implementation of neuran structure → $f(x) = \text{sign}(w^T x + w_0)$
↳ linear combination of input features. coefficients → weights.
Result of summation is processed by an activation function giving output 0 or 1.



We can represent primitive booleaun function (AND, OR, NOT), but not complex formula like XOR. We need network with 2 layers.

$$f(x) = \text{sign}(\hat{w}^T \hat{x}) \qquad \hat{w} = \begin{pmatrix} w_0 \\ w \end{pmatrix} \qquad \hat{x} \begin{pmatrix} 1 \\ x \end{pmatrix} \rightarrow x_0, \text{bias}$$

↳ shorter way

PARAMETER LEARNING

Function of the parameters to be optimized -> measure of the error on the training set D -> LOSS FUNCTION: $\ell(y, f(x))$ loss we have to pay for predicting $f(x)$ instead of y

$$E(w; D) = \sum_{(x,y)\in D} \ell(y, f(x))$$ -> minimize E lead to overfitting

Gradient Descent common approach to minimize error function.
Progressively find a LOCAL minima of the error function.

(1) Initialize w (random)
(2) Iterate until gradient is approximately zero: $w = w - \eta \nabla E(w; D)$
↳ η is the LEARNING RATE: amount of movement at each gradient step.
Too low η implies slow converge. Too high η implies oscillations.
↳ Techniques exists to adaptively modify η.

Perceptron Training Rule The error is the sum of the functional margins ($f(x)$) of incorrectly classified examples:

$$E(w; D) = \sum_{(x,y)\in D_E} -yf(x)s$$

- D_E is the set of current training errors $yf(x) \leq 0$
- s is the scaling factor to adjust the learning rate

↳ We are taking in consideration the confidence of our predictions.
For this reason it's called CONFIDENCE LOSS.

If we apply gradient descent on previous equation $E(w; D)$ the amount of update is:

$-\eta \nabla E(w; D) = \eta \sum_{(x,y)\in D_E} yx$ * you go opposite to the gradient

Following this approach we have to iterate over all the training set at each iteration to compute the gradient of the error for each example
↳ SLOW CONVERGENCE
↳ SOLUTION: STOCHASTIC PERCEPTRON RULE

(1) Initialize weights randomly
(2) Iterate until all examples are correctly classified
(a) for each incorrectly classified training examples (x, y) update the weight vector: $w \leftarrow w + \eta yx$
↳ we make a gradient step for each training error. Each gradient step is very fast. At each iteration we compute the gradient on a different error function.

PERCEPTRON REGRESSION

Linear models are used also for LINEAR REGRESSION.

$Xw = y$ X: training matrix {row -> samples, column -> features}
y: output training matrix

Giving solution: $w = X^{-1}y$
↳ NOT WORK X is not square
Not exact solution tipically exists

We are going to minimize error function -> MEAN SQUARED ERROR (MSE)

$$E(w; D) = \sum_{(x,y)\in D} (y - f(x))^2 = (y - Xw)^T (y - Xw)$$

Closed form solution compute gradient and make it equal to zero $\nabla \tilde{E}(w; D) = 0 \Rightarrow w = (X^T X)^{-1} X^T y$ (pseudo-inverse of X^+)

If there is an exact solution, MSE finds it, otherwise it will provide an approximation which minimizes the squared error.

With a lot of features, inverting $(X^T X)$ can be very expensive.

Is we can compute the gradient and perform gradient descent.

For a single weight, the gradient step is computed:

For a single weight $\frac{\partial E}{\partial w_i} = \sum_{(x,y) \in D} (y - f(x))(-x_i)$

MULTICLASS CLASSIFICATION

Solve multiclass classification by means of combination of binary classification tasks.

- **ONE VS ALL** positive examples $\in$ class / negative examples $\notin$ class } 1 classificatore binario x ogni classe
  - m decision hyperplanes, one per class
  - Each classifier provides a confidence → take class with max. confidence
  - In decision boundaries $f_i(x)$ $f_j(x)$ we don't know which class choose.
- **ONE VS ONE** positive examples $\in$ class / negative examples $\in$ other class } 1 classificatore per ogni coppia di classi
  - $\frac{n(m-1)}{2}$ binary classifier, if we have m classes

GENERATIVE LINEAR CLASSIFIER

The task is to compute the posteriori $p(y|x)$.

In gaussian distributions, linear decision boundaries are when Σ is shared among classes ($\Sigma_i = \Sigma$). → Only under this assumption Gaussian is a linear classifier.

Naive Bayes classifier → Linear if all features are independent one with respect to the other given the class.

$f_i(x) = P(x|y_i)P(y_i)$

↳ In both cases we can use the LOG function.

CH. 13 SUPPORT VECTOR MACHINE

NO NON-LINEAR SVM / SVM REGRESSION (8)

SVM are a popular method for linear (and non-linear) classification.

- linear classifier selecting hyperplane maximizing separation margin between classes (LARGE MARGIN CLASSIFIERS)
- solution only depends on a small subset of training samples (support vectors)
- sound generalization theory (bounds on error based on margin).
- can be easily extended to non linear separation (kernel machines).

Classifier margin Given a training set D, a classifier confidence margin is

$$\rho = \min_{(x,y)\in D} y f(x)$$

This is the minimal confidence of the classifier in a correct prediction (which corresponds to $yf(x)$ as for perceptron) and has to be positive in order to correctly separate the classes.

The GEOMETRIC MARGIN is the same value divided by the norm of w:

$$\frac{\rho}{\|w\|} = \min_{(x,y)\in D} \frac{y f(x)}{\|w\|}$$

Canonical Hyperplane there is an infinite number of equivalent hyperplanes that encodes for the same hyperplane: → Indicate multiple solutions

$$a(w^T x + w_0) = 0 \qquad a \neq 0$$

The CANONICAL HYPERPLANE having the confidence (classifier margin) equal to 1

$$\rho = \min_{(x,y)\in D} y f(x) = 1$$

and its geometric margin is: VEDI FIG. 13.2 PAG. 149

$$\frac{\rho}{\|w\|} = \frac{1}{\|w\|}$$

HARD MARGIN SVM

THEOREM: Margin Error Bound

Consider a set of decision functions $f(x) = \text{sign}(w^T x)$ with $\|w\| \leq \Lambda$ and $\|x\| < R$ for some $R, \Lambda > 0$.

Let $\rho > 0$ and ν denote the fraction of training samples with margin smaller than $\rho / \|w\|$.

For all distributions P generating the data, with probability at least $1-\delta$ (δ prob. of error) over the drawing m training patterns, and for any $\rho > 0$ and $\delta \in (0,1)$, the probability that a test pattern drawn for P will be misclassified is bound from above by:

$$\nu + \sqrt{\frac{c}{m}\left(\frac{R^2\Lambda^2}{\rho^2} \ln^2 m + \ln\left(\frac{1}{\delta}\right)\right)} \qquad c = \text{universal constant}$$

(ν: component of all margin errors)

↳ This theorem shows a bound on the generalization error of a classifier which is trained to be an hard margin SVM maximized.

↳ The probability of test error depends on
- number of margin errors ν (samples with margin $< \rho / \|w\|$)
- number of training sample (error depend on $\sqrt{(\ln^2 m)/m}$)
- size of the margin (error depends on $1/\rho^2$)

Learning Problem The learning objective is $\min_{w, w_0} \frac{1}{2}\|w\|^2$

subject to $y_i(w^T x_i) + w_0 \geq 1$

hard margin is called this way because it require all the examples to have a confidence at least equal to 1. → not always possible

↳ Quadratic optimization problem

TECHNIQUES FOR SOLVE THE PROBLEM

Karush-Kuhn-Tucker approach (KKT) convert problem into an unconstrained problem with the same solution.

Let's have a general constrained problem:

$\min_z f(x)$ subject to $g_i(z) \geq 0 \ \forall i$

Introduce $\alpha_i \geq 0$ (Lagrange multiplier) for each constraint and rewrite

$$\min_z \max_{\alpha \geq 0} f(z) - \sum_i \alpha_i g_i(x)$$ i: all possible constraints

↳ Remove constraints adding it into the objective

The optimal solution(s) x^* for this problem is/are also optimal solution for the original constraint problem.

If there is at least 1 constraint that x' doesn't satisfy this leads to a non-valid solution for the original problem.

If all constraints are satisfied, maximization over the α will set all elements of the summation to zero, so that z' is a solution for $\min_z f(z)$.

Applying KKT to SVM we can formalize: $\min_{w,w_0} \frac{1}{2}\|w\|^2$ subject to $y_i(w^T x_i + w_0) \geq 1$

constraints can be included in minimization:

Lagrange multipliers $\mathcal{L}(w, w_0, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i (w^T x_i + w_0) - 1 \right)$

↳ minimized respect to w, w_0. Maximized respect to α_i

$\nabla_w \mathcal{L} = 0 \Rightarrow w = \sum_{i=1}^{m} \alpha_i y_i x_i$ → write primal variables in terms of dual (α)

$\frac{\partial \mathcal{L}}{\partial w_0} \mathcal{L} = 0 \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0$

↳ substituting in lagrangian: $\mathcal{L}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j$

↳ To be maximized due to the dual variables α

The objective becomes: $\max_\alpha \mathcal{L}(\alpha)$ with constraints → $\alpha_i \geq 0$ $i \in (1, m)$

↘ $\sum_{i=1}^{m} \alpha_i y_i = 0$

⇓

Formulation in dual variables. Dual constraints are simpler.

↳ choice depends on number of variables

DECISION FUNCTION $f(x) = w^T x + w_0 = \sum_{i=1}^{m} \alpha_i y_i x_i^T x + w_0$

Primal | Dual

Why we call this method support vector? Lagrangian is a min-max problem ($\min_w \max_\alpha$), the result is a SADDLE POINT. Each of the conditions should be equal to zero, therefore each element in the summation should be zero. This happens in two ways:

1. α_i are equal to zero
2. $y_i (w^T x_i + w_0) - 1 = 0$ → confidence for example should be 1.

   There are 2 hyperplanes with confidence = 1 → Pos. examples conf = 1

   ↘ neg examples conf = -1

   These are the only points with $\alpha_i > 0$ → support vectors.

Decision hyperplane $f(x) = 0$ is defined only in terms of the support vectors.

Bias w_0 can be computed from KKT conditions: $w_0 = \frac{1 - y_i w^T x_i}{y_i}$ (x_i: support vector)

↳ It's usually averaged over all support vectors

SOFT MARGIN SVM

PROBLEM WITH HARD MARGIN: adding new points, previous hyperplane are not longer valid. -> OVERFITTING

SOLUTION: Soft margin SVM -> maximize margin but with soft constaint: some exeptions of falsely classified samples are allowed.

We add a slack variable ξ_i for each sample in the dataset, then we will sum them according to a multiplicative factor C which is a hyperparameter:

$$\min_{w \in X, w_0 \in \mathbb{R}, \xi \in \mathbb{R}^m} \frac{\|w\|^2}{2} + C \sum_{i=1}^{m} \xi_i$$

subject to: $y_i (w^T x_i + w_0) \geq 1 - \xi_i, \quad \xi_i \geq 0$

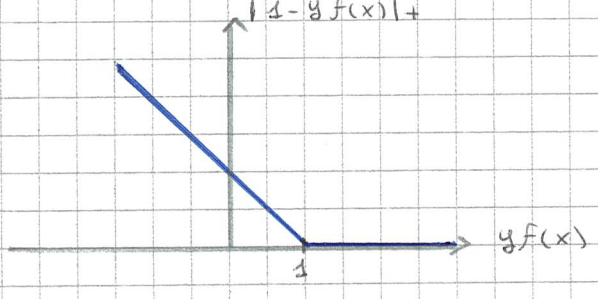
$\xi_i = 0$ -> correct classification
$\xi_i > 0$ -> miss-classification
Larger $\sum_{i=1}^{m} \xi_i$, larger the number of miss-classified examples

Collecting slack variables $C \sum_{i=1}^{m} \xi_i$ trying to combine margin maximization and penalty minimization.
C gives a trade-off between maximizing the margin and fitting examples.

Regularization Theory: Objectives to be learned that combine a complexity term and training errors:

$$\min_{w \in X, w_0 \in \mathbb{R}, \xi \in \mathbb{R}^m} \frac{\|w\|^2}{2} + C \sum_{i=1}^{m} loss(y_i, f(x_i))$$

In soft margin ξ_i is the loss of false predictions: $\xi_i = Loss(y_i, f(x_i))$
ξ_i is non-negative -> $Loss(y_i, f(x_i)) = |1 - y_i (w^T x_j + w_0)|_+$
↳ PLOT:



Hinge loss function
(Not a smooth loss)

Lagrangian for soft margin SVM

KKT formulation: $L = \frac{1}{2}\|w\|^2 + C \sum_{i=0}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i (y_i (w^T x_i + w_0) - 1 + \xi_i) - \sum_{i=0}^{m} \beta_i \xi_i$

Primal variable: w, w_0 Dual variables: α, β

$\nabla_w L \Rightarrow w = \sum_{i=1}^{m} \alpha_i y_i x_i \quad \frac{\partial L}{\partial w_0} \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 \quad \frac{\partial L}{\partial \xi} \Rightarrow C - \alpha_i - \beta_i = 0$

$L(\alpha) = \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum \alpha_i \Rightarrow$ correspond exactly to hard.

Dual Formulation: $\max_{\alpha_i \in \mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j$

subject to $0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{m} \alpha_i y_i = 0$

At the saddle point it holds that for all i: $\alpha_i (y_i (w^T x_i + w_0) - 1 + \xi_i) = 0$
$\beta_i \xi_i = 0$

Support vectors are examples for which $(y_i (w^T x_i + w_0)) \leq 1$

STOCHASTIC GRADIENT DESCENT

Compute gradient on a single ~~xa~~ example: $E(w,(x_i,y_i)) = \frac{\lambda}{2}\|w\|^2 + |1 - y_i \langle w, x_i \rangle|_+$

Whenever we don't have a single gradient in a point, we can perform a subgradient. Its indicator function will be:

$$\mathbb{1}[y_i \langle w, x_i \rangle < 1] = \begin{cases} 1 & \text{if } y_i \langle w, x_i \rangle < 1 \\ 0 & \text{otherwise} \end{cases}$$

The subgradient of a function f in a point x_0 is any vector v such that for any x hold the condition: $f(x) - f(x_0) \geq v^T (x - x_0)$

Large scale SVM learning

PSEUDOCODE

(1) Initialize $w_1 = 0$
(2) For $t = 1$ to T:
   a. Randomly choose (x_{it}, y_{it}) from D
   b. Set $\eta_t = \frac{1}{\lambda t}$
   c. Update w: $w_{t+1} = w_t - \eta_t \nabla_w E(w; (x_{it}; y_{it}))$
(3) Return w_{T+1}

CH. 15 KERNEL MACHINES

PROBLEM: with SVM we can deal non-linear problem mapping in non-linear features as combination of features. It becomes computationally unfeasible in high dimensional cases.
SOLUTION: kernel trick

Kernel Trick replacing the dot products that appear in the dual formulation of non-linear SVMs with an equivalent kernel function:

$$K(x, x') = \Phi(x)^T \Phi(x')$$ Lineare

The kernel function uses example in input space (not feature). We can build a function that can compute this without to explicitly build the map.

The dual optimization problem is: $\max_{\alpha \in \mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i y_i \alpha_j y_j \Phi(x_i)^T \Phi(x_j)$

subject to: $\alpha_i \geq 0$, $\sum_{i=i}^{m} \alpha_i y_i = 0$

Dual decision function: $f(x) = \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i)^T \Phi(x') = \sum_{i=1}^{m} \alpha_i y_i K(x_i, x')$

Feature map $\Phi: X \to \mathcal{H}$ Φ is a function mapping each example to an higher dimensional space :(

Examples x are replaced with their feature mapping $\Phi(x)$.
Maps all possible conjuctions of features { of a certain degree d → HOMOGENEUS ; up to a certain degree → INHOMOGENEUS

What are kernels?

HOMOGENEUS: $K(x, x') = (x^T x')^d$ → result is a scalar.
A kernel it always correspond to a dot product and some feat. space.

INHOMOGENEUS: $K(x, x') = (1 + x^T x')^d$

KERNELS A valid kernel what works properly, is a function defined over the cartesian product of the input space: $K: X \times X \to \mathbb{R}$ that corresponds to a dot product in SOME feature space $K(x, x') = \Phi(x)^T \Phi(x')$, kernels can be seen as similarities between objects.

VALIDITY OF A KERNEL

If we built learning systems on non-vector objects we have to invent the kernel ourselves, therefore we need to verify that is a valid one.

GRAM MATRIX: given examples $\{x_1, \dots, x_m\}$ and a kernel function K, the Gram matrix K is the symmetric matrix of pairwise kernels between examples: $K_{ij} = K(x_i, x_j)$ $\forall i, j$ → simmetric.
Is a positive definite if $\sum_{i,j=1}^{m} c_i c_j K_{ij} \geq 0$ $\forall c \in \mathbb{R}^m$
All eigenvectors are non negative.

To show validity check the satisfaction of at least one of these conditions:
(1) prove its positive definiteness (difficult)
(2) find corresponding feature map: explication of Φ dot-product combination by making the feature map explicit
(3) use kernel combination properties in order to build a new kernel, the operation that we can perform on kernels will preserve their properties.

Kernelizing different SVMs

Example support vector regression:

Dual problem $\max_{\alpha \in \mathbb{R}^m} -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\Phi(x_i)^T\Phi(x_j) - \varepsilon\sum_{i=1}^{m}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} y_i(\alpha_i^* - \alpha_i)$

subject to $\sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0$

Regression function: $f(x) = w^T\Phi(x) + w_0 = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\Phi(x_i)^T\Phi(x) + w_0$

Kernelizing a perceptron

Initialize $\alpha^{(0)} = 0$, the coefficients to be learnt

Iterate through the examples, and for each miss-classified example we perform an update: $\alpha_i^{(t+1)} \leftarrow \alpha_i^{(t)} + \eta y_i$

Classification function becomes $f(x) = \sum_{i=1}^{m}\alpha_i K(x_i, x)$

STOCHASTIC PERCEPTRON: $f(x) = w^T x = \sum_{i=1}^{d} w_i x_i + w_0$, update $w^{(t+1)} \leftarrow w^{(t)} + \eta y_i x_i$

TYPES OF KERNELS

- Linear $K(x, x') = x^T x'$
- Polynomial $K_{c,d}(x, x') = (x^T x' + c)^d$
- Gaussian: $K_\sigma(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$

Inverso della similarità

If is $\approx 0 \Rightarrow K_\sigma(x, x') = 1$

x' is the mean, x = sample

depends on width parameter σ

Gaussian kernel can uniformly approximate any arbitrary continuous function

KERNELS ON STRUCTURED DATA

Kernels are generalization of dot products to arbitrary domains.

It's possible to design kernels over structured data like sequences, trees or graphs. Idea is designing a pairwise function measuring the similarity of two objects. This measure has to satisfy the p.d. conditions to be a valid kernel.

MATCH (DELTA) KERNEL: $K_\delta(x, x') = \delta(x, x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$ simplest kernel on struct. Checks equality.

SPECTRUM KERNEL: feature space is space of all possible k-grams (subsequences)

An efficient procedure based on suffix trees allows to compute kernel without explicitly building feature maps.

Kernel combination: Build a kernel by combining pieces means build a valid kernel by following some rules that ensures validity:

- simpler kernels can be combined using certain operation $(+, \times, \dots)$
- allow to combine simpler kernels to design complex kernel
- correctly using combination operators guarantees that complex kernels are preserved.

SUMMATION: $(K_1 + K_2)(x, x') = [\Phi_1(x)\ \Phi_2(x)]\begin{pmatrix}\Phi_1(x') \\ \Phi_2(x')\end{pmatrix}$

MULTIPLICATION: $(K_1 \times K_2)(x, x') = \Phi_{12}(x)^T\Phi_{12}(x')$

LINEAR COMBINATION: $K_{sum}(x, x') = \sum_{k=1}^{K}\beta_k K_k(x, x')$

The weights of linear combination can be learned simultaneously to the predictor weights (αs) → Allows to select which kernel is more useful

NORMALIZATION: $\hat{K}(x, x') = \frac{K(x, x')}{\sqrt{K(x, x)\,K(x', x')}}$ cosine normalization

COMPOSITION: $(K_\sigma \circ K)(x, x') = \exp\left(-\frac{K(x, x) - 2K(x, x') + K(x', x')}{2\sigma^2}\right)$

↳ project in higher dimensional space

KERNELS ON GRAPHS

WEISTFEILER-LEHMAN GRAPH KERNEL: efficient graph kernel for large graph.

WL ISOMORPHISM TEST -> if it says that 2 graphs are different, they are surely diff.

Given $G=(V,E)$ and $G'=(V',E')$, with $n=|V|=|V'|$. Let $\Sigma(G)=\{I(v) \mid v \in V\}$ be the set of labels in G, and let $\Sigma(G)==\Sigma(G')$. Let label(s) be a function assigning a unique label to a string.

- Set $I_0(v)=I(v)$ for all v.
- For $i \in [1, n-1]$
  - (1) For each node v in G and G'
  - (2) Let $M_i(v)=\{I_{i-1}(u) \mid u \in neigh(v)\}$
  - (3) concatenate the sorted labels of $M_i(v)$ into $S_i(v)$
  - (4) Let $I_i(v)=label(I_{i-1}(v) \circ S_i(v))$
  - (5) IF $\Sigma_i(G) \neq \Sigma_i(G')$
    RETURN FAIL
- RETURN PASS

WEISTFEILER-LEHMAN graph KERNEL: $K_{\omega}^{(h)}(G,G')=\sum_{i=0}^{h} K(G^{(i)}, G'^{(i)})$

VEDI SLIDE 29-30 + disegni

CH. 17 ENSEMBLE METHODS

Combine multiple ML models can produce better predictions than those of any single model. Training ensembles can be parallelized, with substantial computational savings.

ENSEMBLING STRATEGIES: - bagging → diversify the training sets
- stacking → diversify the models being trained
- boosting → diversify the importance of examples

Bagging: bootstrap aggregation

Take a learning algorithm $\mathcal{A}$. Extract m different datasets $\mathbb{D}^{(i)}$ from the original training set $\mathbb{D}$. Train one base model per dataset $f^{(i)} = \mathcal{A}(\mathbb{D}^{(i)})$
Combine predictions of different base models $\hat{y} = \text{COMBINE}(f^{(1)}(x), \ldots, f^{(m)}(x))$.

Simply partitioning $\mathbb{D}$ into m subsets produces very small training sets.
Bootstrap resampling extracts $N = |\mathbb{D}|$ samples from $\mathbb{D}$ with replacement.
Repeating the procedure m times to get diverse datasets of the same size as $\mathbb{D}$.

Each example has probability $(1 - 1/N)$ of not being selected at each draw.
Each example has probability $(1 - 1/N)^N$ of not being selected after N draws.
For large enough N, this is 37% → 37% of the dataset are not part of a given training set. → Out-of-Bag instances (OOB) → used for test set

COMBINATION STRATEGIES

- Majority voting: predict the class with most votes $\hat{y} = \text{argmax}_y \sum_{i=1}^{m} \delta(y, \hat{y}^{(i)})$
- Soft voting: predict the class with the largest sum of predicted probability. Assumes base classifier outputs a confidence/probability $\hat{y} = \text{argmax}_y \sum_{i=1}^{m} f_y^{(i)}(x)$
- Mean: predict the mean of the base model outputs $\hat{y} = \sum_{i=1}^{m} f^{(i)}(x)$

Random Forests use decision trees as the base models.
Even if $\mathbb{D}^{(i)} \neq \mathbb{D}^{(j)}$, the learned DTs can be too much correlated. Introduce additional stochasticity in the DT training process. At each node, choose the best feature to split from a random selection of the set of features.

STACKING

Train m base models $f^{(i)}$ on $\mathbb{D}$ with m different learning algorithms $\mathcal{A}^{(i)}$

$$f^{(i)} = \mathcal{A}^{(i)}(\mathbb{D})$$

Use a meta-learner to learn to combine base models

$$g = \mathcal{A}_{\text{META}}([f^{(1)}, \ldots, f^{(m)}], \mathbb{D}')$$

Use the meta-model to output ensemble predictions

$$\hat{y} = g([f^{(1)}(x), \ldots, f^{(m)}(x)])$$

BOOSTING

Take a learner $\mathcal{A}$ and train it on $\mathbb{D}$.
Reweight examples in $\mathbb{D}$ based on their accuracy according to the trained model (harder examples get larger weights).
Train $\mathcal{A}$ again on the reweighted dataset.
Repeat the procedure m times.
Combine the learned models into the final model.

A weak learner learns models with accuracy slightly better than random. A weak learner is easy to implement and train. Applying boosting with weak learners as the base learning algorithm allows to turn them into strong learners.

CH.18 UNSUPERVISED LEARNING

We don't have any labels, only data.

CLUSTERING

Decide a priori the number of clusters. Each cluster is represented by its mean μ_i. The idea is to assign example to the clusters with the closer mean. CLUSTER SAMPLE MEAN: $\mu_i = \frac{1}{n_i} \sum_{x \in D_i} x$

K-MEANS:
(1) Initialize cluster mean $\mu_1, \dots \mu_n$ (random)
(2) Iterate until no mean changes:
   a. assign each example to cluster with nearest mean
   b. update cluster means according to assigned examples.

Distance Metrics → To compute the nearest mean
(1) Standard euclidean distance in $\mathbb{R}^d$: $d(x,x') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$
(2) Generic Minkowski metric for $p \geq 1$: $d(x,x') = (\sum_{i=1}^{d} |x_i - x'_i|^p)^{1/p}$
(3) Cosine similarity: $s(x,x') = \frac{x^T x'}{\|x\| \, \|x'\|}$ cosine of the angle between 2 vectors

Quality of clustering a criteria for defining a quality of clusters is **Sum-of-squared error criterion**. This method tells how bad the approximation of the cluster is using the means.
For each cluster, we take the mean μ_i within each cluster. I compare all examples in the cluster with the mean of the cluster they are in. The error will be computed as the squared error: $\mu_i = 1/n_i \sum_{x \in D_i} x$
The sum-of-squared errors is defined as the sum over the totality of clusters:

$$E = \sum_{i=1}^{k} \sum_{x \in D_i} \|x - \mu_i\|^2$$

k: number of clusters (?)

GAUSSIAN MIXTURE MODEL (GMM)

Each cluster is represented as a gaussian distribution. Need to estimate the mean and possibly the variance of the Gaussian distribution. PAG. 233 x figure
Parameter estimation ML estimation cannot be applied as cluster assignment of examples is unknown. Expectation-Maximization approach:
(1) Compute expected cluster assignment given current parameter setting.
(2) Estimate parameters given cluster assignment
(3) Iterate
Example: estimating means of k univariate Gaussians

SETTING
- A dataset of $x_1, \dots, x_n$ examples is observed
- For each example x_i, cluster assignment is modelled as $z_{i1}, \dots, z_{ik}$ binary latent variables. $z_{ij} = 1$ if Gaussian j generated, 0 otherwise.
- $z_{ij} = 1$ if Gaussian j generated x_i, 0 otherwise.
- Parameters to be estimate are the $\mu_1, \dots, \mu_k$ Gaussians means.
- ALL Gaussians are assumed to have the same variance σ^2

ALGORITHM
(1) Initialize $h = \langle \mu_1, \dots, \mu_k \rangle$ h = hypotesis → unknown, must be estimate
(2) Iterate until difference in ML is below a certain treshold:
   E-STEP: calculate expected value $E[z_{ij}]$ of each latent variable assuming current hypotesis $h = \langle \mu_1, \dots, \mu_k \rangle$ holds.
   M-STEP: calculate a new ML hypotesis $h' = \langle \mu'_1, \dots, \mu'_k \rangle$ assuming values of latent variables are their expected values just computed. Replace $h \leftarrow h'$

Probability that x_i is generated by Gaussian j assuming hypotesis h

$$\text{E-STEP: } E[z_{ij}] = \frac{p(x_i | \mu_j)}{\sum_{i=1}^{k} p(x_i | \mu_i)}$$

Weighted sample mean

$$\text{M-STEP: } \mu'_j = \frac{\sum_{i=1}^{n} E[z_{ij}] x_i}{\sum_{i=1}^{n} E[z_{ij}]}$$

We are given a dataset made of an observed part X and an unobserved part Z. We wish to estimate the hypotesis maximizing the expected log-likelihood for the data, with expectation taken over unobserved data:

$$h^* = \text{argmax}_h \; E_Z[\ln P(X, Z|h)]$$

The unobserved data Z should be treated as random variables governed by the distribution depending on X and h.

Generic Algorithm

(1) Initialize hypothesis h

(2) Iterate until converge:

E-STEP: compute the expected likelihood of an hypothesis h' for the full data, where the unobserved data distribution is modelled according to the current hypothesis h and the obs. data.

$$Q(h', h) = E_Z[\ln P(X, Z|h') \mid h, X]$$

M-STEP: replace the current hypothesis with the one maximizing $Q(h', h)$:

$$h \leftarrow \text{argmax}_{h'} \; Q(h', h)$$

HOW TO CHOOSE THE NUMBER OF CLUSTERS?

ELBOW METHOD: Increasing number of clusters allows for better modeling of data. Needs to trade-off quality of clusters with quantity. Stop increasing number of clusters when advantage is limited. → can be ambiguous

AVERAGE SILHOUETTE METHOD: use quality metric that trade-offs intra-cluster similarity and inter-cluster dissimilarity.

Compute average dissimilarity between i and its cluster

$$a_i = d(i, C) = \frac{1}{|C|} \sum_{j \in C} d(i, j)$$

Compute average dissimilarity between i and each cluster $C' \neq C$, take the minimum:

$$b_i = \min_{C' \neq C} d(i, C')$$

The silhouette coefficient is:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Typically only 1 maximum

HIERARCHICAL CLUSTERING

Clustering doesn't need to be flat. Natural grouping of data is often hierarchical. A hierarchy of clusters can be built on examples:

TOP-DOWN → recursively split clusters into subclusters

BOTTOM-UP → recursively aggregate pairs of clusters

ALGORITHM: AGGLOMERATIVE HIERARCHICAL CLUSTERING (Bottom-up)

(1) INITIALIZE: final number cluster K
initial cluster number $\hat{K} = n$
initial clusters $D_i = \{x_j\}$ $j \in [1, n]$

(2) WHILE $\hat{K} > K$: Find pairwise nearest clusters D_i, D_j
merge D_i and D_j
update $\hat{K} = \hat{K} - 1$

STOPPING CRITERION can be threshold on pairwise similarity.

ALGORITHM STEPWISE OPTIMAL HIERARCHICAL CLUSTERING

(1) INITIALIZE: final cluster number K
initial cluster number $\hat{K} = n$
initial cluster $D_i = \{x_i\}$ $i \in [1, n]$

(2) WHILE $\hat{K} > K$: find best clusters D_i, D_j to merge according to evaluation criteria
merge D_i and D_j
update $\hat{K} = \hat{K} - 1$

* seguite le direzioni più promettenti, basandosi sulla conoscenza attuale

CH.19 REINFORCEMENT LEARNING

The learner is provided a set of possible states S, and for each state, a set of possible actions, A moving it to a next state.
In performing action a from state s, the learner is provided an immediate reward $r(s,a)$. The task is to learn a policy allowing to choose for each state s the action a maximizing the overall reward.
The learner has to deal with problems of delayed reward coming from future moves, and trade-off between exploitation* and exploration.
Typical applications include moving policies for robots and sequential scheduling problems in general.
Sequential Decision Making: an agent needs to take a sequence of decisions. The agent should maximize some utility function. There is uncertainty in the result of a decision.

Markov Decision Process (MDP)

A set of states S in which the agent can be at each time instant.
A possibly empty set of terminal states $S_G \subset S$
A set of actions A the agent can make
A transition model providing the probability of going to a state s' with action a from state s: $P(s'|s,a)$ $s,s' \in S$, $a \in A$ P is the model
A reward $R(s,a,s')$ for making action a in state s and reaching state s'.
→ The probability $P(s'|s,a)$ don't depends on past actions or states, only on s

DEFINING UTILITIES

An environment history is a sequence of states. Utility (U): function defining the total amount of reward of an environment history.
We assume stationary preferences. There are 2 ways to define utilities:

- ADDITIVE REWARDS: $U([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$
- DISCOUNTED REWARDS: $U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma R(s_2) + \dots$ $\gamma \in [0,1]$*

We consider rewards that only depend on the (destination) state.
* Lower discount factor γ, less important the future rewards

MDP: TAKING DECISIONS → optimal policy

A policy π is a full specification of what action to take at each state.
The expected utility of a policy is the utility of an environment history, taken in expectation over all possible histories generated with that policy. An optimal policy π^* is a policy maximizing expected utility.
For infinite horizons, optimal policies are stationary, i.e. they only depend on the current state.
Optimal policies varying r: utility is made with additive rewards.
↳ Determined by the moving reward r
r is the reward of non-terminal state.
[Arrows indicate the best action to take.]
[Star indicates all actions are equally optimal.]

MOVING EXPENSIVE → optimal policy is to reach any terminal state asap.
" CHEAP → optimal policy is avoiding the bad terminal state at all costs.
" GIVES POSITIVE REWARDS → optimal policy is to stay away of terminal states

Utility of states. The utility of a state given a policy π is

$$U^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \mid S_0 = s\right]$$

where S_t is the state reached after t steps using policy π starting from $S_0 = s$. The true utility of a state is its utility under an optimal policy

$$U(s) = U^{\pi^*}(s) = \max_{\pi} U^{\pi}(s)$$

Given the true utility, an optimal policy is as follows:

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s' \in S} P(s'|s,a)\, U(s')$$

- The utility of a state is its immediate reward plus the expected discounted utility of the next state, assuming that the agent chooses and optimal action.

Bellman Equation $U(s) = R(s) + \gamma * \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\, U(s')$

There is a Bellman equation for each state $s \in S$. Utilities of states are solutions of the set of Bellman equations.
The solutions to the set of Bellman equations are unique.
Directly solving the set of equations is hard.

Computing an optimal policy:

(1) Initialize $U_0(s)$ to zero for all s
(2) Repeat
  a. Do Bellman update for each state s: $U_{i+1}(s) \leftarrow R(s) + \gamma * \max_{a \in A} \sum_{s' \in S} P(s'|s,a) U_i(s')$
  b. $i \leftarrow i+1$
(3) until max utility difference below a threshold
(4) return U

The optimal policy can be set as: $\pi^*(s) = \operatorname{argmax}_{a \in A} \sum_{s' \in S} P(s'|s,a)\, U(s')$

POLICY ITERATION:

(1) Initialize π_0 randomly
(2) Repeat
  a. Policy evaluation $U_i(s) = R(s) + \gamma \sum P(s'|s, \pi_i(s))\, U_i(s')$ — $\pi_i(s)$: action that policy $\pi^{(i)}$ prescribes for state s (instead of maximizing w.r.t. all $a \in A$)
  b. Policy improvement $\pi_{i+1}(s) \leftarrow \operatorname{argmax}_{a \in A} \sum_{s' \in S} P(s'|s,a)\, U_i(s')$ — $\pi_i(s)$: action prescribed by policy π for s
  c. $i \leftarrow i+1$
(3) Until no policy improvement
(4) Return π

PARTIAL KNOWLEDGE

Value iteration and policy iteration assume perfect knowledge.
In most cases, some of these aspects are not known.
Reinforcement learning aims at learning policies by space exploration.
POLICY EVALUATION: Policy is given, environment is learned.
POLICY IMPROVEMENT: both policy and environment are learned.
ADAPTIVE DYNAMIC PROGRAMMING (ADP): Policy evaluation

LOOP
a. Initialize s
b. Repeat:
  - receive award r, set $R(s) = r$
  - choose next action $a \leftarrow \pi(s)$
  - take action a, reach step s'
  - update counts $N_{sa} \leftarrow N_{sa} + 1$; $N_{s'|sa} \leftarrow N_{s'|sa} + 1$
  - update transition model $P(s''|s,a) \leftarrow N_{s''|sa} / N_{sa}$
  - update utility estimate $U \leftarrow$ PolicyEvaluation(π, U, P, R, γ)
c. Until s is terminal

(N_{sa}: times a is taken in s; $N_{s'|sa}$: times a is taken in s, leading to s')

The algorithm performs ML estimation of transition probabilities.
Standard policy evaluation to update utility estimate.
Each step is expensive because runs Policy evaluation.

POLICY EVALUATION IN UNKNOWN ENVIRONMENT

TEMPORAL-DIFFERENCE (TD) POLICY EVALUATION: rationale

Avoid running policy evaluation at each iteration. Locally update utility.

If transition from s to s' is observed:

- if s' was always the successor of s, the utility of s should be $U(s) = R(s) + \gamma U(s')$
- The temporal-difference update rule updates the utility to get closer to that situation: $U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$ α is learning rate.

No need for a transition model for utility update. Each step is much faster than ADP. Takes longer to converge.

GREEDY AGENT:

PROBLEM

Policy learning requires combining learning the environment and learning the optimal policy for the environment. A simple option consists of replacing policy evaluation in ADP with optimal policy computation

$$U(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) U(s')$$

Greedy agents always follow the local optimal action → Lack of exploration.

↳ The knowledge of environment is incomplete → suboptimal policy.

LEARNING OPTIMAL POLICY

EXPLOITATION: consists in following promising directions given current knowledge.

EXPLORATION: consists in trying novel directions looking for better alternatives.

A reasonable trade-off should be used in defining the search scheme:

ε-greedy strategy: choose a random move with probability ε, be greedy otherwise.

Assign higher utility estimates to unexplored state-action pairs:

EXPLORATION FUNCTION: $U^+(s) = R(s) + \gamma \max_{a \in A} f\left(\sum_{s' \in S} P(s' \mid s, a) U^+(s'), N_{sa}\right)$

$f(u, n)$ trades off between greed (high values of u) and curiosity (high value of n). Idea is to give more weight to actions that the agent has not tried very often.

TD LEARNING → Learning utilities of actions

- TD policy evaluation can also be adapted to learn an optimal policy.
- If TD is used to learn a state utility function, it needs to estimate a transition model to derive a policy.
- TD can instead be applied to learn a state-action utility function $Q(s, a)$
- The optimal policy corresponds to: $\pi^*(s) = \arg\max_{a \in A} Q(s, a)$

ON/OFF POLICY TD LEARNING

SARSA: on-policy TD learning

LOOP

1. Initialize $Q(s, a) = 0 \quad \forall s \in S, a \in A$
2. Repeat
   a. Receive award r
   b. Choose next action $a \leftarrow \pi^\varepsilon(s)$
   c. Take action a, reach step s'
   d. Choose action $a' \leftarrow \pi^\varepsilon(s')$
   e. Update local utility estimate $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$
3. Until s is terminal

π^ε is an ε-greedy policy based on Q.

In **Q-Learning** off-policy TD learning eliminate action a. and utility estimate is $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a))$

↳ Greedy policy's action

Sarsa vs Q-Learning

- Sarsa is on-policy: it updates Q using the current policy's action. Tends to converge faster, and are easier to use for linear function approximators.
- Q-Learning is off-policy: it updates Q using the greedy policy's action. This method is more flexible, it can even learn from traces generated with an unknown policy.

SCALING TO LARGE STATE SPACES

FUNCTION APPROXIMATION:

All techniques seen so far assume a tabular representation of utility functions.
Tabular representations don't scale to large state spaces.
The solution is to rely on function approximation: approximate $U(s)$ or $Q(s,a)$ with a parameterized function.
The function takes a state representation as input.
The function allows to generalize to unseen states.

TD learning: state utility

TD error: $E(s,s') = \frac{1}{2}(R(s) + \gamma U_\theta(s') - U_\theta(s))^2$ $\quad U_\theta \approx U_{\pi}$ utility approximation
Error gradient wrt function parameters: $\nabla_\theta E(s,s') = (R(s) + \gamma U_\theta(s') - U_\theta(s))(-\nabla_\theta U_\theta(s))$
Stochastic gradient update rule $\quad \theta = \theta - \alpha \nabla_\theta E(s,s')$

TD learning: action utility (Q-learning)

TD error: $E((s,a),s') = \frac{1}{2}(R(s) + \gamma \max_{a' \in A} Q_\theta(s',a') - Q_\theta(s,a))^2$ $\quad Q_\theta(s,a) \approx Q(s,a)$
Error gradient wrt funtion parameters: $\nabla_\theta E((s,a),s') = (R(s) + \gamma \max_{a' \in A} Q_\theta(s',a') - Q_\theta(s,a))(-\nabla_\theta Q(s,a))$
Stochastic gradient update rule $\quad \theta = \theta - \alpha \nabla_\theta E((s,a),s')$